Primjena liste i n-torke u programskom jeziku Python

Čumlievski, Nola

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
Ak. god. 2018./ 2019.

Nola Čumlievski

Primjena liste i n-torke u programskom jeziku Python

Završni rad

dr. sc. Ivan Dunđer

Zagreb, travanj 2019.

Izjava o akademskoj čestitosti

Izjavljujem i svojim potpisom potvrđujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

Sadržaj

1. Uvod

Nagli napredak tehnologije pridonio je povećanoj upotrebi programskih jezika, osobito Pythona. Python zauzima vodeće mjesto u području statistike, umjetne inteligencije, web programiranju i drugim područjima, a u najvećem broju slučajeva predstavlja prvi jezik koji se najčešće podučava i koristi na fakultetima. Sastavljen je kao programski jezik koji se lako uči i njegova sintaksa dizajnirana je da bude lako čitljiva i jednostavna, što čini Python idealnim nastavnim i početničkim programskim jezikom omogućavajući početnicima brzi napredak. Uzimajući u obzir njegovu laku čitljivost i široko područje primjene, ovaj rad bavi se opisom najčešće primijenjenih elementarnih struktura podataka u Pythonu – listama i n-torkama, načinom kreiranja i upotrebe navedenih struktura podataka te opisom određenih metoda i operacija koje se koriste nad listama i n-torkama prikazanih primjerima u Python 3 inačici. U ovom radu također je navedeno što je programski jezik Python i za što se točno koristi, te što su strukture podataka i slijedne kolekcije (ili zbirke) podataka te koja je njihova uloga u ovom programskom jeziku. Cilj ovog rada je pružiti osnovne informacije o programskom jeziku Python, dati definiciju struktura podataka i slijednih kolekcija te omogućiti detaljan uvid u liste i n-torke, tj. u strukture podataka koje možemo pronaći u većini programa napisanih programskim jezikom Python, a koje ujedno predstavljaju lako prilagodljive i korisne složene tipove podataka u Pythonu.

2. Python

Tri osnovne karakteristike programskog jezika Python navedene su u nastavku.

Python je **objektu orijentiran programski jezik**, što znači da se program napisan u ovom tipu programskog jezika temelji na objektima koji kombiniraju podatke i funkcionalnost. Osnovna je zamisao objektu usmjerenog programiranja spajanje strukture podataka i funkcija koje nad njima djeluju u jedan **objekt** (engl. *object*) (Budin et al., 2012:24). Objektu orijentirano programiranje omogućava korisnicima **fokus na podatke njihovog interesa** (npr. informacije o radnicima, rezultati nekog znanstvenog istraživanja ili ankete itd.) te razvoj metoda za efikasno upravljanje određenim podatcima.

Osnovni koncept objektu orijentiranog programiranja jest mogućnost **definiranja objekta** koji sadrži određene podatke te sve informacije koje su programu potrebne kako bi upravljao određenim podatcima. Ovim načinom pri pozivanju metoda nad podatcima nije potrebno određivanje detalja o podatcima jer podatkovni objekt „zna“ sve o sebi[1].

„***Operator overloading***“ je također jedna od pogodnosti objektnog programiranja. Ona predstavlja mogućnost različitog značenja jednog operatora, ovisno nad kojem se tipu podatka koristi (slika 1). U Pythonu, pri korištenju numeričkih vrijednosti, operator „+“ ima svoje uobičajeno značenje zbrajanja, ali pri korištenju stringova (niza znakova), operator „+“ sugerira spajanje tih dvaju nizova[2]. Objektu orijentirani programski jezici nam ujedno pružaju i dosljedan način korištenja objekta i kreiranje novog objekta na temelju neke klase.

```
================================
=========
>>> 12+34
46
>>> "pri" + "mjer"
'primjer'
>>>
```

Slika 1. Primjer različitog značenja operatora „+“

[1] Chun, W. *Core Python Programming*, 2006: 5.
[2] Spector P. *Introduction to Python Programming Course Notes* [online], 2005:12-13. Dostupno na: https://www.stat.berkeley.edu/~spector/python.pdf [03.04.2019].

Python je programski jezik **visoke razine**, što znači da umjesto da se bavi registrima, memorijskim adresama i sličnim tehničkim rješenjima, postupcima i procesima vezanim uz funkcioniranje samog stroja (računala), Python se bavi varijablama, objektima, kompleksnim aritmetičkim ili logičkim (engl. *Boolean*) operatorima, funkcijama, petljama i ostalim računalnim konceptima s fokusom na uporabljivost programskog jezika[3]. Izrazi, funkcije, klase, moduli te paketi koji su u sastavu programskog jezika Python omogućavaju korisniku izradu strukturiranih i dobro organiziranih aplikacija, što je važno kod čitanja i izmjenjivanja koda aplikacije[4].

Python je **interpretirani** programski jezik, što znači da se programski kod izvršava pokretanjem programa (engl. *software*) poznatog kao interpreter, a za interpretaciju koda nije potrebna izvršna datoteka (Hetland, 2017:1). „Interpreter" je zapravo program koji pokreće skripte pisane u interpretiranom programskom jeziku kao što je Python, te koji prima napisanu naredbu u kodu, vrši evaluaciju naredbe te izvještava korisnika o rezultatu naredbe[5].

Python se može koristiti **interaktivno** (slika 2) i **skriptno** (slika 3). Skriptni način rada omogućuje korisniku pisanje programskog koda, njegovu izmjenu i trajno pohranjivanje programa, dok se u interaktivnom načinu rada za svaki izraz ili naredbu vraća odgovarajući rezultat, što pomaže u provjeri sintakse ili točnosti izraza određene naredbe (Dunđer, 2018:7)

Također, program napisan u programskom jeziku Python može se pokrenuti na Windows operacijskom sustavu te na sustavima nalik Unixu kao što su Linux, Mac OS X i BSD (*Berkeley Software Distribution*)[6]. Lako čitljiva sintaksa i dinamičan zapis čine Python idealnim programskim jezikom za programiranje i razvijanje aplikacija u mnogim područjima kao što je navedeno u nastavku.

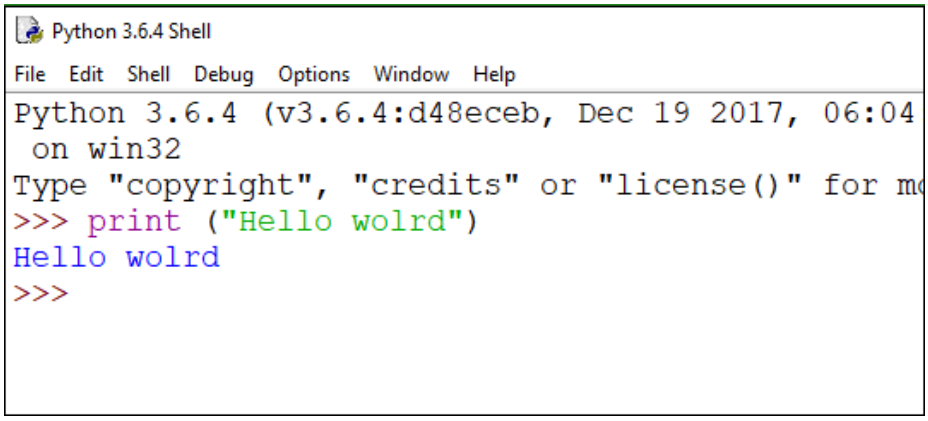
Sustavno/sistemsko programiranje – pisanje prijenosnih, lako održivih, administracijskih i uslužnih alata za određene operacijske sustave (engl. *shell tools*). Programi napisani u Pythonu mogu pretraživati datoteke i direktorije, pokretati druge programe i slično (Lutz, 2013:11).

[3] Spector P. *Introduction to Python Programming Course Notes* [online], 2005: 7. Dostupno na: https://www.stat.berkeley.edu/~spector/python.pdf [03.04.2019].
[4] Kuhlman, D. *A Python Book: Beginning Python, Advanced Python, and Python exercises* [online], 2013: 13. Dostupno na: http://www.davekuhlman.org/python_book_01.pdf [03.04.2019].
[5] Halterman, R. L., *Learning to program with Python* [online], 2011: 4. Dostupno na: https://www.cs.uky.edu/~keen/115/Haltermanpythonbook.pdf [03.04.2019.]
[6] Beazley, D.M. *Python Essential Reference*, 2009: 1-2.

Programiranje grafičkog korisničkog sučelja (GUI) – GUI je način interakcije čovjeka s računalom kroz manipulaciju grafičkim elementima uz pomoć tekstualnih poruka i obavijesti (Lutz, 2013:11).

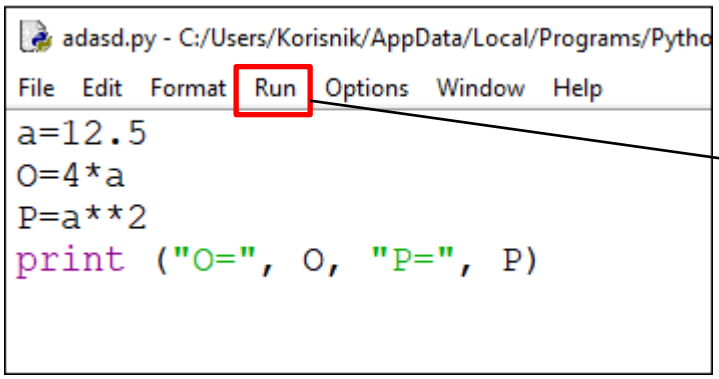
Web programiranje – parsiranje (ili parsanje) i generacija XML i JSON dokumenata; slanje, primanje i parsiranje e-mailova; dohvaćanje web stranica pomoću URL-a; parsiranje HTML-a dohvaćenih web stranica itd[7].

Python također ima veliku ulogu u programiranju igara, rudarenju (tj. dubinskom pretraživanju i analizi) podataka, robotici, umjetnoj inteligenciji, analizi prirodnog jezika, programiranju mobilnih sustava, vizualizaciji podataka, strojnom učenju i mnogim drugim područjima. Uz široku bazu korištenja među pojedincima, Python se također koristi od strane opće poznatih organizacija kao što su: Google, YouTube, Dropbox, BitTorrent, Google's App Engine, Maya, NSA, iRobot, Netflix, Intel, Cisco, Qualcomm, IBM, NASA, Facebook, Instagram, Spotify, SurveyMonkey itd[8].



Slika 2. Interaktivni način rada

[7] Lutz, M. *Learning Python*, 2013: 10-14.
[8] Lutz, M. *Learning Python*, 2013: 9-10.



Služi za pokretanje programa

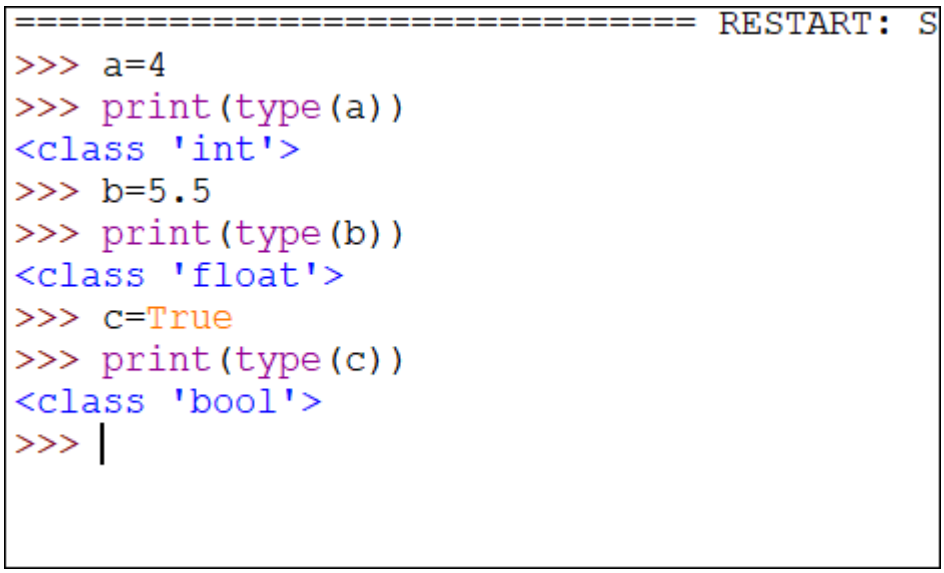
Slika 3. Skriptni način rada

2. Strukture podataka i slijedne kolekcije (zbirke) podataka

Pri rješavanju problema u programskom jeziku Python često se pojavljuje potreba za organiziranjem grupe srodnih podataka. Za takvo organiziranje podataka u Pythonu su uvedene strukture podataka (Necaise, 2011:13). **Strukture podataka** predstavljaju način organiziranja i pohrane podataka pomoću kojih se može pristupiti određenim podatcima, što znači da se strukture podataka koriste za pohranu kolekcije ili zbirke podataka (engl. *collections*) povezanih/srodnih podataka. Također, strukture podataka definiraju vezu između podataka te operacije koje se mogu koristiti nad određenim podatcima (Kuhlman, 2013:21). Programski jezici obično sadrže određene strukture podataka koje predstavljaju dio određenog jezika – **elementarne strukture podataka**[9]. Pod pojmom „elementarna struktura podataka" u Pythonu podrazumijevamo tip objekta koji kao „kontejner" unutar sebe može pohraniti različite vrijednosti. Ukoliko su navedene vrijednosti različitog tipa (npr. broj i tekst) radi se o **heterogenoj elementarnoj strukturi podataka**, a ako su elementi strukture podataka istog tipa (npr. samo brojevi) radi se o **homogenoj elementarnoj strukturi podataka** (Dunđer, 2018:69)**.** Elementarne strukture podataka dijele se u dvije kategorije: **jednostavne** i **složene**.

Jednostavne strukture podataka – poznatije i kao tipovi podataka, sastoje se od jedne vrijednosti zapisane u najosnovnijem obliku, a koje se ne mogu rastaviti na manje dijelove (Summerfield, 2009:107). Kao primjer jednostavnih struktura podataka mogu se izdvojiti integeri (cijeli brojevi), realni brojevi i logički tip podataka, a koji se sastoje od samo jedne numeričke, odnosno logičke vrijednosti[10].

