*M. Essert, I. Vazler:*

# Python - osnove

**Odjel za matematiku Sveučilišta u Osijeku**

*Osijek, 2007.*

# *Sadržaj*

# 1. Python interpreter

Python je interpreterski, interaktivni, objektu orjentirani programski jezik, kojeg je 1990. godine prvi razvio Guido van Rossum. Već do konca 1998., Python je imao bazu od 300.000 korisnika, a od 2000. već su ga prihvatile ustanove kao MIT, NASA, IBM, Google, Yahoo i druge.
Python ne donosi neke nove revolucionarne značajke u programiranju, već na optimalan način ujedinjuje sve najbolje ideje i načela rada drugih programskih jezika. On je jednostavan i snažan istodobno. Više nego drugi jezici on omogućuje programeru više razmišljanja o problemu nego o jeziku. U neku ruku možemo ga smatrati hibridom: nalazi se između tradicionalnih skriptnih jezika (kao što su Tcl, Schema i Perl) i sistemskih jezika (kao što su C, C++ i Java). To znači da nudi jednostavnost i lako korištenje skriptnih jezika, uz napredne programske alate koji se tipično nalaze u sistemskim razvojnim jezicima.
Python je besplatan (za akademske ustanove i neprofitnu upotrebu), open-source software, s izuzetno dobrom potporom, literaturom i dokumentacijom.

## 1.1 Jezične značajke

*Interpretacija međukôda*

Python kôd živi u tekst datotekama koje završavaju na *.py*. Program kompilira kôd u niz *bytecode-ova* koji se spremaju u *.pyc* datoteke koje su prenosive na bilo koje platforme gdje se mogu izvoditi interpretacijom tog međukôda. Slično se izvršava Java kôd. Brzina izvođenja Python kôda istog je reda veličine kao u Javi ili Perlu. Python je napisan u ANSI C i raspoloživ za cijeli niz strojeva i operacijskih sustava uključujući Windows, Unix/Linux i Macintosh.

*Jezik visoke razine*

Osim standardnih tipova podataka (brojevi, nizovi znakova i sl.) Python ima ugrađene tipove podataka visoke razine kao što su liste, n-terci i rječnici.

*Interaktivnost*

Python se može izvoditi u različitim okruženjima. Za razvitak programa najbolji je interaktivni način rada u kojem se programski kôd piše naredbu za naredbom. Ne postoji razlika u razvojnom i izvedbenom (engl. runtime) okolišu.

*Čista sintaksa*

Sintaksa jezika je jednostavna i očevidna. Uvlake zamjenjuju posebne znakove za definiranje blokova kôda, pa je napisani program vrlo pregledan i jednostavan za čitanje.

*Napredne značajke jezika*

Python nudi sve značajke očekivane u modernom programskom jeziku: objektu orijentirano programiranje s višestrukim nasljeđivanjem, dohvaćanje izuzetaka, redefiniranje standardnih operatora, pretpostavljene argumente, prostore imena i pakete.

*Proširivost*

Python je pisan u modularnoj C arhitekturi. Zato se može lako proširivati novi značajkama ili API-ima. (eng. application programming interface).

*Bogate knjižnice programa*

Pythonova knjižnica (engl. *library*), koja uključuje standardnu instalaciju, uključuje preko 200 modula, što pokriva sve od funkcija operacijskog sustava do struktura podataka potrebnih za gradnju web-servera. Glavni Python web site (www.python.org) nudi sažeti index mnogih Python projekata i različitih drugih knjižnica.

*Potpora*

Python ima veliku entuzijastičku zajednicu korisnika koja se svake godine udvostručuje.

## 1.2 Izvođenje Python programa

Python kôd može se izvoditi na više načina. Najčešći je u interaktivnom radu, pozivom:

```
% python
```

nakon čega se pišu naredbe jedna za drugom. Interepreter izvodi naredbu i ispisuje rezultat:

```
>>> print 'Ovo je interaktivni rad'
'Ovo je interaktivni rad'
```

U Windows okruženju postoji više grafičkih rješenja za interaktivan rad. S Pythonom u instalacijskom paketu dolazi IDLE kojeg je razvio Python-ov autor. IDLE koristi Tkinter GUI framework i prenosiv je na sve Python platforme koje imaju Tkinter potporu.
Bolja rješenja od IDLE-a su PythonWin i PyScripter koja korisniku daju puno grafičkih rješenja za jednostavnu Python upotrebu kako u interaktivnom tako i u skriptnom radu.

