o mjestu gdje su navedeni), a drugi je tzv. tabulator, za ljepše formatiranje stringa u vertikalne stupce (odgovara tipki TAB na tipkovnici).

2.5 Nizovi podataka

Definicija 2.8 — Niz podataka.
Niz podataka 'S' je spremnik (eng. container) poredanih članova (eng. items) koji se indeksiraju ili dohvaćaju s pomoću cijelog broja 'n' kao S[n].

Python pruža tri ugrađene (eng. *built-in*) vrste nizova: *stringove* (*obične* i *Unicode*), *n-torke* (ili *n-terce* - ako ćemo ih zvati muškim rodom), *liste*. Stringovi i n-terci su nepromjenjivi objekti, dok su liste promjenjivi objekti.

2.5.1 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 (engl. *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:

Zaslon 2.13 — String-literal.
```
>>> 'Ovo je string literal'
>>> "Ovo je novi string literal"
```

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. Ako se string želi prikazati u više linija, može se koristiti trostruki navodnici, koji se dobiju trostrukim ponavljanjem jednostrukih (''') ili dvostrukih navodnika (""").

Zaslon 2.14 — String-dugi.
```
>>> s="Ovo je prva, \n a ovo druga linija istog stringa"
>>> print s
Ovo je prva,
 a ovo druga linija istog stringa

>>> """A ovo je jedan duuugi string
koji se proteze na vise linija,
u ovom slucaju na tri"""        # Komentar dopusten 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.

Unicode je novi standard za pisanje znakova. Za razliku od ASCII standarda, novi standard uključuje sve znakove iz gotovo svih svjetskih jezika. 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 slijede četiri heksadecimalne znamenke koje opisuju Unicode znak.

Zaslon 2.15 — String-Unicode.
```
>>> 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.

Zaslon 2.16 — String-povezivanje.
```
>>> print 'koliko' 'je' 'tu' 'stringov' u'\xe4' '?'
kolikojetustringovä?                    # Prazna lista
```

2.5.2 Lista

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

Zaslon 2.17 — Lista-primjeri.
```
>>> [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
```

Lista je promjenjivi objekt, tj. može joj je mijenjati broj članova kao i njezine vrijednosti.

Zaslon 2.18 — Lista-promjenjivi objekt.
```
>>> f=[1,2,44,"fgh"]
>>> id(f)
2800051244L
>>> f.append(67)
>>> f
[1, 2, 44, 'fgh', 67]
>>> id(f)
2800051244L
>>> g=f    #kopiranje adrese
>>> id(g)
2800051244L
>>> g.append(345)
>>> g
[1, 2, 44, 'fgh', 67, 345]
>>> f
[1, 2, 44, 'fgh', 67, 345]
>>> id(g)
2800051244L
>>> id(f)
2800051244L
>>> z=g[:]   #kopiranje liste
```

```
>>> z
[1, 2, 44, 'fgh', 67, 345]
>>> id(z)
2800051148L
>>> g.append(11)
>>> g
[1, 2, 44, 'fgh', 67, 345, 11]
>>> f
[1, 2, 44, 'fgh', 67, 345, 11]
>>> z
[1, 2, 44, 'fgh', 67, 345]
>>>
```

Iz ovog primjera ključno je uočiti da lista $[1,2,44,"fgh"]$ ne mijenja svoju memorijsku adresu i nakon dodavanja novog elementa 67. Pridružba $g = f$ kopira memorijsku adresu liste iz varijable f u varijablu g, ali lista ostaje i dalje na istoj adresi, tako da sada obje varijable pokazuju na istu listu. Bitno je primjetiti da pridružba $g = f$ ne znači da će se ovime stvoriti nova lista na nekoj drugoj adresi, nego da se samo kopira njena adresa. Da bi se kopirala cijela lista iz g na neko drugo mjesto može se koristiti $z = g[:]$, pri čemu ta nova lista nije povezana s listom na koju referenciraju varijable g i f.

2.5.3 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.

Zaslon 2.19 — N-terac- tip.

```
>>> (100, 200, 300) # N-terac s tri clana
>>> (3.14,)         # N-terac sa samo jednim clanom
>>> (    )          # 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`.

Zaslon 2.20 — N-terac - generiranje.

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

Kada se n-terac jednom definira, broj članova se ne može naknadno mijenjati. Ako su članovi n-terca nepromjenjivi objekti, onda se takvi n-terci često koriste kao ključevi u rječnicima.

2.5.4 Indeksiranje

Indeksiranje u Pythonu počinje od nule (prvi član u `S` je `S[0]`). Ako `S` ima `L` članova, indeks `n` smije biti `0`, `1`, . . , sve do `L-1` uključno, ali ne i više. Indeks `n` može također biti `-1`, `-2`, . . . ,

sve do `-L` (isključno), ali ne i manji. Negativni `n` označava iste članove u `S` kao što čini `L+n`. Drugim riječima, `S[-1]` je posljednji element od `S`, `S[-2]` je pretposljednji, itd.

Zaslon 2.21 — Indeksiranje niza.