[9] Dunđer, I. *Programiranje,* 2018: 68-69.
[10] Summerfield, M. *Programming in Python 3, A Complete Introduction to the Python Language*, 2009: 107-108.



Slika 4. Primjer jednostavnih struktura podataka

U gore navedenom primjeru (slika 4) prikazane su vrste jednostavnih struktura podataka: varijabla „a“ pripada tipu podataka „integer“ (cijeli brojevi), varijabla „b“, koja pripada tipu podataka „float“ (decimalni brojevi) te varijabla „c“ koja pripada tipu podataka „bool“ (logičke vrijednosti – True, False).

S druge strane postoje elementarne strukture podataka koje se sastoje od više sastavnih dijelova (Summerfield, 2009:108) – to su tzv. složeni tipovi podataka koji mogu sadržavati više različitih tipova i struktura podataka.

U Pythonu stringovi, liste, rječnici, skupovi i n-torke, s obzirom da unutar sebe mogu sadržavati više različitih i raznotipskih vrijednosti, su primjeri složenih tipova podataka[11].

[11] Summerfield, M. *Programming in Python 3, A Complete Introduction to the Python Language*, 2009: 107-108.

```
=========
>>> d=[1,2,4, "primjer"]
>>> print(type(d))
<class 'list'>
>>> e={"a":1, "b":2, "c":3}
>>> print(type(e))
<class 'dict'>
>>> f=("a", "b", "c")
>>> print(type(f))
<class 'tuple'>
>>>
```

Slika 5. Primjer složenih tipova podataka

U ovom primjeru (slika 5) prikazane su vrste složenih tipova podataka; varijabla „d“ koja pripada strukturi podataka klase „list“ (lista), varijabla „e“ koja pripada strukturi podataka klase „dict“ (rječnik) te varijablu „f“ koja pripada strukturi podataka klase „tuple“ (n-torka).

U sljedećem dijelu teksta opisani su određeni termini, često korišteni pri opisivanju struktura podataka. **Kolekcije** ili **zbirke podataka** su grupe vrijednosti bez implicirane organizacije ili veze između individualnih vrijednosti (Necaise, 2011:6). **Kontejner** (engl. *container*) predstavlja bilo koju strukturu podataka ili apstraktne tipove podataka (definirani tipovi podataka koji specificiraju skup podatkovnih vrijednosti i kolekciju definiranih operacija koje se mogu vršiti nad određenih vrijednostima) koji pohranjuju i organiziraju kolekciju (Necaise, 2011: 6). Individualne vrijednosti kolekcije poznate su kao **elementi** kontejnera, a kontejner koji ne sadrži elemente smatra se **praznim**[12]. Python pruža određen broj ugrađenih kontejnera koji uključuju stringove (nizove znakova), liste, n-torke, skupove i rječnike. **Nizovi** (engl. *sequence*) predstavljaju kontejnere u kojima su elementi organizirani u linearnom poretku gdje se svakom elementu može pristupiti pomoću njegove pozicije (indeksa). Python pruža dva nepromjenjiva niza**,** stringove i n-torke te dva promjenjiva niza, listu i rječnik[13].

Liste i n-torke svrstavamo u **slijedne kolekcije** jer je njihovim elementima moguće pristupiti rednim brojem, tj. indeksom (Stojanović, 2012:104). Redoslijed obilaska elemenata slijednih

[12] Necaise, R. D. *Dana Structures and Algorithms Using Python,* [online], 2011: 6. Dostupno na: http://home.ustc.edu.cn/~huang83/ds/Data%20Structures%20and%20Algorithms%20Using%20Python.pdf [05.04.2019].
[13] Necaise, R. D. *Dana Structures and Algorithms Using Python,* [online], 2011: 6-7. Dostupno na: http://home.ustc.edu.cn/~huang83/ds/Data%20Structures%20and%20Algorithms%20Using%20Python.pdf [05.04.2019].

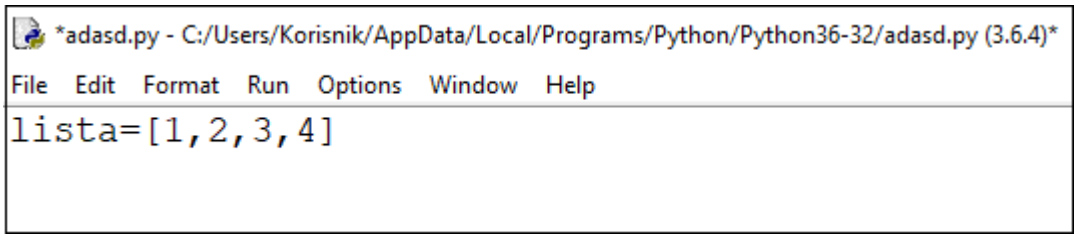
kolekcija određen je indeksima: prvo se obilazi nulti element, zatim prvi, drugi itd., sve do kraja kolekcije (Stojanović, 2012:104). Glavne prednosti slijednih kolekcija su njihova jednostavnost i iznimno brz pristup elementima kolekcije putem rednog broja. Brz dohvat s proizvoljnim indeksom elementa iz slijedne kolekcije moguć je zahvaljujući internoj organizaciji slijednih kolekcija[14]. Naime, reference na elemente u memorijskom spremniku kolekcije (memorijske adrese na kojoj se element nalazi) složeni su u **kontinuiranom slijedu** (Stojanović, 2012:104). Pri dohvaćanju elementa sa željenim rednim brojem, interno se redni broj pomnoži s duljinom memorijske adrese i na to se doda adresa početka niza reference kako bi se dobila adresa na kojoj se nalazi referenca za traženi element. Iako naizgled kompliciran, navedena operacija je vrlo brza jer zahtijeva samo po jedno cjelobrojno množenje i zbrajanje[15].

Formalno, sve objekte koji definiraju operaciju indeksiranja, tj. omogućavaju pristup vrijednostima preko cjelobrojnog indeksa, nazivamo slijednim tipovima ili nizovima (Stojanović, 2012:105). Primjeri strukture podataka sa slijednim smještanjem su: n-torke, stringovi, liste, rječnici (od inačice Pythona 3.6) te nizovi bajtova, a postoje i strukture podataka sa raspršenim slijedom elemenata, kao što su skupovi i zamrznuti skupovi.

[14] Stojanović, A. *Elementi računalnih programa: s primjerima u Pythonu i Scali,* 2012:104.
[15] Stojanović, A. *Elementi računalnih programa: s primjerima u Pythonu i Scali,* 2012:104-105.

3. Liste

Lista (engl. *list*) je struktura podataka uređenog niza koja se sastoji od 0 ili više elemenata ili vrijednosti (Summerfield, 2009:113). Predstavlja jednu od temeljnih slijednih kolekcija u Pythonu koje se koriste za indeksiranje popisa elemenata. Liste i n-torke imaju mnogo zajedničkih karakteristika: mogu sadržavati objekte različitih tipova, može se po njima iterirati i mogu se indeksirati[16]. Međutim, za razliku od n-torki, liste su **izmjenjivi objekti** (engl. *mutable objects*), što znači da se elementi liste mogu mijenjati, brisati i da se mogu dodavati novi elementi, a da se pritom ne stvara novi objekt kolekcije (Stojanović, 2012:111). S obzirom da su takve operacije potrebne u različitim računskim operacijama, liste predstavljaju najčešće korištenu slijednu kolekciju u Pythonu.



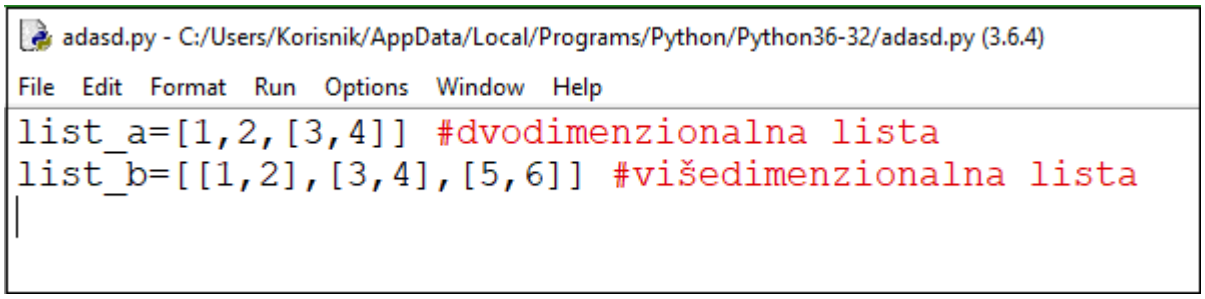
Slika 6. Primjer jednostavne liste

Na gornjoj slici (slika 6) prikazan je jedan od najosnovnijih primjera liste. Elementi (različiti ili u ovom primjeru jednakog tipa) koji se nalaze u strukturi unutar uglatih zagrada predstavljaju listu.

Liste se često koriste kada treba pohraniti podatke kojima ne treba pristupati nasumično (engl. *random*) te kada je potrebno na jednostavan način više puta pristupati pripadajućim podatcima (Dunđer, 2018:69). U liste se smještaju podatci koji su na neki način srodni kako bi se lakše i jednostavnije s njima postupalo. Elementi koji se mogu nalaziti unutar liste mogu biti podatci bilo kojeg tipa (tekst, brojevi, decimalni brojevi, logičke vrijednosti itd.) kao i elementi i tipovi podataka definirani od strane korisnika, uključujući i kolekcije kao što su liste i n-torke[17]. Kada sadržaj jedne liste čine druge (pod)liste, tada se radi o tzv.

[16] Budin L. et al. *Rješavanje problema programiranjem u Pythonu:udžbenik za prirodoslovno-matematičke gimnazije*, 2015: 183-184.
[17] Sturtz, J. (2018) *Lists and Tuples in Python* [online]. Dostupno na: https://realpython.com/python-lists-tuples/ [06.04.2019].

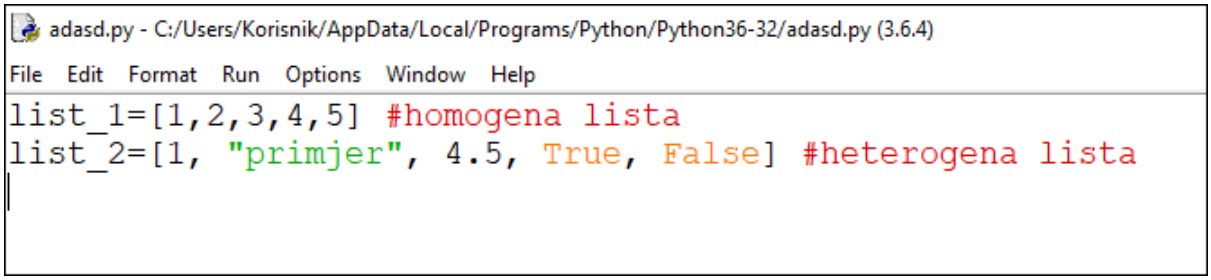
ugniježđenim listama (engl. *nested lists*) koje se često koriste za prikaz (dvodimenzionalnih ili višedimenzionalnih) matrica (Dunđer, 2018:70).



Slika 7. Primjer ugniježđene liste

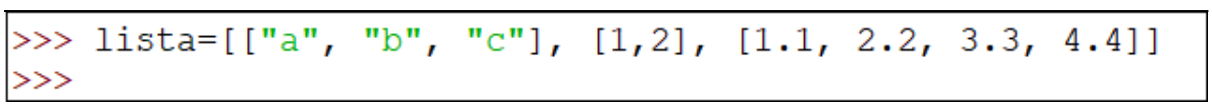
Ovim primjerom (slika 7) je prikazana razlika između dvije različite vrste ugniježđenih lista: dvodimenzionalne, gdje imamo samo jednu listu unutar liste i višedimenzionalne, gdje imamo više lista unutar jedne liste.

Također, svi elementi liste mogu biti istog tipa, što tada predstavlja **homogenu listu**, a mogu biti i različitog tipa, što čini **heterogenu listu** (slika 8).



Slika 8. Primjer homogene i heterogene liste

Lista koja sadrži podliste koje nisu jednake duljine naziva se **raznovrsna lista** (engl. *jagged list*) (Dunđer, 2018:72). Primjer jedne takve liste dan je u nastavku (slika 9).



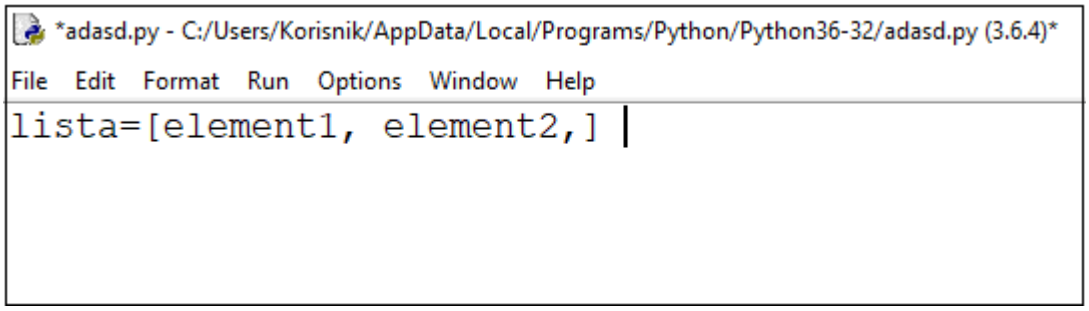
Slika 9. Primjer raznovrsne liste

Liste mogu biti nadopunjene, sortirane, ispražnjene i preokrenute, tj. obrnute (engl. *reversed*). Također je moguće smanjivanje i povećavanje liste, kao i razdvajanje i spajanje jedne liste s drugim listama. Jedan ili više elemenata mogu se dodavati, ažurirati ili ukloniti prema volji korisnika (Chun, 2006:171). Uspoređujući liste u Pythonu s nizovima u drugim programskim jezicima, gdje veličina poretka mora biti unaprijed poznata i svi elementi moraju biti istog tipa, u Pythonu elementi liste mogu sadržavati različite tipove podataka i mogu se dinamički (tijekom uporabe) povećavati ili smanjivati[18].

[18] Chun, W. *Core Python Programming*, 2006: 171.