*Skriptni rad*

Programi se spremaju u skripte s pomoću običnog text editora ili Python orjentiranog grafičkog okruženja, a onda se kao Unix skripte pozivaju iz sistemske linije.

*Programski moduli*

Moduli s Python naredbama izvode se iz sistemske jezgre pozivom python interpreter zajedno s modulom:

```
% python moj_modul.py
```

*Umetnuti (embeded) kôd*

Iako se češće unutar Pythona mogu pozivati funkcije iz drugih programa (npr. C-a za slučaj programskog ubrzanja), moguće je Python kôd u obliku izvornih tekst naredbi izvoditi i unutar drugog programa, koristeći Python runtime API:

```
#include <Python.h>
. . .
Py_Initialize();
PyRun_SimpleString("x = a + 2*pi");
```

## 2. Tipovi podataka

Računalni program je algoritam zadan programskim naredbama koje se izvršavaju nad nekom *vrstom* ili *tipom* (binarno spremljenih) *podataka*. Sve podatčane vrijednosti u Pythonu predstavljene su objektima, a svaki *objekt* ima svoj tip ili vrstu. *Literal* označuje vrijednost podatka koja se direktno pojavljuje u programskoj naredbi:

```
2347                    # Cjelobrojni literal
13.514                  # Realni (Floating-point) literal
5.0J                    # Imaginarni literal
'hello'                 # String literal, niz znakova
```

Osim alfanumeričkih znakova, Python koristi i posebne simbole i to kao međaše (graničnike) (npr. za početak i završetak string literala koristi simbole jednostrukih ili dvostrukih navodnika) ili kao simbole aritmetičko-logičkih i odnosnih (relacijskih) operatora. Simbol '#' koristi se za početak komentara i ne obrađuje se od strane Python interpretera.

Koristeći literale i međaše, mogu se također stvarati podaci drugih tipova, npr.

```
[ 63, 'faks', 8.6 ]     # Listina, lista ili popis
( 450, 320, '600' )     # n-terac (tuple)
{ 'a':72, 'b':1.4 }     # rječnik (dictionary)
```

Tipovi objekata mogu se složiti u kategorije (tablica 2.1), pa razlikujemo brojeve, nizove, klase, datoteke, module i sl. Neki objekti mogu se mijenjati (npr. liste), a neki ne mogu (npr. stringovi).

**Tablica 2.1. Ugrađeni (built-in) tipovi u Python programskom jeziku**

| Kategorija tipa podataka | Ime tipa podatka | Opis |
|---|---|---|
| Prazno (None) | NoneType | 'null' objekt |
| Brojevi | IntType<br>LongType<br>FloatType<br>ComplexType | Cijeli broj<br>Dugi cijeli broj<br>Realni broj s pom. zarezom<br>Kompleksni broj |
| Nizovi | StringType<br>UnicodeType<br>ListType<br>TupleType<br>XRangeType<br>BufferType | Niz znakova (string)<br>Unicode (string)<br>Listina, popis ili lista<br>n-torka<br>Vraćeno iz xrange()<br>Vraćeno iz buffer() |
| Preslikavanje | DictType | Rječnik |
| Klase, razredi | ClassType | Definicija klase |
| Instanca klase | InstanceType | Stvaranje instance klase |
| Datoteka | FileType | Datoteka – podaci na mediju |
| Moduli | ModuleType | Modul (skup objekata) |
| Mogu se pozvati (Callable) | BuiltinFunctionType<br>BuiltinMethodType<br>ClassType<br>FunctionType<br>InstanceType<br>MethodType<br>UnboundMethodType | Ugrađene funkcije<br>Ugrađene metode<br>Objekt klase<br>Korisnička funkcija<br>Instanca objekta klase<br>Ograničena (bound) metoda<br>Neograničena metoda klase |

| | | |
|---|---|---|
| Nutarnji tipovi | CodeType | Byte-compilirani kôd |
| | FrameType | Izvedbeni okvir |
| | TracebackType | Složaj slijeda izuzetaka |
| | SliceType | Tip kriške (odlomka) |
| | EllipsisType | U proširenim odlomcima (extended slices) |

Python program pristupa vrijednostima podataka preko referenci. Referenca je ime koje se odnosi na neku specifičnu lokaciju u memoriji na kojoj je vrijednost (objekt) spremljena. Reference poprimaju oblike varijabli, atributa i članova (items). Jedna ili više Python programskih naredbi može se spremiti u *funkciju*. Funkcija može imati ulazne argumente i vraćati izlazne vrijednosti.