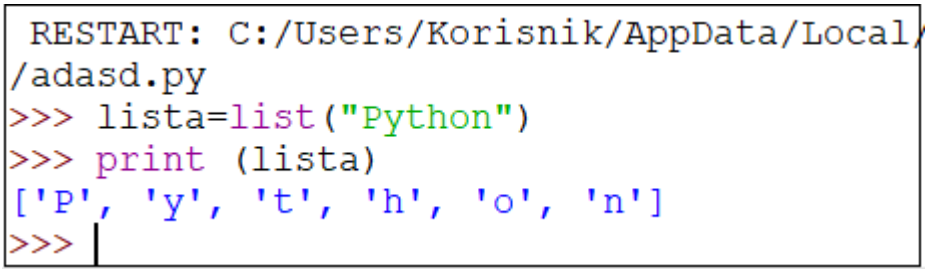
3.1. Kreiranje liste

Liste se mogu stvarati, tj. kreirati na dva različita načina. Prvi je način kroz **pridruživanje** (statički način), a drugi je način kroz **obuhvaćanje liste** (engl. *list comprehension*) (aktivni način) (Budin et al., 2015:183). **Pridruživanje** je najjednostavniji način stvaranja liste (slika 10), vrši se nabrajanjem elemenata liste odvojenih zarezom unutar uglatih zagrada (Budin et al., 2015:183). Prazna lista stvara se pomoću praznog para uglatih zagrada ([ ]). Najčešće nakon zadnjeg elementa liste nema zareza, međutim, dopušteno je navesti zarez i nakon zadnjeg elementa, što je u potpunosti sintaksno ispravno (Dunđer, 2018: 73).



Slika 10. Najjednostavniji način stvaranja liste

Liste se također mogu stvarati pomoću funkcije „list()“, pri čemu će početni sadržaj liste biti određen elementima prebrojivih argumenata funkciji (Kalafatić et al., 2016:221). U dolje navedenom primjeru (slika 11) prikazuje se stvaranje liste pomoću navedene funkcije, gdje rezultat (lista) sadrži 6 elemenata od kojih svaki predstavlja jedan znak (slovo) zadane riječi u obliku stringa.



Slika 11. Stvaranje liste konstruktorom

S druge strane, Python predviđa i mogućnost stvaranja lista transformiranjem nekog drugog prebrojivog objekta uz pomoć tzv. obuhvaćajućih izraza za tvorbu liste, što se naziva **obuhvaćanje liste** (engl. *list comprehension)*[19]. Obuhvaćanje liste podrazumijeva proces stvaranja liste gdje je svaki element liste novi rezultat nastao provođenjem operacije nad svakim članom drugog niza ili rezultat (pod)nizova postojećih elemenata koji zadovoljavaju određeni uvjet (Rossum, 2016:37). Generalna sintaksa obuhvaćanja liste jest: **[izraz for vrijednost in element if uvjet]**.

Dolje navedeni primjer (slika 12) prikazuje nastajanje liste uz pomoć određenog izraza gdje su elementi nove liste elementi već postojeće određene liste koji zadovoljavaju određeni uvjet. Izraz [x for x in lista if x[0]==“i“] jasno izražava da je potrebno kreirati novu listu pod nazivom „rijeci“, od elemenata označenih slovom „x“ koji su sadržani u kreiranoj listi „lista“, ali za koje vrijedi x[0]==“i“, što znači da tražimo elemente liste čije je prvo slovo „i“, tj. čija je vrijednost na nultom indeksu „i“. Lista „rijeci“ se zapravo gradi obuhvaćajući određene elemente liste „lista“.

```
>>> lista=["informacija", "semestar", "ispit", "rokovi", "ocjena"]
>>> rijeci=[x for x in lista if x[0]=="i"]
>>> print(rijeci)
['informacija', 'ispit']
>>>
```

Slika 12. Primjer izraza za tvorbu liste s određenim elementima postojeće liste

Na sljedećem primjeru (slika 13) prikazano je nastajanje nove liste uz provođenje određene operacije/funkcije nad postojećom listom kako bi se dobila nova lista s novim elementima. Kreirana je prazna lista „brojevi“, u koju su „for“ petljom i „range()“ funkcijom svrstani brojevi od 0-4 (bez petice), te se navodi da se u postojeću praznu listu metodom „append()“ nadodaju brojevi manji za 1 od brojeva pridodanih „range()“ funkcijom (tj. 0, 1, 2, 3, 4). Kao rezultat Python vraća listu s brojevima [-1, 0, 1, 2, 3] u kojoj su svi brojevi u odnosu na prvu listu umanjeni za 1.

[19] Stojanović, A. *Elementi računalnih programa: s primjerima u Pythonu i Scali*, 2012: 114-115.

```
=========
>>> brojevi=[]
>>> for x in range(5):
        brojevi.append(x-1)


>>> brojevi
[-1, 0, 1, 2, 3]
>>>
```

Slika 13. Primjer operacije za tvorbu liste s novim elementima

Listu također možemo kreirati pomoću određivanja veličine s početnom vrijednosti za svaki element. Ovaj način (slika 14) funkcionira za svaki (složeni) tip podataka (Hünniger, 2015:47).

```
>>> lista=[1]*4
>>> print(lista)
[1, 1, 1, 1]
>>> lista_1=["abc"]*3
>>> print(lista_1)
['abc', 'abc', 'abc']
>>>
```

Slika 14. Stvaranje liste iniciranjem veličine

Međutim, ako stvaramo listu nadovezivanjem Python kopira svaki objekt po referenci. Ovo predstavlja problem kod izmjenjivih podataka, ako imamo npr. višedimenzionalnu listu gdje je svaki pojedini element podlista. Lista se može stvoriti, ali u slučaju mijenjanja podataka javlja se problem[20].

[20] Hünniger D., ur. *Python Programming* [online], 2015: 40-41. Dostupno na: https://upload.wikimedia.org/wikipedia/commons/9/91/Python_Programming.pdf [07.04.2019]

```
>>> lista=[1]*3
>>> lista_2=[lista]*3
>>> print(lista_2)
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
>>> lista_2[0][2]=2
>>> print(lista_2)
[[1, 1, 2], [1, 1, 2], [1, 1, 2]]
>>>
```

Slika 15. Problem kod višedimenzionalne liste stvorene nadovezivanjem

Na gore navedenom primjeru (slika 15) prikazano je kako Python koristi istu referencu za elemente unutarnje, tj. ugniježđene liste kao i za elemente vanjske liste. Naime, pri pokušaju promjene treće brojke prve unutarnje liste („lista_2[0] [2]=2“), Python je promijenio treći element svih unutarnjih lista, jer koristi istu referencu za pristup listama. Ovaj problem moguće je izbjeći na način da se za stvaranje liste koriste metode obuhvaćanja liste.

3.2. Operacije nad listama

U nastavku su opisane operacije nad listama.

3.2.1. Indeksiranje liste

Lista predstavlja linearnu strukturu podataka u kojoj se svaki element može **dohvatiti ili mijenjati** pomoću njegovog indeksa, koji obilježava položaj određenog elementa u samoj strukturi. Prvi element u strukturi ima indeks jednak 0, a zadnji element ima indeks jednak ukupnom broju elemenata u listi umanjen za 1[21].

Sintaksa za pristup elementima liste sastoji se od operatora indeksiranja, tj. operatora uglatih zagrada (engl. *bracket operator*), a broj unutar uglatih zagrada specificira indeks. Lista zapravo predstavlja vezu između indeksa i elemenata, te se takva vrsta veze naziva **mapiranje**; svaki indeks mapiran je na jedan od elemenata liste[22]. Za indeks možemo upotrijebiti bilo koji cijeli broj (engl. *integer*), te ako je indeks negativne vrijednosti, vrijednosti se računaju s kraja liste (ili stringa i n-torke) prema početku kolekcije (Downey, 2012: 107). Također, vrijednost elementa na nekoj proizvoljnoj poziciji u listi može biti izmijenjena, što znači da nova vrijednost može biti pridružena bilo kojem elementu prema potrebi (Dunđer, 2018:77). Pri pokušaju pisanja ili čitanja elementa koji ne postoji, Python izbacuje „IndexError“ (Downey, 2012:107).

```
>>> lista= [5,6,7,8,9]
>>> print(lista[0])
5
>>> print(lista[3])
8
>>> lista[0]=1
>>> print(lista[0])
1
>>> print(lista[-2])
8
>>>
```

Slika 16. Primjer dohvaćanja i mijenjanja elemenata liste

[21] Necaise. R.D. *Data structures and algorithms using Python* [online], 2011:41. Dostupno na: http://home.ustc.edu.cn/~huang83/ds/Data%20Structures%20and%20Algorithms%20Using%20Python.pdf [07.04.2019]

[22] Downey, A.B. *Think Python*, 2012: 107.

U gore navedenom primjeru (slika 16) prikazano je dohvaćanje i mijenjanje elemenata. Nakon prve „print()“ naredbe, Python kao element s indeksom 0 izbacuje vrijednost „5“, međutim, nakon što smo pomoću indeksiranja promijenili nulti element u „1“ (lista [0]=1), Python vraća jedinicu kao prvi element liste. U zadnjem redu je prikazano dohvaćanje elemenata od kraja liste. Drugi element brojeći od kraja liste prema početku jest „8“.

```
>>> lista=[1,2,3,4]
>>> print(lista[4])
Traceback (most recent call last):
  File "<pyshell#65>", line 1, in <module>
    print(lista[4])
IndexError: list index out of range
>>>
```

Slika 17. Primjer: „IndexError“

U ovom primjeru (slika 17) prikazan je „IndexError“. Naime, kreirana lista sastoji se od 4 elementa s indeksima 0, 1, 2 i 3, i pokušavajući dohvatiti element s indeksom 4, Python vraća pogrešku (klasu iznimke) zbog pokušaja dohvaćanja elementa liste indeksom izvan granica (dosega) strukture.

3.2.2. Kopiranje liste

Postoje dva načina kopiranja liste: duboko i površno, tj. plitko kopiranje.

Dubokim kopiranjem jedna lista preuzima i memorijsku lokaciju (adresu) druge liste, što zapravo znači da ta dva objekta (liste) referenciraju 1 objekt u memoriji. Lista se duboko kopira korištenjem jednostrukog znaka jednakosti: „=“. Na dolje navedenom primjeru (slika 18) prikazano je da navedene dvije liste dijele istu memorijsku lokaciju nakon dubokog kopiranja. Funkcija „id()“, iskorištena u primjeru, vraća identitet objekta, tj. cijeli broj koji je jedinstven za navedeni objekt i koji predstavlja memorijsku adresu objekta.

```
>>> lista1=[1,2,3]
>>> lista2=lista1
>>> id(lista1)
93867504
>>> id(lista2)
93867504
```

Slika 18. Primjer dubokog kopiranja liste

Površnim, tj. plitkim kopiranjem nastaju dvije liste koje su jednakog sadržaja, ali svaka od njih ima jedinstvenu memorijsku adresu. Lista se površno kopira na isti način kao i kod dubokog kopiranja, uz iznimku postavljanja dvotočke (**:**) u uglate zagrade iza znaka jednakosti te imena liste. Nakon površnog kopiranja stvara se prava kopija liste gdje pojedini elementi liste pokazuju na iste izvorne vrijednosti, međutim, nakon što se pojedina vrijednost u jednoj listi izmijeni ta promjena se neće odraziti na drugu listi[23].

Na slici broj 19 prikazano je da, zbog postavljanja gore navedenog znaka ([**:**]) na kraju izraza jednakosti, dobivene liste imaju različitu memorijsku lokaciju, za razliku od duboko kopiranih lista.

```
>>> lista3=[4,5,6]
>>> lista4=lista3[:]
>>> id(lista3)
93867624
>>> id(lista4)
91727392
>>>
```

Slika 19. Površno kopiranje liste

3.2.3. Izrezivanje liste

Izrezivanje liste (engl. *list slicing*) predstavlja proces odvajanja dijela određene liste i stvaranje nove (pod)liste (Lutz, 2013: 243). Osnovno izrezivanje podrazumijeva indeksiranje liste pomoću dva cijela broja odvojena dvotočkom (Dunđer, 2018:109); dakle podlista se

[23] Budin L. et al. *Rješavanje problema programiranjem u Pythonu:udžbenik za prirodoslovno-matematičke gimnazije*, 2015: 192-193.

specificira pomoću cijelih brojeva i *slice* operatora (**:**)[24]. Komad liste (engl. *list slice*) jest izraz sintakse: **list[begin:end:step]** gdje je „**list“** varijabla koja se referira na objekt (listu), „**begin“** jest cijeli broj (integer) koji predstavlja početni indeks nove podliste i **„end“** jest integer koji je za jedan veći od indeksa zadnjeg elementa nove podliste. „**Step**“ je integer koji specificira veličinu koraka („skoka“) kojim se prolazi listom da bi se obuhvatile samo određene vrijednosti[25]. Veličina koraka od npr. 3 bi uključivala svaki treći element liste u navedenom rasponu. Ukoliko prvi broj u izrazu koji definira izrezivanje nije prikazan, kao prva vrijednost se uzima početak liste (element s indeksom 0), a ukoliko drugi broj nije naveden, kao druga vrijednost pri izrezivanju uzima se kraj liste (zadnji element)[26].

```
>>> lista=["a", "b", "c", "d", "e", "f"]
>>> lista[1:4]
['b', 'c', 'd']
>>> lista[5]
'f'
>>> lista[:3]
['a', 'b', 'c']
>>> lista[3:]
['d', 'e', 'f']
>>> lista[:]
['a', 'b', 'c', 'd', 'e', 'f']
>>> lista[::-1]
['f', 'e', 'd', 'c', 'b', 'a']
>>> lista[::-2]
['f', 'd', 'b']
```

Slika 20. Izrezivanje liste

U navedenom primjeru (slika 20), prvo izrezivanje ([1:4]) ispisuje elemente počevši s elementom indeksa 1 do elementa indeksa 3 (kod izrezivanja liste, element s indeksom drugog znaka se ne uključuje u listu). Drugo izrezivanje ([:3]) ispisuje sve elemente u listi koji se nalaze prije elementa s indeksom 3, a vrijedi i obrnuto, kod trećeg izrezivanja ([3:]), s obzirom da se dvotočka (engl. *slice operator*) nalazi s desne strane broja, Python ispisuje elemente počevši od elementa s indeksom 3 (uključujući element s trećim indeksom) do kraja

[24] Downey, A.B. *Think Python*, 2012: 108.
[25] Dunđer, I. *Programiranje*, 2018: 111.
[26] Dunđer, I. *Programiranje*, 2018: 110.

liste. Zadnja dva primjera izrezivanja liste pokazuju da, ako je integer na poziciji „step“ negativan, Python iterira (kreće se kroz listu) od kraja liste prema početku, kao kod indeksiranja. Kada je negativan indeks veći od -1 kao u primjeru (-2), tada Python ispisuje elemente preskačući određen broj elemenata ovisno o navedenom indeksu (u navedenom primjeru Python, budući da je indeks negativan (-2), ispisuje svaki drugi element liste).

Pomoću *slice operatora* također možemo promijeniti (ažurirati) više elemenata liste odjednom (Downey, 2012:108). Na primjeru u nastavku (slika 21) prikazana je izmjena 1. do 3. elementa liste.

```
>>> lista=[1,2,3,4,5,6]
>>> lista[0:3]=[4,5,6]
>>> print(lista)
[4, 5, 6, 4, 5, 6]
>>>
```

Slika 21. Mijenjanje elemenata pomoću *slice operatora*

3.2.4. Konkatenacija liste

Za kombiniranje sadržaja (elemenata) dviju lista koristi se znak plus (+) među navedenim listama (Spector, 2005:20). Kao rezultat proizlazi jedna lista čija je duljina jednaka zbroju duljina prethodno navedenih dviju lista i koja sadrži sve elemente dviju kombiniranih lista na način da su poredani najprije elementi prve liste koje zatim slijede elementi druge liste (slika 22). Konkatenacija se vrlo često koristi pri kombiniranju dviju lista[27].

```
>>> lista1=[1,2,3]
>>> lista2=["a", "b", "c"]
>>> lista1+lista2
[1, 2, 3, 'a', 'b', 'c']
>>>
```

Slika 22. Primjer konkatenacije liste

[27] Spector P. *Introduction to Python Programming Course Notes* [online], 2005: 19-21. Dostupno na: https://www.stat.berkeley.edu/~spector/python.pdf [06.04.2019].

3.2.5. Nadovezivanje liste

Znak zvjezdice (*) koristi se kao operator za množenje sadržaja liste (Spector, 2005:20). Rezultat primjene množenja na sadržaj liste je jedna lista s elementima izvorne liste ponovljenim onoliko puta koliko je specificirano množiteljem (slika 23). Također, pomoću nadovezivanja sadržaja liste mogu se stvoriti ugniježđene liste[28].

```
>>> 3*[1]
[1, 1, 1]
>>> lista=[1,2,3]
>>> 3*lista
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> 3*[lista]
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
>>>
```

Slika 23. Primjer nadovezivanja sadržaja liste

3.2.6. Usporedba lista

Pomoću **„in“ operatora** vrši se provjera da li određena vrijednost ili element pripada sadržaju liste (Halterman, 2011:201). S lijeve strane operatora pruža se određena vrijednost, a s desne strane se navodi lista čiji se sadržaj pretražuje. Kao rezultat provjere proizlazi 1 (True) ako se određena vrijednost nalazi u listi ili 0 (False) ako se vrijednost ne nalazi u listi (slika 24).

S druge strane postoji i **„not in“ operator**, koji provjerava da li određeni element nije sadržan u listi, gdje proizlazi suprotan rezultat: True ako se element ne nalazi u listi, i False ako se element nalazi u listi[29].

[28] Spector P. *Introduction to Python Programming Course Notes* [online], 2005: 21. Dostupno na: https://www.stat.berkeley.edu/~spector/python.pdf [03.04.2019].
[29] Lott, S. F. *A Programmer's Introduction to Python: Building Skills in Python* [online]. Dostupno na: http://www.itmaybeahack.com/book/python-2.6/html/index.html [08.04.2019.]

```
>>> lista=[1,2,3, "abc", "def", "ghi"]
>>> "abc" in lista
True
>>> 4 in lista
False
>>> "abc" not in lista
False
>>> 4 not in lista
True
>>>
```

Slika 24. Primjeri „in“ i „not in“ operatora

Ostali znakovi za usporedbu, tj. relacijski operatori (<, <=, >, >=, ==, !=) mogu se također koristiti nad listama (slika 25). Elementi liste se uspoređuju element po element. Ako su elementi koji se uspoređuju istog tipa, vrijede uobičajena pravila usporedbe određenih elemenata. Međutim, ako elementi nisu istog tipa, Python uspoređuje određene elemente prema imenu, budući da nema druge osnove na temelju koje bi mogao usporediti elemente[30].

Značenje znakova dano je u nastavku: < (manji), <= (manji od ili jednak), > (veći od), >= (veći od ili jednak), == (jednak), != (različit od).

[30] Lott, S. F. *A Programmer's Introduction to Python: Building Skills in Python* [online]. Dostupno na: http://www.itmaybeahack.com/book/python-2.6/html/index.html [08.04.2019.]

```
>>> lista1=[1,2,3]
>>> lista2=[1,2,3]
>>> lista3=[4,5,6]
>>> lista1==lista2
True
>>> lista1!=lista3
True
>>> lista1!=lista2
False
>>> lista1>lista2
False
>>> lista1<lista3
True
>>> lista1<=lista2
True
>>>
```

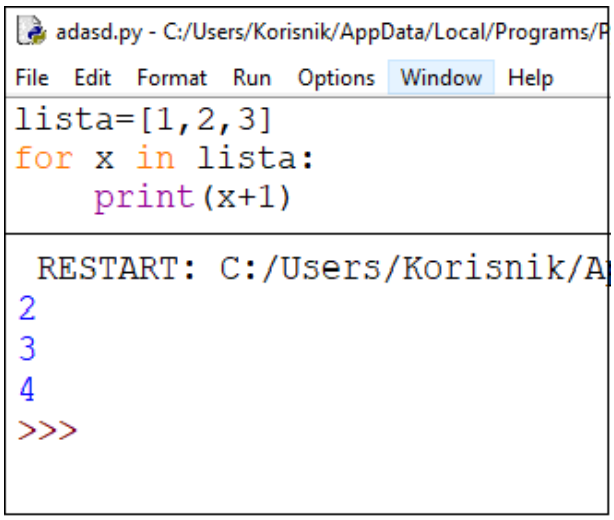
Slika 25. Primjer usporedbe lista

3.3. Iteracija kroz listu

Proces prolaska kroz petlju nazivamo **iteracijom**. **Petlje** („for“ i „while“) i **brojač** (kod „while“ petlje) koriste se kada je potrebno dio programskog koda izvršiti nad svakim elementom liste[31].

„For“ petlja koristi se često u programiranju zbog smanjivanja veličine programa i mogućnosti proizvoljnih ponavljanja blokova programskog koda, što uvjetuje zadržavanju jednake funkcionalnosti i rezultata programa. Petlja „for“ može se često zamisliti kao skraćena inačica petlje „while“, s obzirom da se „for“ petljom često može postići isti rezultat programa manjom količinom koda[32].

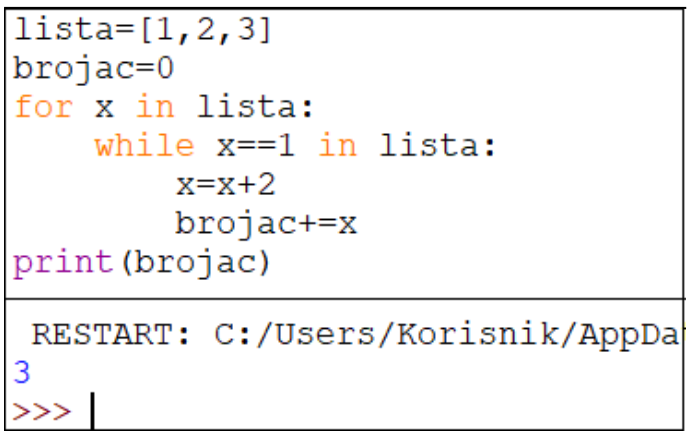
[31] Dunđer, I. *Programiranje,* 2018: 91.
[32] Dunđer, I. *Programiranje,* 2018: 90-93.



Slika 26. Primjer „for“ petlje

U gore navedenom primjeru (slika 26) prikazan je jednostavan primjer „for“ petlje. „For“ petlja iterira kroz svaki element liste (u ovom slučaju 1, 2 i 3) te svaki element uvećava za 1.

Petlja „while“ također izvršava iteraciju kroz listu, no uz znatno veću količinu programskog koda u odnosu na petlju „for“ (Dunđer, 2018: 92). Međutim, postoje određeni problemi koji se ne mogu lako riješiti pomoću petlje „for“ te se tada koristi petlja „while“. Naime, „for“ petlja je vrsta petlje s unaprijed poznatim brojem ponavljanja, dok je „while“ petlja vrsta petlje s unaprijed nepoznatim brojem ponavljanja, zbog čega nekada može više odgovarati zadatku[33]. „While“ petlja često se koristi i u kombinaciji s „for“ petljom.



Slika 27. Primjer korištenja „while“ petlje

[33] Dunđer, I. *Programiranje*, 2018: 90-93.

U gore navedenom primjeru (slika 27) prikazan je primjer korištenja „while“ petlje. U ovom slučaju, „while“ petljom određujemo uvjet (x==1) te se daljnje naredbe („x=x+2“ i „brojac+=x“) izvršavaju samo ako je ispunjen navedeni uvjet, što u ovom slučaju jest (prvi element liste jednak je jedinici). Naredbom „x=x+2“ pridružujemo prvom elementu „2“, te se naredbom „brojac+=x“ pridodaje zbroj prethodno navedenoj varijabli brojac, te kao rezultat proizlazi „3“ (1+2).

3.4. Funkcije za liste

Funkcija predstavlja skup naredbi koje primaju ulazne podatke (engl. *input*), vrše specifičnu obradu određenih podataka što rezultira izlaznim podatcima (engl. *output*). To su potprogrami koji primaju 0 ili više argumenata i vraćaju jednu vrijednost[34]. Funkcije najčešće zahtijevaju jedan ili više argumenata, stoga se i pri određivanju izraza koriste oble zagrade, tj. (). Funkciju također možemo prepoznati i po tome što je navedena prije poziva i ne zahtijeva notaciju točkom. Uz to, funkcije mogu stajati samostalno, npr. print() je ispravna naredba (Dunđer, 2018:84).

U sljedećem dijelu rada navedene su ugrađene funkcije za liste uz nekoliko funkcija koje zahtijevaju pozivanje određenog modula (npr. random, math, …).

Funkcija range(x, z): može poslužiti za stvaranje liste. Argument x predstavlja početni broj (koji je uključen u rezultat), a argument z predstavlja broj do kojeg se brojanje izvršava, no rezultat ne uključuje broj z (slika 28).

```
>>> lista=[]
>>> for x in range(3):
        lista.append(x**2)


>>> lista
[0, 1, 4]
>>>
```

Slika 28. Funkcija „range()“

Funkcija print(L): ispisuje sadržaj liste L (slika 29).

[34] Dunđer, I. *Programiranje*, 2018: 83.

```
>>> lista=[1,2,3, "a", "b"]
>>> print(lista)
[1, 2, 3, 'a', 'b']
>>>
```

Slika 29. Funkcija „print()“

Funkcija list(x): konvertira sekvencu x u listu (slika 30).

```
>>> sekvenca="Pada kiša"
>>> list(sekvenca)
['P', 'a', 'd', 'a', ' ', 'k', 'i', 'š', 'a']
>>>
```

Slika 30. Funkcija „list()“

Funkcija len(L): vraća duljinu liste L, tj. broj elemenata liste (slika 31).

```
>>> lista=[1,2,3,4,5,7, "A", "B", "G", "T"]
>>> len(lista)
10
>>>
```

Slika 31. Funkcija „len()“

Funkcija min(L): vraća najmanju vrijednost među elementima liste L (slika 32).

Funkcija max(L): vraća najveću vrijednost među elementima liste L (slika 32).

```
>>> lista=[1, 2, 3, 4,-2,-10, 5, 10]
>>> min(lista)
-10
>>> max(lista)
10
>>>
```

Slika 32. Funkcije „min()“ i „max()“

Funckija any(L): vraća True ako postoji najmanje jedna vrijednost unutar liste L koja je True. Vraća False u slučaju odsustva vrijednosti, None, {} (prazan rječnik), [] (prazna lista), () (prazna n-torka), set() (prazan skup)[35].

Funkcija all(L): vraća True ako su svi elementi liste L True. U slučaju prazne liste Python također vraća True.

```
>>> lista1=[(), None, {}]
>>> any(lista1)
False
>>> all(lista1)
False
>>> lista2=[1, 2, "a", None]
>>> any(lista2)
True
>>> all(lista2)
False
>>> lista3=[1, 2, "b", "v"]
>>> any(lista3)
True
>>> all(lista3)
True
>>>
```

Slika 33. Funkcije „any()“ i „all()“

[35] Lott, S. F. *A Programmer's Introduction to Python: Building Skills in Python* [online]. Dostupno na: http://www.itmaybeahack.com/book/python-2.6/html/index.html [10.04.2019.]

Lista „lista1“ u primjeru (slika 33) se sastoji od elemenata bez vrijednosti te su rezultati obiju funkcija False. Lista „lista2“ sastoji se od 4 elementa, od kojih je jedan bez vrijednosti, te je rezultat funkcije „any()“ True jer lista sadrži 3 vrijednosti koje su True, a rezultat funkcije „all()“ jest False zbog odsustva vrijednosti jednog elementa (None). Lista „lista3“ u primjeru se sastoji samo od vrijednosti koje su True te su rezultati obiju funkcija True.

Funkcija enumerate(L, start): iterira kroz listu L i vraća n-torku tipa (indeks, element). Numerira sve elemente u sekvenci (slika 34). Argument “start” predstavlja broj kojim započinje brojanje[36].

```
>>> lista=["jagode", "trešnje", "šljive", "lubenica"]
>>> list(enumerate(lista,1))
[(1, 'jagode'), (2, 'trešnje'), (3, 'šljive'), (4, 'lubenica')]
>>>
```

Slika 34. Funkcija „enumerate()“

Funkcija sorted(L): sortira listu L (slika 35).