Provjera tipa nekog objekta ostvaruje se pozivom funkcije `type()`:

```
>>> type("Dobro došli u Python svijet!")  # string literal
<type 'str'>
>>> type(512) # numerički literal
<type 'int'>
>>> k=2.178   # varijabla k
>>> type(k)
<type 'float'>
>>> type ({ 'a':72, 'b':1.4 })
<type 'dict'>
>>> z=2+3j    # varijabla z
>>> type(z)
<type 'complex'>
```

Objekti (varijable, funkcije, klase i dr.) mogu se spremati u module, a moduli u pakete. Modul se u memoriju učitava s pomoću naredbe '`import`'.

```
import sys                  # učitava se modul sistemskih funkcija
```

Pojedinačni podatak ili funkcija iz modula dohvaća se naredbom '`from ... import ...`':

```
from math import sin, cos   # učitavaju se samo sin i cos funkcije
```

Dohvaćanje vrijednosti podatka preko atributa objekta postiže se sintaksom '`objekt.atribut`'. Objekt može imati više atributi. U slučaju da je atribut neka funkcija, odgovarajuća sintaksa je '`objekt.atribut()`'.

```
>>>import math
>>> print math.pi, math.sin(2.3)
3.14159265359 0.745705212177
```

Dohvaćanje vrijednosti podatka preko članova, često se naziva *indeksiranje* i odnosi se samo na strukturirane tipove podataka (liste, stringove i sl.). Indeksacija počinje s nultim (0) indeksom.

```
>>> x=[1, 3, 9, 16]         # lista s četiri podatka
>>> print x[3]              # dohvaćanje trećeg podatka
16
```

Vrijednost reference može se doznati pozivom funkcije `id()`. Pridruživanjem objekata referenca se ponavlja, tj. objekti se ne dupliciraju.

```
>>> a=123
>>> id(a)
3695888
>>> b=a       # pridružba b sa a; objekt se ne kopira, samo adresa
>>> b
123
>>> id(b)     # b kao i a pokazuje na istu memorijsku adresu
3695888
>>> a=a+1     # stvoren je novi objekt
>>> print 'a=',a,'  b=',b
a= 124   b= 123
>>> print 'id(a)=',id(a),'  id(b)=',id(b)        # ne pokazuju isto!
id(a)= 3695876   id(b)= 3695888
```

*Izraz* (engl.expression) je kombinacija vrijednosti (literala), varijabli i operatora. Vrijednost izračunatog izraza ispisuje se na zaslon računala korištenjem naredbe *print*:

```
>>> print 'gruba aproksimacija pi = ', 22./7
gruba aproksimacija pi =  3.14285714286
```

Programske naredbe Pythona temelje se na pridruživanju (objekata referencama), upravljanju tijekom programa (if, else,...), programskim petljama (for, while, ...) i pozivima funkcija i klasa. Funkcije obuhvaćaju jednu ili više naredbi. Postoji velik broj ugrađenih (built-in) funkcija, a ostale stvara korisnik. Funkcije, zajedno s podacima, grupiraju se u klase. Funkcije definirane unutar klasa, zovu se *metode.*

*Identifikator* je ime objekta, tj. ime varijable, funkcije, klase i sl. Identifikatori ne smiju koristiti neku od 29 ključnih riječi Python-a (tablica 2.2.), jer su one pridružene osnovnim Python naredbama.

**Tablica 2.2. Python ključne riječi**

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

## 2.1 Brojevi

Ugrađeni brojevni objekti u Pythonu podržavaju cijele brojeve (obične i dugačke), brojeve s pomičnim zarezom (realne brojeve), i kompleksne brojeve. Objekti brojeva u Pythonu su nepromjenljivi (unmutable) objekti, što znači da bilo kakva aritmetička operacija na brojevnom objektu, uvijek stvara novi brojevni objekt.

```
>>> a=1234
>>> id(a)
19431452
>>> a=a+0
>>> id(a)
18681652
>>> print a
1234
```