Funkcija reversed(L): iterira kroz listu L u suprotnom slijedu (slika 35).

```
>>> lista=[1,2,3,4,5,8,3,4,14,12,15]
>>> sorted(lista)
[1, 2, 3, 3, 4, 4, 5, 8, 12, 14, 15]
>>> list(reversed(lista))
[15, 12, 14, 4, 3, 8, 5, 4, 3, 2, 1]
>>>
```

Slika 35. Funkcije „sorted()“ i „reversed()“

Funkcija repr(L): ispisuje reprezentativni oblik objekta L (slika 36).

[36] Lott, S. F. *A Programmer's Introduction to Python: Building Skills in Python* [online]. Dostupno na: http://www.itmaybeahack.com/book/python-2.6/html/index.html [10.04.2019.]

```
>>> lista1=[1, "a", "b", 5]
>>> repr(lista1)
"[1, 'a', 'b', 5]"
>>>
```

Slika 36. Funkcija „repr()“

Funkcija map(x, L): primjenjuje funkciju x na svakom elementu liste L.

```
>>> lista=[1,2,3,4]
>>> kvadrati=list(map(lambda x: x**2, lista))
>>> print(kvadrati)
[1, 4, 9, 16]
>>>
```

Slika 37. Funkcija „map()“

U ovom primjeru (slika 37) na mjestu funkcije nalazi se lambda. U Pythonu, **lambda funkcija** predstavlja anonimnu funkciju, tj. funkciju bez imena. Njezina sintaksa sastoji se od argumenta i izraza. U navedenom primjeru, argument je x (svaki element liste), a izraz jest „x**2“, što znači da Python treba izračunati kvadrat svakog elementa liste i stvoriti novu listu s dobivenim rezultatima.

Naredba del(L[x]): u pozadini pokreće metodu nad objektom koji treba izbrisati (slika 38). Pomoću navedene naredbe možemo izbrisati element, određeni raspon ili cijelu listu L.

```
>>> lista=[1,2,3,4,5,6]
>>> del(lista[0])
>>> print(lista)
[2, 3, 4, 5, 6]
>>> del(lista[1:2])
>>> print(lista)
[2, 4, 5, 6]
>>> del(lista[1:3])
>>> print(lista)
[2, 6]
>>> del(lista)
>>> print(lista)
Traceback (most recent call last):
  File "<pyshell#210>", line 1, in <module>
    print(lista)
NameError: name 'lista' is not defined
>>>
```

Slika 38. Naredba „del()“

Funkcija sum(L): vraća zbroj svih elemenata liste L (slika 39).

```
>>> lista=[2,6,9,4,34,65,867]
>>> sum(lista)
987
>>>
```

Slika 39. Funkcija „sum()“

Funkcija filter(x, lista): stvara novu listu na temelju filtera funkcije x (Dunđer, 2018:87).

```
>>> lista=range(-5, 5)
>>> lista1=list(filter(lambda x: x>0, lista))
>>> print(lista1)
[1, 2, 3, 4]
>>>
```

Slika 40. Funkcija „filter()“

Na ovom primjeru (slika 40) prikazana je „filter()“ funkcija. Lista „lista1“ jest nova lista stvorena iz liste „lista“ i sadrži elemente liste „lista“ koji su veći od 0, specificirani izrazom „x>0“.

3.5. Metode za liste

Metode predstavljaju funkcije ili procese koji se koriste uz neki objekt (Hetland, 2017: 38). U Pythonu sve vrijednosti podataka su objekti i svaki tip podataka sadrži skup metoda koje se mogu koristiti nad objektima tog tipa (Lambert, 2009: 139). Drugim riječima, metoda je funkcija koja je dostupna za određeni objekt s obzirom na **tip objekta** (Hetland, 2017: 38)**.** Općenito, sintaksa metode izgleda ovako: **objekt.metoda(argumenti)**. Poziv metode izgleda poput poziva funkcije osim što se kod metoda objekt navodi prije imena metode, odvojen notacijom[37]. Također, za razliku od funkcija, metoda ne može stajati samostalno.

Zbog promjenjive prirode liste, metode modificiraju originalnu listu nad kojom se provodi radnja (za razliku od npr. stringova i n-torki gdje metode stvaraju novi složeni tip podataka, a početni objekt ostaje netaknut)[38]. Za takve se funkcije kaže da djeluju „u mjestu“ (engl. *in place*). Kako bi izvorna lista bila sačuvana, treba načiniti njezinu kopiju i permutirati tu kopiju (Summerfield, 2009: 114-116). U sljedećem djelu rada navedene su ugrađene metode koje se mogu primijeniti nad listama.

Metoda list.append(x): dodaje element x na kraj liste (slika 41).

Metoda list.extend(L): povećava/širi listu dodavajući sve navedene elemente zadanoj listi (slika 41).

```
>>> lista=[1,2,3]
>>> lista.append(4)
>>> print(lista)
[1, 2, 3, 4]
>>> lista.extend(range(5))
>>> print(lista)
[1, 2, 3, 4, 0, 1, 2, 3, 4]
>>>
```

Slika 41. Metode „append()“ i „extend()“

[37] Hetland, M. L. *Beginning Python:From Novice to Professional*, 2017: 38-39.
[38] Sturtz, J. (2018) *Lists and Tuples in Python* [online]. Dostupno na: https://realpython.com/python-lists-tuples/ [10.04.2019].

Metoda list.insert(i, x): dodaje element x na poziciju „i“ u listi (slika 42).

Metoda list.remove(x): uklanja prvi element iz liste čija je vrijednost x. Pojavljuje se greška ako navedeni element ne postoji (slika 42).

```
>>> lista=[1,2, "a", "b", "c"]
>>> lista.insert(2, 5)
>>> print(lista)
[1, 2, 5, 'a', 'b', 'c']
>>> lista.remove("b")
>>> print(lista)
[1, 2, 5, 'a', 'c']
>>> lista.remove(4)
Traceback (most recent call last):
  File "<pyshell#240>", line 1, in <module>
    lista.remove(4)
ValueError: list.remove(x): x not in list
>>>
```

Slika 42. Metode „insert()“ i remove()“

Metoda list.pop([i]): uklanja i vraća element koji se nalazi na navedenoj poziciji „i“ (slika 43). Ako indeks nije specificiran, metoda **list.pop()** uklanja i vraća zadnji element liste. Uglate zagrade oko argumenta „i“ ukazuju na to da je navođenje indeksa opcionalno (Rossum, 2016:29).

```
>>> lista=[1,"a", 2, "b", 3, "c"]
>>> lista.pop(2)
2
>>> lista.pop()
'c'
>>> print(lista)
[1, 'a', 'b', 3]
>>>
```

Slika 43. Metoda „pop()“

Metoda list.clear(L): uklanja sve elemente iz liste L (slika 44).

Metoda list.index(x): vraća indeks prvog elementa u listi čija je vrijednost x. Također se pojavljuje greška u slučaju da nema navedenog elementa (slika 44).

Metoda list.count(x): vraća broj koliko se puta element x pojavljuje u listi (slika 44).

```
>>> lista1=[1,2,3]
>>> lista1.clear()
>>> print(lista1)
[]
>>> lista2=[1,2, "a", "b", "c"]
>>> lista2.index("a")
2
>>> lista2.index(5)
Traceback (most recent call last):
  File "<pyshell#250>", line 1, in <module>
    lista2.index(5)
ValueError: 5 is not in list
>>> lista2.append(2)
>>> lista2.count(2)
2
>>>
```

Slika 44. Metode „clear()“, „index()“ i „count()“

Metoda list.sort([key], [reverse=True/False]): sortira elemente u listi, slično kao i funkcija „sorted“ (slika 45). Argument „key“ je opcionalan, omogućuje zadavanje funkcije kojom se definira usporedba elemenata, npr. ako je potrebno sortirati elemente prema njihovoj dužini, koristi se funkcija **len()** kao *key* funkcija (Hetland, 2017:41). Parametar „reverse“ označava argument koji je također opcionalan i koji se odnosi na način sortiranja elemenata (silazno/uzlazno) (Dunđer, 2018: 104).

Metoda list.reverse(): preokreće redoslijed elemenata u listi (slika 45).

Metoda list.copy(): vraća površnu, tj. plitku kopiju liste (slika 45). Postupak je sličan korištenju L[:] za površno kopiranje liste te korištenju funkcije list (L), gdje oboje također kopiraju listu L.

```
>>> lista=[3,4,1,43,23,64,23]
>>> lista.sort(key=None, reverse=False)
>>> print(lista)
[1, 3, 4, 23, 23, 43, 64]
>>> lista.reverse()
>>> print(lista)
[64, 43, 23, 23, 4, 3, 1]
>>> lista.copy()
[64, 43, 23, 23, 4, 3, 1]
>>>
```

Slika 45. Metode „sort()“, „reverse()“ i „copy()“

Metoda random.shuffle(L): ova metoda nalazi se u modulu „random“. Obavlja nasumično preslagivanje elemenata liste L (nasumičnu permutaciju liste). Argument L predstavlja listu koju metoda koristi (slika 46). Ova funkcija također ne generira novu listu te je izvorni redoslijed elemenata liste bespovratno izgubljen (Budin et al., 2015:197-198).

Metoda random.choice(L): vraća nasumični element navedene liste L (slika 46).

Metoda random.sample(L, x): vraća x nasumičnih elemenata liste L, pri čemu će se svaki element liste u rezultatu može pojaviti najviše jednom (Budin et al., 2015:198) (slika 46). Za ovu metodu, kao i za metodu „choice()“ i „shuffle()“, potrebno je prije dozivanja metoda pozvati modul „random“ („import random“).

```
>>> lista=[1,4,7,34,"abc", "bf", "t", "f"]
>>> import random
>>> random.shuffle(lista)
>>> print(lista)
[4, 1, 't', 34, 'bf', 'abc', 7, 'f']
>>> random.choice(lista)
'f'
>>> random.choice(lista)
1
>>> random.sample(lista, 3)
['f', 7, 4]
>>>
```

Slika 46. Metode „shuffle()“, „choice()“ i „sample()“

3.6. Red i stog

Zahvaljujući svojim obilježjima „kontejnera“ i promjenjivosti, liste su veoma fleksibilne te je moguće kreirati druge vrste struktura podataka koristeći liste. Dvije najznačajnije memorijske strukture podataka takvog obilježja jesu stog i red (Chun, 2006:185).

Stog (engl. *stack*) je kolekcija objekata koji su nadodani i uklonjeni po principu „LIFO“ (Last-In-First-Out) (slika 47). U prijevodu, element koji zadnji ulazi u stog prvi napušta strukturu podataka. Metode „append()“ i „pop()“ mogu se iskoristiti za stvaranje standardnog stoga (Goodrich et al., 2013: 229). Metoda „append()“ svrstava objekt na kraju liste (na vrh stoga), te se metodom „pop()“ zadnje dodani element uklanja i vraća[39]. Stogovi su korišteni u mnogim aplikacijama, uključujući sljedeće: internetski web preglednik pohranjuje adrese nedavno posjećenih stranica u stogu. Prilikom posjete korisnika novoj stranici, adresa navedene stranice je nadodana u stog prije posjećenih adresa. Preglednik tada omogućava korisniku da se vrati na prethodno posjećenu stranicu koristeći gumb „nazad“. Aplikacije za uređivanje teksta pružaju „undo“ gumb, mehanizam koji „vraća“ nedavne izmjene teksta i vraća dokument u prijašnje stanje (prije zadnje promjene). Ova operacija moguća je pomoću pohrane navedenih izmjena u stogu[40].

Red (engl. *queue*) je kolekcija objekata koji su nadodani i uklonjeni po principu „FIFO“ (First-In-First-Out) (Goodrich et al., 2013: 239) (slika 47). Element koji prvi ulazi u red prvi napušta strukturu podataka. Iste dvije metode („append()“ i „pop()“) mogu se iskoristiti za stvaranje standardnog reda. Metoda „append()“ svrstava objekt na kraj reda, a pozivom na metodu „pop(0)“ uklanja se i vraća prvi element reda[41]. Kao i stogovi, korišteni su u mnogim aplikacijama, uz sljedeće: trgovine, kazališta, rezervacijski centri i druge slične uslužne djelatnosti procesiraju zahtjeve korisnika prema „FIFO“ principu. Red bi bio logičan izbor za strukturu podataka za prihvaćanje poziva centra službe za korisnike. „FIFO“ redovi također su korišteni od strane mnogih kompjuterskih uređaja, kao što su pisač ili web poslužitelj koji odgovara na određene zahtjeve/upite korisnika[42].

[39] Sturtz, J. (2018) *Lists and Tuples in Python* [online]. Dostupno na: https://realpython.com/python-lists-tuples/ [12.04.2019].
[40] Goodrich et al. *Data Structures and Algorithms in Python*, 2013: 229-230.
[41] Sturtz, J. (2018) *Lists and Tuples in Python* [online]. Dostupno na: https://realpython.com/python-lists-tuples/ [12.04.2019]
[42] Goodrich et al. *Data Structures and Algorithms in Python*, 2013: 239..

```
>>> stog=[]
>>> stog.append("a")
>>> stog.append(2)
>>> stog.append("abc")
>>> print(stog)
['a', 2, 'abc']
>>> stog.pop()
'abc'
>>> print(stog)
['a', 2]
>>> red=[]
>>> red.append("b")
>>> red.append(1)
>>> red.append("def")
>>> red.pop(0)
'b'
>>> print(red)
[1, 'def']
>>>
```

Slika 47. Primjer stoga i reda

4. N-torke

N-torke (engl. *tuple*) predstavljaju još jednu od ugrađenih slijednih kolekcija u Pythonu, koja se sastoji od 0 ili više objektnih referenci/elemenata (Summerfield, 2009: 108). N-torke su, za razliku od lista, nepromjenjive (engl. *immutable*), što znači da nije moguće dodavati, brisati ili mijenjati poredak elemenata kao kod lista. Iznimku predstavlja promjenjiva struktura podataka unutar n-torke, kada je moguće promijeniti sadržaj ugniježđene strukture. Iako u jednom pogledu jest nedostatak, nepromjenjivost je poželjna kada je potrebno sačuvati integritet podataka predstavljenih n-torkom (Stojanović, 2012: 105). Slovo „n“ u izrazu „n-torka“ označava broj elemenata n-torke.

```
>>> t1=(1,)
>>> t2=(1,2)
>>> t3=(1,2,"a",)
>>>
```

Slika 48. Različite n-torke

Prema gore navedenom primjeru (slika 48), varijabla „t1“ bila bi 1-torka (engl. *singleton*), varijabla „t2“ bila bi 2-torka te varijabla „t3“ 3-torka itd.

Također, za razliku od lista, gdje su elementi omeđeni uglatim zagradama, elementi n-torke omeđeni su oblim zagradama[43]. N-torka se sastoji od određenog broja vrijednosti odvojenih zarezom unutar oblih zagrada: **n-torka=(1, 2, 5, „a“, „b“)**[44]. N-torka koja se sastoji od jednog elementa (engl. *singleton*) formira se na način da se određenom izrazu (elementu) pridruži zarez (kod liste je to opcionalno), a dodavanje zareza i kod n-torki s više elemenata također ispravno[45]. Iako ne podržavaju listine metode, n-torke dijele većinu svojstava s listama. Osnovna svojstva n-torke navedena su u nastavku.

N-torke su kolekcije arbitrarnih objekata: kao i liste, n-torke su pozicijski uređene kolekcije elemenata, tj. održavaju red sadržaja s lijeve strane na desnu te mogu sadržavati bilo koji tip podataka (Lutz, 2013: 276).

[43] Lambert, K. A., *Fundamentals of Python: From First Programs through Dana Structures*, 2009: 173.
[44] Swaroop, C. H. *A Byte of Python* [online], 2013: 65-66. Dostupno na: http://files.swaroopch.com/python/python3/byte_of_python.pdf [12.04.2019].
[45] Summerfield, M. *Programming in Python 3: A Complete Introduction to the Python Language*, 2009: 108.

N-torkama se pristupa pomoću indeksa: kao kod stringova i lista, elementima n-torke pristupa se pomoću indeksa. Podržavaju sve operacije bazirane na pristupu indeksom, kao što su indeksiranje i izrezivanje (engl. *slicing*).

N-torke spadaju u nepromjenjive strukture podataka: poput lista, n-torke su također nizovi i podržavaju mnoge iste operacije. Međutim, n-torke su nepromjenjive te ne podržavaju operacije koje mijenjaju podatke „u mjestu" (engl. *in place change operations*) koje se mogu aplicirati na listama[46].

N-torke su fiksne duljine, heterogene i podržavaju ugnježđivanje: zbog svojstva nepromjenjivosti n-torki, ne može se promijeniti duljina niza bez stvaranja njegove kopije. S druge strane, n-torke mogu sadržavati sve tipove objekata, uključujući druge složene objekte (liste, rječnike, druge n-torke) te time podržavaju ugnježđivanje[47].

N-torke su nizovi objektnih referenci: poput lista, n-torke se smatraju nizovima objektnih referenci. N-torke također pohranjuju pristupačne točke na druge objekte (reference) i pristup elementima se obavlja znatno brže nego kod lista[48].

Pristupanje elementima n-torke: za razliku od lista, elementima n-torke pristupa se procesom raspakiravanja n-torke (engl. *unpacking*), indeksiranja ili pomoću atributa u slučaju „imenovane n-torke" (funkcija „namedtuple()")[49].

Razlozi zašto je ponekad pogodnije koristiti n-torke umjesto liste su: n-torke su procesirane brže od lista. U slučaju kreiranja konstantnog skupa vrijednosti koji se ne mijenja i kroz koji je potrebno jednostavno iterirati, koriste se n-torke. Elementi unutar n-torke su, prema definiciji, zaštićeni od izmjene (modifikacije). Na ovaj način, nije moguće slučajno promijeniti vrijednosti. N-torke mogu se također koristiti kao ključevi za rječnike te u procesu formatiranja stringa sadržavajući vrijednosti koje se ugnježđuju u string[50].

[46] Lutz, M. *Learning Python,* 2013: 275-277.
[47] Lutz, M. *Learning Python,* 2013: 276.
[48] Lutz, M. *Learning Python,* 2013: 276.
[49] Rossum G. *Python tutorial* [online], 2018: 36-37. Dostupno na: http://www.cobsco.com/python-3.6.5-docs-pdf/tutorial.pdf [12.04.2019].
[50] Jackson, C. *Learning to Program Using Python* [online], 2009: 64-65. Dostupno na: https://www.ida.liu.se/~732A47/literature/PythonBook.pdf [12.04.2019].

```
>>> rjecnik={'a':0, 'b':1}
>>> lista=rjecnik.items()
>>> print(lista)
dict_items([('a', 0), ('b', 1)])
>>> lista=[('a',0), ('b',1)]
>>> rjecnik=dict(lista)
>>> print(rjecnik)
{'a': 0, 'b': 1}
>>>
```

Slika 49. N-torke kao ključevi za rječnike

U gore navedenom primjeru (slika 49), u prvom dijelu korištena je metoda „items()“ nad rječnikom, te kao ispis proizlazi lista s n-torkama koje sadrže dva elementa, gdje jedan element n-torke predstavlja ključ, a drugi element vrijednost, ovisno o tome koliko ključeva i vrijednosti ima u rječniku (u ovom slučaju dva). U drugom dijelu primjera je lista koja sadrži n-torke, a koja se funkcijom „dict(lista)“ konvertira u rječnik – ovdje se vidi kako je moguće kreirati rječnik uz pomoć n-torki.

```
>>> element1=1
>>> element2=2
>>> print("%s + 1 jednako je %d"% (element1, element2))
1 + 1 jednako je 2
>>>
```

Slika 50. Primjer formatiranja stringa

U navedenom primjeru (slika 50) prikazano je formatiranje stringa. Varijable „element1“ i „element 2“ ubacuju se na odgovarajuća mjesta u nizu znakova. Izraz „%s“ znači da element treba biti prikazan u string formatu, a izraz „%d“ da element treba biti prikazan kao broj.

4.1. Kreiranje n-torke

Za kreiranje n-torke s jednim elementom (engl. *singleton*) potrebno je uključiti zarez nakon vrijednosti. Drugi način za stvaranje n-torke jest pomoću ugrađene funkcije „tuple()“ (slika 51). Bez navedenog argumenta, navedena funkcija stvara praznu n-torku[51]. Prazna n-torka također nastaje kada određenoj varijabli pridružimo prazne oble zagrade. U slučaju da je

[51] Hetland, *Beginning Python*, 2017: 42-42.

navedeni argument niz (lista, string ili n-torka), rezultat funkcije je n-torka koja sadrži elemente niza (Downey, 2012:135).

```
>>> t=()
>>> type(t)
<class 'tuple'>
>>> a=tuple()
>>> print(a)
()
>>> type(a)
<class 'tuple'>
>>>
>>> t=(1,)
>>> type(t)
<class 'tuple'>
>>> t=1,
>>> type(t)
<class 'tuple'>
>>> t=tuple("banana")
>>> print(t)
('b', 'a', 'n', 'a', 'n', 'a')
>>> type(t)
<class 'tuple'>
>>>
```

Slika 51. Kreiranje n-torke

N-torke mogu se kreirati i pomoću nekih drugih funkcija. Kao primjer navedena je funkcija „divmod()“ koja dopušta 2 argumenta i vraća n-torku koja se sastoji od dvije vrijednosti, količnika i ostatka dijeljenja (Downey, 2012:137).

```
>>> n_torka=divmod(30,4)
>>> print(n_torka)
(7, 2)
>>>
```

Slika 52. Funkcija „divmod()“

Na navedenom primjeru (slika 52) vidi se da navedena funkcija uzima dva argumenta. Prvi argument predstavlja djeljenik (u ovom slučaju „30“), a drugi argument jest djelitelj („4“). Kao rezultat funkcije nastaje n-torka s dvije vrijednosti, prvi element jest količnik, dok drugi element predstavlja ostatak dijeljenja.

Kreiranje n-torki također je moguće pomoću ugrađene funkcije „zip()“, koja prima dvije ili više sekvenci i sažima ih u listu n-torki gdje svaka n-torka sadrži jedan element iz svake sekvence (Downey, 2012: 138). Primjer stvaranja n-torke pomoću ove funkcije nalazi se na slici 53.

```
>>> string="abc"
>>> lista=[1,2,3]
>>> lista2=list(zip(string,lista))
>>> print(lista2)
[('a', 1), ('b', 2), ('c', 3)]
>>>
```

Slika 53. Funkcija „zip()“

4.2. Imenovane n-torke

N-torke se obično koriste za reprezentaciju jednostavnih struktura podataka (Beazley, 2009: 263), a problem kod ove složene strukture podataka jest pristup elementima (podatcima) putem numeričkog indeksa, dovodeći do koda koji može djelovati zbunjujuće za čitanje te ga je teško održavati osim ako je korisnik u stanju zapamtiti vrijednosti svake numeričke vrijednosti indeksa (problem raste sukladno rastu n-torke)[52]. Python podržava tip kontejnera, podklasu n-torke nalik rječnicima, zvanu **imenovana n-torka** (Lutz, 2013: 281). Funkcija za stvaranje imenovane n-torke, „namedtuple()“, koja je dostupna u Pythonovom modulu „collections“, **omogućava pristup elementima putem pozicije i imena atributa** i može biti konvertirana u formu nalik rječniku za pristup putem „ključa“. Imena atributa proizlaze iz elemenata i nisu u potpunosti nalik ključevima u rječnicima, ali su veoma slični[53]. Sintaksa funkcije „namedtuple()“ glasi: „**namedtuple(typename, [fieldnames])**“, gdje nastaje podklasa n-torke s imenom „typename“, dok „fieldnames“ predstavlja listu imena atributa

[52] Beazley D.M. *Python Essential Reference*, 2009: 263.
[53] Lutz, M. *Learning Python*, 2013: 281.

specificiranih kao string. Imena koja se nalaze u ovoj listi trebala bi biti validni identifikatori Pythona, ne smiju započinjati s podvlakom i specificirana su istim redoslijedom kao elementi koje sadrži n-torka[54].

```
>>> from collections import namedtuple
>>> prvi=namedtuple("prvi", ["ime", "prezime", "godina"])
>>> ana= prvi("Ana", "Anić", "2019")
>>> print(ana)
prvi(ime='Ana', prezime='Anić', godina='2019')
>>> ana[0], ana[1]
('Ana', 'Anić')
>>>
>>> ana.ime, ana.prezime, ana.godina
('Ana', 'Anić', '2019')
>>>
```

Slika 54. Primjer imenovane n-torke

Na gornjoj slici (slika 54) prikazan je primjer kreiranja imenovane n-torke. Naredbama „ana[0]“ i „ana[1]“ dohvaćamo elemente n-torke pomoću pozicije elemenata (kao i indeksiranje kod lista), a naredbama „ana.ime“ i „ana.prezime“ dohvaćamo elemente prema imenu atributa koji je određen u imenovanoj n-torci.

```
>>> rjecnik=ana._asdict()
>>> rjecnik["ime"], rjecnik["prezime"], rjecnik["godina"]
('Ana', 'Anić', '2019')
>>> rjecnik
OrderedDict([('ime', 'Ana'), ('prezime', 'Anić'), ('godina', '2019')])
>>>
```

Slika 55. Konvertiranje u rječnik

Konvertiranje imenovane n-torke u rječnik, što je prikazano na gornjem primjeru (slika 55), također podržava dohvaćanje elemenata putem „ključa“ (u ovom slučaju ključ je ime atributa), slično kao kod rječnika. Imenovane n-torke, što se može uočiti pomoću primjera, predstavljaju hibridni tip podataka n-torke/rječnika (Lutz, 2013: 282). Ovakav tip n-torke

[54] Beazley D.M. *Python Essential Reference*, 2009: 264.

gradi nove klase n-torke unoseći **metodu pristupanja elementima pomoću atributa** za svako imenovano polje, koja mapira ime atributa i njegovu poziciju (Lutz, 2013: 281).

4.3. Operacije nad n-torkama

Većina operacija koje se provode nad listama (opisanih u poglavlju 3.2. – Operacije nad listama) mogu se također koristiti nad n-torkama.