Literali cijelih brojeva mogu biti decimalni, oktetni, ili heksadecimalni. Decimalni literal je predstavljen nizom znamenki gdje je prva znamenka različita od nule. Oktetni literal je određen s početnom 0 iza koje ide niz oktetnih znamenki (0 do 7). Na sličan način heksadecimalni literal koristi početni niz 0x nakon čega slijedi niz heksadecimalnih znamenki (0 do 9 i A do F bilo velikim ili malim slovom). Na, primjer:

```
1,  23,  3493              #Decimalni cijeli brojevi
01,  027,  06645           #Oktetni cijeli brojevi
0x1,  0x17,  0xda5         #heksadecimalni cijeli brojevi
```

Bilo kojem literalu cijelog broja može može se dodati slovo 'L' ili 'l' kako bi se označio dugački cijeli broj (long integer). Na primjer:

```
1L,  23L, 99999333493L             #Dugački decimalni cijeli brojevi
01L,  027L,  01351033136165L       #Dugački oktetni cijeli brojevi
0x1L,  0x17L,  0x17486CBC75L       #Dugački heksadec. cijeli brojevi
```

Razlika između dugačkog i običnog cijelog broja je u tome što dugački cijeli broj nema predodređenu numeričku granicu; može biti toliko dug koliko računalo ima memorije. Običan cijeli broj uzima nekoliko okteta memorije i ima minimalnu i maksimalnu vrijednost koju diktira arhitektura stroja. sys.maxinit je najveći dostupni običan cijeli broj, dok je sys.maxinit-1 najveći negativni.

```
>>> print sys.maxint # za uobičajeni stroj najveći cijeli broj je
2147483647
>>> 2L**500   # 2 na 500-tu potenciju
32733906078961418700131896968275991522166420460430647894832913680961337964046745548832700923259041571508866841275600710092172565458853930533285275893 76L
```

Realni literal (broj s pomičnim zarezom) predstavljen je nizom decimalnih znamenki koje uključuju decimalni zarez, tj. točku (.), exponent (e ili E, te + ili - iza, s jednom ili više znamenki na kraju), ili oboje. Vodeći znak decimalnog literala ne smije biti e ili E, a može biti bilo koja znamenka ili točka (.). Na primjer:

```
0.,  0.0,  .0,  1.,  1.0,  1e0,  1.e0, 1.0e0
```

Pythonova decimalna vrijednost odgovara uobičajena 53 bita preciznosti na modernim računalima.

Kompleksni broj sastavljen je od dviju decimalnih vrijednosti, jedne za realni, a druge za imaginarni dio. Moguće je pristupiti dijelovima kompleksnog objekta z kao samo-čitajućim „read-only“ atributima z.real i z.imag. Imaginarni literal dobije se dodavanjem znaka 'j' ili 'J' realnom literalu:

```
0j,  0.j,  0.0j,  .0j,  1j,  1.j,  1.0j,  1e0j,  1.e0j,  1.0e0j
```

Znak J (ili j) na kraju literala označuje kvadratni korijen od -1, što je uobičajena oznaka imaginarnog dijela u elektrotehničkoj praksi (neke druge discipline koriste znak 'i' u tu svrhu, ali Python je izabrao znak j).

Treba primijetiti da brojevni literali ne uključuju predznak: ako postoji + ili – ispred broja, onda su to posebni operatori.

## 2.2 Nizovi

Niz je spremnik (engl. container) članova (engl. items) koji se indeksiraju ili dohvaćaju ne-negativnim cijelim brojevima. Python pruža tri ugrađene (engl. built-in) vrste nizova za stringove (obične i Unicode), n-terace, i liste. Knjižnički i ekstenzijski moduli pružaju druge vrste nizova, a korisnik također može sam napisati svoje. Nizovi se mogu obrađivati na više načina.

### 2.2.1 Stringovi – nizovi alfanumeričkih znakova

Ugrađeni objekt string je poredan skup znakova koji se koristi za skladištenje i predstavljanje podataka na tekstovnoj bazi. Nizovi znakova u Pythonu su nepromjenljivi (eng. immutable), što znači da se novom operaciijom na nizu znakova, uvijek proizvede novi niz, a ne modificira stari. Objekti stringa imaju ugrađeno više metoda.

Literalni niz znakova može biti pod navodnicima jednostrukim, dvostrukim ili trostrukim navodnicima. String u navodnicima je niz od nula ili više znakova unutar identičnih znakova navodnika. Na primjer:

```
'Ovo je string literal'
"Ovo je još jedan string literal"
```

Dvije različite vrste navodnika imaju identičnu funkciju. Mogu se koristiti tako da apostrofiramo string unutar stringa, što je često jednostavnije nego apostrofirati string upotrebom posebnog znaka (\' za jednostruki ili \“ za dvostruki navodnik):

```
' jel\' me netko tražio?'          # eksplicitni navodnik u stringu
" jel' me netko tražio?"           # Na ovaj način je čitljivije
```

Ako se string želi prikazati u više linija, ona da na koncu svake stavlja znak lijeve kose crte (\):

```
"Ovo je prva, \
a ovo druga linija istog stringa"  # Komentar nije dopušten na
                                   # liniji sa znakom \
```

U stringu se dakako mogu umetati i posebni znakovi (\n za novu liniju, \t za tabulator i sl.), ako se takav niz želi programom ispisivati:

```
"Ovo je prva, \n\
a ovo druga linija istog stringa"  # za
```

Drugi pristup je uporaba stringa s trostrukim navodnicima, koji se dobiju trostrukim ponavljanjem jednostrukih (''') ili dvostrukih navodnika (""").

```
"""A ovo je jedan duuugi string
koji se proteže na više linija,
u ovom slučaju na tri"""           # Komentar dopušten samo na kraju
```

U ovakvom literalu stringa s tri navodnika, automatski su sačuvani novi redovi, pa se njihovi kontrolni znakovi ne trebaju dodavati u niz. Nije dopuštena ni upotreba nekih kontrolnih (tzv. 'escape') znakova (tablica 2.3), kao na primjer znaka lijeve kose crte (engl. backslash; \\)

**Tablica 2.3. 'Escape' znakovi**

| Niz | Meaning | ASCII/ISO code |
|---|---|---|
| \<novi_red> | Konac linije se zanemaruje | Nema ga |
| \\ | Kosa crta ulijevo, backslash | 0x5c |
| \' | Jednostruki navodnik | 0x27 |
| \" | Dvostruki navodnik | 0x22 |
| \a | Zvono, bell | 0x07 |
| \b | Brisanje ulijevo, backspace | 0x08 |

| | | |
|---|---|---|
| \f | Nova stranica, form feed | 0x0c |
| \n | Nova linija, newline | 0x0a |
| \r | Skok u novi red, carriage return | 0x0d |
| \t | Tabulator, tab | 0x09 |
| \v | Vertikalni tabulator | 0x0b |
| \0ooo | Oktalna vrijednosti ooo (\0000 do \377) | kako je zadano |
| \xhh | Heksadecimalna vrijednost hh (\x00 do \xff) | kako je zadano |
| \uhhh | Unicode vrijednosti | Samo za Unicode str. |

Unicode je novi standard za pisanje znakova. Za razliku od ASCII standarda, novi standard uključuje sve znakove iz gotovo svih svjetskihjezika. Unicode literalni string ima istu sintaksu kao obični literalni string uz dodatak znaka 'u' ili 'U' koji se piše odmah ispred početnog navodnika. Unicode literalni nizovi znakova mogu koristiti '\u' iza kojeg sljede četiri heksadecimalne znamenke koje opisuju Unicode znak.

```
>>> a=u'str\xf6m gr\xfcn'
>>> print a
ström grün
```

Više string literala bilo koje vrste napisanih u slijedu, compiler će povezati u jedan string objekt.

```
>>> print 'koliko' 'je' 'tu' 'stringov' u'\xe4' '?'
kolikojetustringovä?
```

### 2.2.2 N-terac

N-terac je nepromjenljivi niz članova. Članovi u n-tercu su bilo koji objekti, istih ili različitih tipova. N-terac se definira nabrajanjem objekata odvojenih zarezima (,). Zadnjem članu u nizu takodjer se može dodati zarez. N-terac sa samo jednim članom mora imati zarez na kraju, jer inače gubi tip n-terca. Prazan n-terac je označen s praznim parom zagrada. Članovi se mogu grupirati, pa nastaju ugnježdeni n-terci.

```
(100, 200, 300)      # N-terac s tri člana
(3.14,)              # N-terac sa samo jednim članom
(    )               # Prazan n-terac
```

Za generiranje n-terca, osim nabrajanjem, moguće je pozvati i ugrađenu funkciju `'tuple()'`. Ako je x neki niz, onda `tuple(x)` vraća n-terac s članovima jednakima članovima niza x.

```
>>> x='abrakadabra'
>>> tuple(x)
('a', 'b', 'r', 'a', 'k', 'a', 'd', 'a', 'b', 'r', 'a')
>>> y='sezame'
>>> (x,y)
('abrakadabra', 'sezame')
```