Indeksiranje n-torke: kao i kod liste, elementima n-torke možemo pristupati pomoću indeksa, ali ih, za razliku od lista, pomoću indeksa nije moguće mijenjati. Na donjoj slici (slika 56) prikazano je koje klase pogrešaka Python vraća pokuša li korisnik promijeniti element n-torke ili dohvatiti element koji ne postoji. Naime, pokušavajući dohvatiti element n-torke po indeksu 7, Python vraća „IndexError“ što znači da je navedeni indeks u argumentu veći od duljine n-torke i da ga Python ne može dohvatiti. Pri pokušaju mijenjanja drugog elementa n-torke (pod indeksom 1) Python vraća „TypeError“, što znači da navedeni tip podataka (u ovom slučaju n-torka) ne podržava proces koji je naredbom „n_torka[1]=5“ zadan interpreteru.

```
>>> n_torka=(1, 2, 3,"a", "b", "c", "d")
>>> print(n_torka[0], n_torka[2], n_torka[5])
1 3 c
>>> print(n_torka[7])
Traceback (most recent call last):
  File "<pyshell#89>", line 1, in <module>
    print(n_torka[7])
IndexError: tuple index out of range
>>> n_torka[1]=5
Traceback (most recent call last):
  File "<pyshell#90>", line 1, in <module>
    n_torka[1]=5
TypeError: 'tuple' object does not support item assignment
>>>
```

Slika 56. Indeksiranje n-torke

Izrezivanje n-torke: kao liste, n-torke je moguće izrezivati.

```
>>> n_torka=("a", "b", "c", 4, 5, 6, 7)
>>> n_torka[1:4]
('b', 'c', 4)
>>> n_torka[:3]
('a', 'b', 'c')
>>> n_torka[3:]
(4, 5, 6, 7)
>>> n_torka[::]
('a', 'b', 'c', 4, 5, 6, 7)
>>> n_torka[::-1]
(7, 6, 5, 4, 'c', 'b', 'a')
>>> n_torka[::-2]
(7, 5, 'c', 'a')
>>> n_torka[0:3]=(1,2,3)
Traceback (most recent call last):
  File "<pyshell#98>", line 1, in <module>
    n_torka[0:3]=(1,2,3)
TypeError: 'tuple' object does not support item assignment
>>>
```

Slika 57. Izrezivanje n-torke

Kao što je prikazano na gornjoj slici (slika 57), postupak izrezivanja analogan je izrezivanju liste, sve do točke kada se pokuša određeni dio n-torke promijeniti. Pri pokušaju mijenjanja prva tri elementa n-torke izrazom „n_torka[0:3]=(1, 2, 3)“, Python interpreter javlja „TypeError“, dajući do znanja korisniku da nije moguće promijeniti element n-torke. Međutim, ako je element n-torke promjenjivog tipa podataka, kao što je prikazano na donjoj slici (slika 58), moguće je promijeniti sadržaj navedenog elementa unutar n-torke. U spomenutom primjeru, prvi element liste unutar n-torke („1“) promijenjen je u izmijenjenu vrijednost („4“).

```
>>> n_torka=(1,2, [1,2,3])
>>> n_torka[2][0]=4
>>> print(n_torka)
(1, 2, [4, 2, 3])
```

Slika 58. Mijenjanje liste unutar n-torke

Konkatenacija, nadovezivanje i usporedba n-torki: kao i kod lista, n-torke je također moguće zbrajati i nadovezivati te uspoređivati (slika 59) koristeći se aritmetičkim operatorima („+“ i „*“), odnosno relacijskim operatorima („==“, „!=“, „<“, „>“, „<=“, „>=“)[55].

```
>>> n_torka=(1,2,3)
>>> n_torka2=("a", "b")
>>> n_torka+n_torka2
(1, 2, 3, 'a', 'b')
>>> 3*(4,)
(4, 4, 4)
>>> 3*n_torka
(1, 2, 3, 1, 2, 3, 1, 2,
>>> n_torka3=(1,2,3)
>>> n_torka==n_torka3
True
>>> n_torka!=n_torka2
True
>>> n_torka!=n_torka3
False
>>> n_torka>n_torka3
False
>>> n_torka<n_torka3
False
>>> n_torka<=n_torka3
True
>>>
```

Slika 59. Konkatenacija, nadovezivanje i usporedba n-torki

Također, operatori „in“ i „not in“, petlje „for“ i „while“ te konstrukti „if“, „if-else“, „if-elif-else“, uz to što se mogu koristiti nad listama, mogu se također koristiti i nad n-torkama (slika 60).

[55] Hetland, M. L. *Beginning Python: From Novice to Professional*, 2017: 43-44.

```
>>> n_torka=(1, 6, 4, 76, "gh", "hrt" "zth")
>>> 6 in n_torka
True
>>> 2 in n_torka
False
>>> "g" in n_torka
False
>>> "a" not in n_torka
True
>>> 4 not in n_torka
False
>>>
```

Slika 60. Operatori „in“ i „not in“

```
>>> n_torka=(1,2,3)
>>> brojac=0
>>> for x in n_torka:
        if x<=1:
                x=x+2
                brojac+=x
        else:
                x=x+3
                brojac+=x
        print(brojac)


3
8
14
>>>
```

Slika 61. Iteracija kroz n-torku

U gore navedenom primjeru (slika 61) prikazan je primjer iteracije kroz n-torku pomoću „for“ petlje i konstrukta „if-else“. Izraz „for x in n_torka“ naređuje iteraciju kroz svaki element n-torke, „if x<=1“ određuje da se sljedeći niz naredbi izvršava samo ako je element n-torke manji ili jednak jedinici (prvi element n-torke), što zapravo predstavlja prvi rezultat „3“ (1+2). Izraz „else“ se izvršava samo ako je rezultat „if“ izraza False, što vrijedi za drugi i treći element te se izvršava blok naredbi ispod „else“ izraza. Rezultati su „8“, jer je rezultat

„3“ nakon prve iteracije i dodaje se pet (3+5=8) te „14“ kada nadodamo iteraciju zadnjeg elementa (8+6).

4.2.3. Pakiranje i raspakiravanje n-torke

Pakiranje n-torke (engl. *tuple packing*) predstavlja proces dodjeljivanja određenog broja vrijednosti n-torki. Termin „pakiranje“ koristi se jer su podatci „pakirani“ u n-torku (Jackson, 2009: 65). **Raspakiravanje n-torke** (engl. *tuple unpacking)* je metoda koja se koristi za izvlačenje elemenata iz n-torke. Pri pridruživanju „pakiranih“ objekata novoj n-torci, individualne vrijednosti stare n-torke pridružene su (raspakirane) u varijable nove n-torke (slika 62).

```
>>> t=('a', 'b', 'c', 'd')
>>> (e1,e2,e3,e4)=t
>>> e1
'a'
>>> e2
'b'
>>> e3
'c'
>>> e4
'd'
>>>
```

Slika 62. Pakiranje i raspakiravanje n-torke

Glavni razlog zašto je moguće pridruživanje više vrijednosti n-torke određenim varijablama jest taj što je duljina n-torke unaprijed poznata[56]. Svaka vrijednost koja se nalazi na desnoj strani izraza jednakosti pridružena je svojoj odgovarajućoj varijabli s lijeve strane izraza jednakosti (Downey, 2012: 137). Prije pridruživanja, Python evaluira sve elemente koji se nalaze na desnoj i lijevoj strani te provjerava je li broj vrijednosti jednak broju varijabli, te ako nije, vraća „ValueError“ (slika 63). Na desnoj strani može se nalaziti bilo koji tip kolekcije (string, lista, n-torka)[57].

[56] Sturtz, J. (2018) *Lists and Tuples in Python* [online]. Dostupno na: https://realpython.com/python-lists-tuples/ [14.04.2019].

[57] Downey, A. B. *Think Python*, 2012: 136-137).

```
>>> t=(1,2,3)
>>> (a,b,c,d,e,f)=t
Traceback (most recent call last):
  File "<pyshell#273>", line 1, in <module>
    (a,b,c,d,e,f)=t
ValueError: not enough values to unpack (expected 6, got 3)
>>> (a,b)=t
Traceback (most recent call last):
  File "<pyshell#274>", line 1, in <module>
    (a,b)=t
ValueError: too many values to unpack (expected 2)
>>>
```

Slika 63. „ValueError“ kod raspakiravanja n-torke

Jedna od prednosti pakiranja/raspakiravanja n-torke jest što omogućuje zamjenu objekata „u mjestu“ (engl. *in place)* (Jackson, 2009: 64). U Pythonu, zamjena vrijednosti između dvije unaprijed određene varijable često je korisna te se pomoću n-torki može izvršiti koristeći se samo jednim izrazom (slika 64). U drugim programskim jezicima potrebno je uvesti privremenu varijablu kako bi zamjena vrijednosti bila moguća[58]. Kao i kod raspakiravanja, potreban je jednak broj varijabli s obje strane izraza jednakosti.

```
>>> a="abc"
>>> b="def"
>>> a,b
('abc', 'def')
>>> a,b=b,a
>>> a,b
('def', 'abc')
>>>
```

Slika 64. Zamjena vrijednosti varijabli

[58] Jackson, C. *Learning to Program Using Python* [online], 2009: 64-65. Dostupno na: https://www.ida.liu.se/~732A47/literature/PythonBook.pdf [14.04.2019].

4.4. Funkcije i metode za n-torke

Većina funkcija koje se koriste nad listama mogu se koristiti i nad n-torkama. Ugrađene funkcije za n-torke navedene su u nastavku[59].

Funkcija tuple(x): stvara n-torku iz druge sekvence x. Ova funkcija može konvertirati string ili listu u n-torku (slika 65).

```
>>> lista=[1,2,3,4]
>>> string="Dobar dan"
>>> tuple(lista)
(1, 2, 3, 4)
>>> tuple(string)
('D', 'o', 'b', 'a', 'r', ' ', 'd', 'a', 'n')
>>>
```

Slika 65. Funkcija „tuple()“

Funkcija len(N): za n-torke ova funkcija vraća broj elemenata n-torke N (slika 66).

Funkcija max(N): za n-torke ova funkcija vraća najveći element n-torke N (slika 66).

Funkcija min(N): za n-torke ova funkcija vraća najmanji element n-torke N (slika 66).

Funkcija sum(N): za n-torke ova funkcija vraća zbroj svih elemenata n-torke N (slika 66).

```
>>> n_torka=(1, 54, 76, 3, 65, -54, -100)
>>> len(n_torka)
7
>>> min(n_torka)
-100
>>> max(n_torka)
76
>>> sum(n_torka)
45
>>>
```

Slika 66. Funkcije „len()“, „min()“, „max()“ i „sum()“

[59] Lott, S. F. *A Programmer's Introduction to Python: Building Skills in Python* [online]. Dostupno na: http://www.itmaybeahack.com/book/python-2.6/html/index.html [14.04.2019].

Funkcija any(N): za n-torku N kao i liste vraća True ako postoji barem jedan element u nizu koji je True (slika 67).

Funkcija all(N): za n-torku N kao i liste, vraća True ako su svi elementi True (slika 67).

```
>>> n_torka1=(None, (), 1, 2, "a")
>>> n_torka2=(None, {}, (), [],)
>>> n_torka3=(1, 2, 4, "f", "gfs")
>>> any(n_torka1)
True
>>> any(n_torka2)
False
>>> any(n_torka3)
True
>>> all(n_torka1)
False
>>> all(n_torka2)
False
>>> all(n_torka3)
True
>>>
```

Slika 67. Funkcije „all()“ i „any()“

Funkcija enumerate(N): iterira kroz n-torku N vraćajući 2-torke tipa „(indeks, element)“, tj. nabraja sve elemente niza nadodavajući redom elementima indekse (slika 68).

```
>>> n_torka=(1, 5, 8, "f", "rgt", 4.6)
>>> tuple(enumerate(n_torka))
((0, 1), (1, 5), (2, 8), (3, 'f'), (4, 'rgt'), (5, 4.6))
>>>
```

Slika 68. Funkcija „enumerate()“

Funkcija sorted(): iterira kroz n-torku sortiranim redoslijedom (slika 69).

Funkcija reversed(): iterira kroz n-torku suprotnim redoslijedom (slika 69).

```
>>> n_torka=("banane", "jabuke", "trešnje", "lubenice")
>>> n_torka1=(4, 62, 87, -43, 65, -98, 6.5)
>>> sorted(n_torka)
['banane', 'jabuke', 'lubenice', 'trešnje']
>>> sorted(n_torka1)
[-98, -43, 4, 6.5, 62, 65, 87]
>>> tuple(reversed(n_torka))
('lubenice', 'trešnje', 'jabuke', 'banane')
>>> tuple(reversed(n_torka1))
(6.5, -98, 65, -43, 87, 62, 4)
>>>
```

Slika 69. Funkcije „sorted()“ i „reversed()“

Na gore navedenom primjeru (slika 69), funkcija „sorted()“ prvu n-torku sortira po abecedi, a drugu prema veličini brojeva.

Postoje samo dvije metode koje se mogu vršiti direktno nad n-torkama:

Metoda N.count(x): prebrojava koliko se puta element x pojavljuje u n-torci N (slika 70).

Metoda N.index(x): vraća indeks prvog elementa n-torke N čija je vrijednost x (slika 70).

```
>>> n_torka=(1,2,6,7,6,6,4,5)
>>> n_torka.count(6)
3
>>> n_torka.index(6)
2
>>>
```

Slika 70. Metode nad n-torkama

U slučaju potrebe korištenja ostalih metoda nad n-torkama koriste se dvije ugrađene funkcije: funkcija „list()“ koja prihvaća n-torku i vraća listu s identičnim elementima, te funkcija „tuple()“, koja prihvaća listu i vraća n-torku. Navedene funkcije veoma su korisne za rješavanje „TypeError“-a koji uključuje liste i n-torke[60].

[60] Spector P. *Introduction to Python Programming Course Notes* [online], 2005:49-50. Dostupno na: https://www.stat.berkeley.edu/~spector/python.pdf [15.04.2019].

```
>>> n_torka=(1, 6, "marelica", "oblak", 7.6)
>>> n_torka.append(4)
Traceback (most recent call last):
  File "<pyshell#318>", line 1, in <module>
    n_torka.append(4)
AttributeError: 'tuple' object has no attribute 'append'
>>> n_torka1=list(n_torka)
>>> n_torka1.append(4)
>>> print(n_torka1)
[1, 6, 'marelica', 'oblak', 7.6, 4]
>>> n_torka1=tuple(n_torka1)
>>> print(n_torka1)
(1, 6, 'marelica', 'oblak', 7.6, 4)
>>> n_torka1[0]=5
Traceback (most recent call last):
  File "<pyshell#324>", line 1, in <module>
    n_torka1[0]=5
TypeError: 'tuple' object does not support item assignment
>>> n_torka1=list(n_torka1)
>>> n_torka1[0]=5
>>> n_torka1=tuple(n_torka1)
>>> print(n_torka1)
(5, 6, 'marelica', 'oblak', 7.6, 4)
>>>
```

Slika 71. Metode nad „n-torkama“

Na gore navedenom primjeru (slika 71) prikazano je da upotreba ostalih metoda nad n-torkom nije moguća osim ako nije prethodno konvertirana u listu.