### 2.2.3 Lista

Lista, listina ili popis je promjenljiv poredani niz članova objekata. Članovi u listi su bilo kakvi objekti različitih vrsta. Lista se definira nabrajanjem članova, odijeljenih zarezima (,) i smještenih unutar uglatih zagrada ([ ]). Dopušteno je iza zadnjeg člana liste, ostaviti još ejdan zarez. Prazna lista se označava praznim parom uglatih zagrada. Evo nekih primjera:

```
[42, 3.14, 'zdravo' ]       # Lista s tri člana
[123]                       # Lista s jednim članom
['a', [-45j, 'b'], 4.5]     # ugnježđena lista s tri člana
[ ]                         # Prazna lista
```

Na sličan način, kao i s generiranjem n-teraca, moguće je pozvati prikladnu funkciju `'list()'` za generiranje listi. Na primjer:

```
>>> a='ovo'
>>> b='je'
>>> c='lista'
>>> d=[a,b,c]        # tvorba liste nabrajanjem članova
>>> print d
['ovo', 'je', 'lista']
>>> list(d)          # tvorba liste pozivom funkcije
['ovo', 'je', 'lista']
>>> list(a)          # tvorba liste pozivom funkcije
['o', 'v', 'o']
>>> type(d)
<type 'list'>
>>> type(a)
<type 'str'>
```

Treba primjetiti kako se tvorba listi preko `list()` funkcije uvijek realizira nad pripadnim tipom objekta.

### 2.2.4 Temeljne operacije i metode s nizovima

Dohvaćanje elementa bilo kojeg niza (stringa, n-terca, liste) postiže se indeksiranjem. Dio niza, odlomak ili kriška (engl. slice) dobiva se sintaksom `'i:j'` gdje je `'i'` početni indeks, a `'j'` završni indeks kriške (tablica 2.4.). Dužina niza dobiva se pozivom funkcije `len()`, a maksimalni i minimalni član niza s funkcijama `max()`, odnosno `min()`.

**Tablica 2.4. Operacije i metode nad svim nizovima**

| Član | Opis |
|---|---|
| *s*`[`*i*`]` | Vraća element *i* u nizu *s* |
| *s*`[`*i*`:`*j*`]` | Vraća kršku – niz elemenata od i-tog do j-tog indeksa |
| `len(`*s*`)` | Vraća broj elemenata u *s* |
| `min(`*s*`)` | Vraća minimalni elemenat iz *s* |
| `max(`*s*`)` | Vraća maksimalni elemenat iz *s* |

Promjenljivi nizovi (liste) imaju mogu mijenjati članove ili kriške članova odjednom, kao i brisati članove i skupine članova.

```
>>> a=(1,3,5,7,9)
>>> print a[0], a[3]
1 7
>>> b='ovo je string'
>>> print b[9],b[0],b[-1]
r o g
>>> c=[7,'marko',-5,'kompleksni']
>>> print c[3],c[1]
 kompleksni marko
>>> print len(a),len(b),len(c)
5 13 4
>>> print max(a), max(b), max(c)
9 v marko
>>> print min(a), min(b), min(c)
1   -5
>>> print a[1:3],b[7:12],c[0:2]
(3, 5) strin [7, 'marko']
>>>
```

Treba primjetiti kako se dohvaćanje članova preko indeksa u kriškama ostvaruje od početnog indeksa do konačnog, ali koji se pritom isključuje, ne uzima u obzir. Negativan indeks pak dohvaća članove od kraja niza. Tako je '-1' indeks za zadnji član, '-2' za predzadnji i tako dalje.

Budući da se u kategoriji nizova samo liste mogu mijenjati direktno, postoje pridružbe članovima liste i funkcije brisanja članova (tablica 2.5.). Obje operacije mogu se izvesti nad pojedinačnim i skupnim članovima liste.

**Tablica 2.5. Operacije i metode nad promjenljivim nizovima**

| Član | Opis |
|---|---|
| `s[i] = v` | Pridružba članu na i-tom mjestu |
| `s[i:j] = t` | Pridružba skupini članova |
| `del s[i]` | Brisanje člana |
| `del s[i:j]` | Brisanje skupine članova |

```
>>> lista=['tko zna','bilje','siroko mu','polje']
>>> lista[1]='bolje'
>>> lista
['tko zna', 'bolje', 'siroko mu', 'polje']
>>> lista[1:3]=[',zna!']
>>> lista
['tko zna', ',zna!', 'polje']
>>> del lista[-1]
>>> lista
['tko zna', ',zna!']
```

## 2.3 Rječnik

Preslikavanje (engl. mapping) je skup objekata indeksiranih s pomoću gotovo slobodnih vrijednosti koje se zovu ključevi (engl. keys). Tako nastali objekti su promjenljivi, a za razliku od nizova, nisu poredani.

Python nudi jednu vrstu preslikavanja, riječničku vrstu (engl. dictionary). Knjužnički i ekstenzijski moduli pružaju još vrsta preslikavanja, a druge može načiniti korisnik sam. Ključevi u riječniku mogu biti različitih tipova, ali moraju biti jednoznačni (engl. hashable). Vrijednosti u riječniku su također objekti i to mogu biti različitih tipova. Član u rječniku je par ključ/vrijednost (engl. key/value). O rječniku se možete razmišljati kao o asocijativnom polju.

Eksplicitno stvaranje rječnika provodi se nizom parova ključ:vrijednost odvojenih zarezima, koji se smještaju unutar vitičastih zagrada. Dopušten je i zarez nakon zadnjeg člana. Ako se ključ pojavljuje više od jednom u rječniku, samo se jedan od članova s tim ključem sprema, jer ključ mora biti jedincat. Drugim riječima, rječnici ne dozvoljavaju duplikate ključeva. Prazan se rječnik označava praznim parom zagrada. Evo nekih riječnika:

```
{'x':42, 'y':3.14, 'z':7}  # Rječnik s tri člana i string ključevima
{1: 2, 3:4}          # Rječnik s dva člana i cjelobrojnim ključevima
{ }                  # Prazan rječnik
```

Tvorbu rječnika moguće je izvesti i s pomoću ugrađene funkcije `dict()`. Na primjer:

```
>>> dict([['a',12],['b',54]])
{'a': 12, 'b': 54}
>>> dict(a='zagreb', d='ogulin', e='Osijek')
{'a': 'zagreb', 'e': 'Osijek', 'd': 'ogulin'}
>>> dict([[12,'akumulator'],['baterija',4.5]])
{'baterija': 4.5, 12: 'akumulator'}
```

`dict( )` bez argumenata stvara i vraća prazan riječnik. Ako se ključ pojavljuje više nego jednom u popisu (argumentima funkcije *dict*), samo će se posljednji član s tim kjučem zadržati u rezultirajućem rječniku.

## 2.4 Ugrađene metode nizova i rječnika

Budući da su nizovi i preslikavanja objekti, a svaki objekt ima članove (varijable) i metode (funkcije), korisno je nabrojati (tablica 2.6.-2.8.) i iskušati ugrađeno.

**Tablica 2.6. String metode**

| Metoda | Opis |
|---|---|
| *s* `.capitalize()` | Pretvara po slovo od *s* u veliko slovo. |
| *s* `.center(`*width*`)` | Centrira string upolju duljine *width* . |
| *s* `.count(`*sub* `[,`*start* `[,`*end* `]])` | Broji pojavljivanja podstringa *sub* u stringu *s.* |
| *s* `.encode([`*encoding* `[,`*errors* `]])` | Vraća kodiranu inačicu stringa |
| *s* `.endswith(`*suffix* `[,`*start* `[,`*end* `]])` | Provjerava kraj stringa za suffix. |
| *s* `.expandtabs([`*tabsize* `])` | Proširuje tabulatore praznim mjestima |
| *s* `.find(`*sub* `[,`*start* `[,`*end* `]])` | Pronalazi prvo pojavljivanje zadanog podstringa *sub* . |
| *s* `.index(`*sub* `[,`*start* `[,`*end* `]])` | Pronalazi prvo pojavljivanje zadanog podstringa *sub* uz podizanje izuzetka, ako ga nema |
| *s* `.isalnum()` | Provjerava jesu li svi znakovi alfanumerički |
| *s* `.isalpha()` | Provjerava jesu li svi znakovi alfabetski. |
| *s* `.isdigit()` | Provjerava jesu li svi znakovi znamenke. |
| *s* `.islower()` | Provjerava jesu li svi znakovi pisani malim slovima. |
| *s* `.isspace()` | Provjerava jesu li svi znakovi praznine. |