5. Razlike, ograničenja i upotreba lista i n-torki

Razlike i ograničenja: glavnu razliku između liste i n-torke, osim same sintakse, predstavlja svojstvo (ne)promjenjivosti. Slučaj gdje je bolje upravljati nepromjenjivom strukturom podataka bio bi slučaj manipulacije osjetljivim podatcima koji se ne bi trebali mijenjati u programu. S druge strane, prednost promjenjivih struktura podataka jest dinamično upravljanje skupovima podataka. Moguće ih je stvarati i nadopunjavati novim elementima „u mjestu“ te s vremenom i izbrisati određene elemente. U ovakvom slučaju, tip podataka trebao bi biti promjenjiv. No, s obzirom na ugrađene funkcije „list()“ i „tuple()“, moguće je konvertirati tipove podataka iz jednog tipa u drugi, tj. pomoću ovih funkcija moguće je stvoriti n-torku iz liste i suprotno. U slučaju rada s n-torkom i potrebe za listom zbog potrebe ažuriranja određenih elemenata, funkcija „list()“ pokazala se veoma korisnom. Pri radu s listom i potrebom korištenja nepromjenjive strukture podataka (n-torke) u određenoj funkciji gdje je manipulacija podatcima s druge strane nepoželjna, funkcija „tuple()“ mogla bi se svakako pokazati korisnom[61].

Dodavanje elemenata: dodavajući element(e) listi, Python interpreter izvršava navedenu naredbu „u mjestu“, no da bi dodali element n-torci, potrebno je za svako dodavanje individualnih elemenata stvoriti novu n-torku što je znatno sporiji proces nego kod lista[62].

Varijable i reference: u sljedećem primjeru (slika 72) prikazan je problem kopiranja lista pri pridruživanju varijabli.

```
>>> a=[1,3,5,7,9]
>>> b=a
>>> b[1]=10
>>> a
[1, 10, 5, 7, 9]
>>>
```

Slika 72. Varijable i reference

Pomoću izraza „b=a“ objekt liste se ne kopira. Zapravo se dvije varijable („a“ i „b“) referenciraju na isti objekt liste. Prethodno stvorena varijabla „a“ „drži“ lokaciju objekta

[61] Chun, W. *Core Python Programming*, 2006: 200.
[62] Elghamrawy K. *Python:What is the difference between a List and a Tuple?*, [online]. Dostupno na: https://www.afternerd.com/blog/difference-between-list-tuple/ [19.04.2019].

(liste) u memoriji, te izrazom „b=a“ kopira se adresa lokacije objekta (ali ne i pravi objekt) na varijablu „b“. Kao rezultat postoje dvije reference („a“ i „b“) na isti objekt (listu). Drugim riječima, pri ispisu izraza „b[1]=10“ dobije se isti efekt kao ispis izraza „a[1]=10“, što se u pravilu ne može dogoditi s nepromjenjivim strukturama podataka. U ovako malim isječcima koda to ne predstavlja problem, no u slučaju većeg projekta s mnogo referenci, ova osobina promjenjive strukture podataka predstavljala bi potencijalno problem[63].

Upotreba: liste se mogu koristiti za pohranu vrijednosti istog i različitog tipa podataka (npr. imena, brojeve ili druge sekvence). Kroz liste je moguće iterirati te time provoditi iste naredbe nad svakim elementom unutar liste, što je posebno korisno kod pohrane promjenjivih, fleksibilnih podataka. Liste mogu, primjerice, sakupljati atribute (kategorije) o ljudima na pozicijski slijedan način[64].

```
>>> Marko=["Marko Marić", 40, 5000, "vozač"]
>>> Ana=["Ana Anić", 35, 7500, "farmaceut"]
>>> Marko[0]
'Marko Marić'
>>> Marko[0].split()[-1]
'Marić'
>>> Ana[2]*=1.25
>>> Ana[2]
9375.0
>>>
```

Slika 73. Upotreba lista (primjer)

Na gornjem primjeru (slika 73) prikazana je upotreba liste. Baza podataka neke firme sadrži Marka i Anu, dvoje radnika navedene firme na različitim radnim pozicijama. Svaka lista sadrži informacije o imenu i prezimenu radnika, starosti, iznosu plaće te radnom mjesto. Potrebno je saznati kako se Marko preziva i povećati Aninu plaću za 25%. Izrazom „Marko[0].split() [-1]“ prvo se dijeli string „Marko Marić“ na području razmaka (nije specificirano drugačije u zagradama) te se indeksom „[-1]“ izražava želja za ispisom drugog dijela stringa (Markovo prezime). Izrazom „Ana[2]*=1.25“ Anina je plaća od 7500 kn

[63] Elghamrawy K. *Python:What is the difference between a List and a Tuple?*, [online]. Dostupno na: https://www.afternerd.com/blog/difference-between-list-tuple/ [15.04.2019].
[64] Lutz, M. *Programming Python: Powerful Object-Oriented Programming*, 2010: 4-5.

uvećana za 25% (9375 kn). Za implementaciju slične baze mogu poslužiti i imenovane n-torke.

```
>>> from collections import namedtuple
>>> prvi=namedtuple("Radnici", ["Ime", "Prezime", "Godina", "Plaća",
"Radno_mjesto"])
>>> Marko=prvi("Marko", "Marić", 40, 5000, "vozač")
>>> Ana=prvi("Ana", "Anić", 35, 7500, "farmaceut")
>>> Marko.Prezime
'Marić'
>>> Ana.Plaća
7500
>>> Ana[3]*=1.25
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    Ana[3]*=1.25
TypeError: 'Radnici' object does not support item assignment
>>>
```

Slika 74. Primjena n-torke (primjer)

Na gore navedenom primjeru (slika 74) prikazan je isti primjer samo uz upotrebu imenovane n-torke. Dohvaćanje podataka imenovane n-torke čine lakšim – da bi Markovo prezime bilo dohvaćeno ne treba se pamtiti indeks podatka već ime atributa („Prezime“). Međutim, povišica Anine plaće ne može se ažurirati u bazi jer promjene podataka unutar n-torke nisu moguće.

N-torke obično pohranjuju heterogene podatke[65]. N-torke su obično korištene u slučajevima gdje izraz ili korisnički definirana funkcija može sigurno pretpostaviti da se kolekcija vrijednosti (n-torka) koja je korištena neće mijenjati (Swaroop, 2013: 65). Zbog nepromjenjive prirode n-torki, mogu se koristiti za pohranu različitih tipova podataka o određenom objektu. N-torka se, uz slučaj što se ponekad treba koristiti kako bi neki kod funkcionirao, koristi pri upravljanju osjetljivim podatcima. N-torke se također koriste umjesto liste u slučaju kada vrijednosti kolekcije nisu namijenjene promjenama, što može zaštititi

[65] Jackson, C. *Learning to Program Using Python* [online], 2009: 52-53. Dostupno na: https://www.ida.liu.se/~732A47/literature/PythonBook.pdf [15.04.2019].

podatke od slučajne modifikacije[66]. N-torke se također mogu koristiti kao ključevi u rječnicima i u formatiranju stringova, dok liste ne mogu[67].

[66] Sturtz, J. (2018) *Lists and Tuples in Python* [online]. Dostupno na: https://realpython.com/python-lists-tuples/ [06.04.2019].
[67] Jackson, C. *Learning to Program Using Python* [online], 2009: 52-65. Dostupno na: https://www.ida.liu.se/~732A47/literature/PythonBook.pdf [15.04.2019].

6. Zaključak

U programskom jeziku Python za prikaz povezanih podataka koriste se strukture podataka. Dvije vrlo često korištene strukture podataka su liste i n-torke. S jedne strane imamo liste koje su promjenjive strukture podataka. Liste predstavljaju dinamičnu slijednu kolekciju objekata varijabilne duljine koje je moguće nadopunjivati i mijenjati „u mjestu“. S druge strane postoje n-torke. N-torke predstavljaju statičnu slijednu kolekciju objekata unaprijed poznate duljine koje nije moguće mijenjati „u mjestu“, te da bi dobili n-torku koja sadrži već postojeće i nove podatke, potrebno je izraditi novu strukturu. Uz zajednička svojstva ovih struktura podataka moguće je većinu navedenih funkcija i metoda koristiti nad obje strukture (većina metoda uz prethodnu pretvorbu n-torki u liste po potrebi). Liste se koriste pri prikazu, pristupanju i manipulaciji podatcima promjenjive prirode, dok n-torke koristimo za prikaz i pristupanje nepromjenjivim podatcima koji su na taj način zaštićeni od neželjene modifikacije.

7. Literatura

1. Beazley, David M. (2009) *Python Essential Reference.* V.4. Addison-Wesley Professional.

2. Budin, Leo et al. (2015) *Rješavanje problema programiranjem u Pythonu / udžbenik za prirodoslovno-matematičke gimnazije.* V.4. Zagreb: Element.

3. Wesley Chun. (2001) *Core Python Programming.* V.1. New Jersey: Prentice Hall Professional.

4. Downey, Allen B. (2012) *Think Python.* V.1. Sebastopol: O'Reilly Media, Inc.

5. Dunđer, Ivan. (2018) *Programiranje / udžbenik.* V.1.3. Zagreb.

6. Elghamrawy K. *Python: What is the difference between a List and a Tuple?,* [online]. Dostupno na: https://www.afternerd.com/blog/difference-between-list-tuple/ [15.04.2019].

7. Halterman, Richard L. (2011) *Learning to program using Python* [online]. Dostupno na: https://www.cs.uky.edu/~keen/115/Haltermanpythonbook.pdf [22.04.2019].

8. Hetland, Magnus L. (2017) *Beginning Python: From Novice to Professional.* V.3. New York: Springer Science+Business Media.

9. Hünniger, Dirk et al. (2015) *Python Programming* [online]. Dostupno na: https://upload.wikimedia.org/wikipedia/commons/9/91/Python_Programming.pdf [22.04.2019].

10. Goodrich, Michael T., Tamassia, Roberto i Goldwasser, Michael H. (2013) *Data Structures & Algorithms in Python.* New Jersey: John Wiley & Sons, Inc.

11. Jackson, Cody. (2009) *Learning to Program Using Python* [online]. Dostupno na: https://www.ida.liu.se/~732A47/literature/PythonBook.pdf [22.04.2019].

12. Kalafatić, Zoran et al. (2016) *Python za znatiželjne: Sasvim drukčiji pogled na programiranje.* V.1. Zagreb: Element.

13. Kuhlman, Dave. (2013) *A Python Book: Beginning Python, Advanced Python, and Python Exercises* [online]. V. 1.3a. Dostupno na: http://www.davekuhlman.org/python_book_01.pdf [22.04.2019].

14. Lambert, Kenneth A. (2010) *Fundamentals of Python: From First Programs through Data Structures.* Boston: Course Technology.

15. Necaise, Rance D. () *Data structures and algorithms using Python* [online]. New Jersey: John Wiley & Sons, Inc. Dostupno na: http://home.ustc.edu.cn/~huang83/ds/Data%20Structures%20and%20Algorithms%20Using%20Python.pdf [22.04.2019].

16. Lott, Steven F. (2010) *Building Skills in Python: A Programmer's Intrduction to Python* [online]. Dostupno na: http://www.itmaybeahack.com/book/python-2.6/html/index.html [22.04.2019].

17. Lutz, Mark (2013) *Learning Python.* V.5. Sebastopol: O'Reilly Media, Inc.

18. Lutz, Mark (2010) *Programming Python: Powerful Object-Oriented Programing.* V.4. Sebastopol: O'Reilly Media, Inc.

19. Rossum, Guido. (2016) *Python Tutorial* [online]. V.3.5.1.Delaware: Python Software Foundation. Dostupno na: https://dev.rbcafe.com/python/python-3.5.1-pdf/tutorial.pdf [22.04.2019].

20. Rossum, Guido i Drake, Fred L. (2003) *Python Language Reference Manual* [online]. V.2.3. Bristol: Network Theory Limited. Dostupno na: http://knuth.luther.edu/~bmiller/CS151/Spring05/pythonref.pdf [22.04.2019].

21. Spector, Phil. (2005) *Introduction to Python Programming Course Notes* [online]. Dostupno na: https://www.stat.berkeley.edu/~spector/python.pdf [22.04.2019].

22. Stojanović, Aleksandar (2012) *Elementi računalnih programa: s primjerima u Pythonu i Scali.* V.1. Zagreb: Element.

23. Sturtz, John. (2018) *Real Python: Lists and Tuples in Python* [online]. Dostupno na: https://realpython.com/python-lists-tuples/ [22.04.2019].

24. Summerfield, Mark. (2009) *Programming in Python 3: A Complete Introduction to the Python Language.* V.2. Boston: Pearson Education, Inc.

25. Swaroop, C. H. (2013) *A Byte of Python* [online]. Dostupno na: http://files.swaroopch.com/python/python3/byte_of_python.pdf [22.04.2019]..

8. Popis slika

9. Sažetak

Ovaj rad bavi se opisom struktura podataka, preciznije jednom od nepromjenjivih struktura podataka (n-torkama) te jednom od promjenjivih struktura podataka (listom) u programskom jeziku Python te njihovim osnovnim karakteristikama. Navedene su specifične karakteristike za obje strukture podataka, način stvaranja pojedine strukture podataka, njihov zapis te operacije i metode koje se mogu primijeniti nad navedenim strukturama podataka. Također je opisano u kojim slučajevima se određene vrste struktura podataka koriste, za koju svrhu, na koji način, kada je primjereno koristiti promjenjive (liste), a kada nepromjenjive (n-torke) strukture podataka te koja su ograničenja kod uporabe navedenih struktura podataka. Navedeni su određeni primjeri pomoću kojih se može stvoriti jasnija slika o navedenim strukturama podataka i o načinu njihovog korištenja te o tome kako se definiraju. Ideja rada je definirati razlike, prednosti i mane ovih naizgled sličnih struktura podataka te opisati adekvatne načine njihove uporabe koristeći pri tome interpretacijski programski jezik Python.

Ključne riječi: Python, N-torka, Lista, Strukture podataka

Summary

This paper deals with the description of data structures, more precisely with one of the immutable data structures (tuples) and one of the mutable data structures (list) in the programming language Python and their basic characteristics as well. Specific characteristics are being listed for both data structures, also the ways of creating an individual data structure, their notation and operations and methods which can be applied on the previously stated data structures. It is also described in which cases are certain types of data structures used, for what purpose, in what way, when it is appropriate to use mutable (lists) and when immutable (tuples) data structures and what are the usage limitations for both aforementioned data structures. Certain examples are stated by which one can get a clearer picture about the stated data structures, the way they are used and their definition. The idea of this paper is to define the differences, advantages and flaws of these seemingly similar data structures and to describe adequate ways of their application while using the interpreted programming language Python.

Keywords: Python, Tuple, List, Data structures