| | |
|---|---|
| *s* .istitle() | Provjerava jeli string pisan kao naslov (prvo slovo svake riječi napisano velikim slovom). |
| *s* .isupper() | Provjerava jesu li svi znakovi pisani velikim slovima. |
| *s* .join(*t* ) | Povezuje stringove u listi *t* koristeći *s* kao međaš. |
| *s* .ljust(*width* ) | Lijevo poravnanje *s* u stringu duljine *width* . |
| *s* .lower() | Vraća *s* pretvoren u string s malim slovima. |
| *s* .lstrip() | Odstranjuje prazna mjesta ispred stringa. |
| *s* .replace(*old* , *new* [,*maxreplace* ]) | Zamjenjuje podstring *old* sa *new* . |
| *s* .rfind(*sub* [,*start* [,*end* ]]) | Nalazi zadnji pojavak podstringa *sub* |
| *s* .rindex(*sub* [,*start* [,*end* ]]) | Nalazi zadnji pojavak podstringa *sub* ili javlja izuzetak |
| *s* .rjust(*width* ) | Desno poravnanje *s* u stringu duljine *width*. |
| *s* .rstrip() | Odstranjuje prazna mjesta iza stringa. |
| *s* .split([*sep* [,*maxsplit* ]]) | Dijeli string koristeći *sep* kao međaš. *maxsplit* je naveći broj dijeljenja koji će se izvršiti. |
| *s* .splitlines([*keepends* ]) | Dijeli string u listu linija. Ako je *keepends* jednak 1, čuvaju se kontrolni znakovi novih redaka. |
| *s* .startswith(*prefix* [,*start* [,*end* ]]) | Provjerava da li string započinje s *prefix*-om . |
| *s* .strip() | Odstranjuje prazna mjesta i ispred i iza stringa. |
| *s* .swapcase() | Vraća velika slova za string malih sloa i obratno. |
| *s* .title() | Vraća verziju stringa kao naslova. |
| *s* .translate(*table* [,*deletechars* ]) | Mijenja string koristeći transformacijsku tablicu znakova. |
| *s* .upper() | Vraća string pretvoren u velika slova. |

**Tablica 2.7. Metode liste**

| Metoda | Opis |
|---|---|
| *li*.append(*x* ) | Dodaje novi element x na kraj liste *li*. |
| *li*.extend(*t* ) | Dodaje novu listu *t* na kraj liste *li*. |
| *li*.count(*x* ) | Broji pojavke od*x* u listi *li*. |
| *li*.index(*x* ) | Vraća najmanji *i* za koji je *s[i] == x* . |
| *li*.insert(*i*,*x* ) | Umeće *x* na indeksu *i* . |
| *li*.pop([*i* ]) | Vraća element *i* i briše ga iz liste. Ako se *i* izostavi, onda se vraća zadnji element. |
| *li*.remove(*x* ) | Traži *x* i briše ga iz liste *li*. |
| *li*.reverse() | Reverzira (obrće) članove liste *li* na mjestu. |
| *li*.sort([*cmpfunc* ]) | Sortira (slaže) članove liste *li* na mjestu. *cmpfunc* je funkcija za usporedbu |

**Tablica 2.8. Metode i operacije tipova preslikavanja (rječnika)**

| Član | Opis |
|---|---|
| *di* [*k* ] | Vraća član od *di* s ključem *k* . |
| *di* [*k* ] = *x* | Postavlja *di* [*k* ] na *x* . |
| del *di* [*k* ] | Briše *di* [*k* ] iz *di* . |
| *di* .clear() | Briše sve članove iz *di* . |
| *di* .copy() | Vraća kopiju od *di* . |

| | |
|---|---|
| *di* .has_key(*k* ) | Vraća 1 ako *di* ima ključ *k* , a 0 inače. |
| *di* .items() | Vraća listu od (*key* ,*value* ) parova. |
| *di* .keys() | Vraća listu od vrijednosti ključeva. |
| *di* .update(*b* ) | Dodaje sve objekte iz rječnika *b* u *di* . |
| *di* .values() | Vraća listu svih vrijednosti spremljenih u *di* . |
| *di* .get(*k* [,*v* ]) | Vraća *di*[*k* ] ako nađe; inače vraća *v* . |
| *di* .setdefault(*k* [, *v* ]) | Vraća *di*[*k* ] ako nađe; vraća *v* ipostavlja *di*[*k* ]=*v* . |
| *di* .popitem() | Vraća slučajne (*key* ,*value*) parove kao e-torke iz *di*. |