```
>>> niz=u"Ali ja vatru samo slušat umijem"
>>> niz[0]
u'A'
>>> niz[-1]
u'm'
>>> lista=['I', u'čudnovata', 'spopada', 'me' ,'sjeta']
>>> lista[1]
u'\u010dudnovata'
>>> type(lista[1])
<type 'unicode'>
>>> lista[-2]
'me'
>>> torka=(u'što', 'njezin', 'jezik', 'ne', 'razumijem')
>>> torka[2], torka[4]
('jezik', 'razumijem')
>>>
```

Indeksiranje koristimo i u slučaju promjene nekog elementa u nizu, ali samo kod lista, jer su stringovi i n-torke nepromjenljivi (eng. inmutable) objekti. Promjena stringa izvodi se stvaranjem novog s dijelom starih i novih elemenata.

Zaslon 2.22 — Promjenljivi i nepromjenljivi nizovi.

```
>>> lista=['I', u'čudnovata', 'spopada', 'me' ,'sjeta']
>>> lista[3]='nas'
>>> lista
['I', u'\u010dudnovata', 'spopada', 'nas', 'sjeta']
>>> niz=u"Ali ja vatru samo slušat umijem"
>>> niz[0]='I'
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: 'unicode' object does not support item assignment
>>> niz='I'+u'li ja vatru samo slušat umijem'
>>> niz
u'Ili ja vatru samo slu\u0161at umijem'
>>>
```

Elementi liste i n-torke mogu biti bilo kojeg tipa, ne samo stringa, kao u gornjem primjeru. Dohvaćanje članova takodjer ne mora biti pojedinačno, već može u nizu, po kriškama (eng. *slice*).

2.5.5 Kriške

Podniz `S` može se označiti i/ili dohvatiti s pomoću kriške, uporabom sintakse `S[i:j]`, gdje su `i` i `j` cijeli brojevi. `S[i:j]` je podniz `S` od `i`-tog člana do `j`-tog člana, ali ne uključujući `j`-ti član. Bilo koji ili oba indeksa smiju biti manji od nule. Negativni indeks označuje mjesto u nizu s obzirom na kraj niza, kao što je to u indeksiranju pokazano.

Treba primjetiti kako u Pythonu svi dosezi uključuju donju, a isključuju gornju granicu. Kriška može biti prazan podniz ako je `j` manje od `i` ili ako je `i` veće ili jednako `L`, duljini niza

S. Ako je j jednako nuli, i se može izostaviti, a ako se ide do konca niza (L) onda se i j smije izostaviti. Cijeli niz S može se dakle indeksirati sa S[:].

Zaslon 2.23 — Kriška niza.

```
>>> niz[7:23]
u'vatru samo slu\u0161a'
>>> print niz[7:24]
vatru samo slušat
>>> lista[-3:-1]
['spopada', 'me']
>>> torka[1:]
('njezin', 'jezik', 'ne', 'razumijem')
>>>
```

Kriška može biti zadana i korakom, uporabom sintakse S[i:j:k], gdje je k također cijeli broj i predstavlja korak unutar kriške.

Zaslon 2.24 — Kriška niza.

```
>>> niz[0:10:1]
u'Ali ja vat'
>>> niz[0:10:2]
u'Aij a'
>>> niz[10:0:-1]
u'rtav aj il'
>>> niz[-21:0:-1]
u'rtav aj il'
>>> niz[-20::-1]
u'rtav aj ilA'
>>>
```

2.5.6 Operatori nizova

Nizovi istog tipa nadovezuju se jedan na drugog s pomoću operatora '+'. Isto tako moguće je množiti bilo koji niz S s cijelim brojem n s pomoću operatora '*'. Rezultat S*n ili n*S je povezivanje n kopija od S. Ako je n nula ili manje od nule, rezultat je prazan niz istog tipa kao S.

Operator *in* u x in S provjerava je li objekt x jednak bilo kojem članu u nizu S. Ako jest, vraća True, a False ako nije. Slično tomu, x not in S operator je isto kao not(x in S).

Zaslon 2.25 — Operatori nizova.

```
>>> niz1=['O', 'moja', 'je', 'leđa', 'lagano']
>>> niz2=['krcula', 'mandolina']
>>> niz1+niz2
['O', 'moja', 'je', 'le\xc4\x91a', 'lagano', 'krcula', 'mandolina']
>>> niz2*2
['krcula', 'mandolina', 'krcula', 'mandolina']
>>> 'lagano' in niz1
True
>>> 'mandolina' in niz1
False
>>> 'lagano' in niz1 and 'mandolina' in niz2
True
```

```
>>>
```

Primjer je pokazan s listom, a isto bi se mogao pokazati i s n-torkom ili stringom. Elementi stringa su uvijek znakovi, dok kod liste i n-torke mogu biti bilo kojeg tipa.

Korištenje indeksa većeg ili jednakog duljini niza `L` ili manjeg od `-L`, javlja pogrešku, tj. podiže (izaziva) *iznimku* (eng. *exception*).

2.6 Rječnik - preslikavanje

Definicija 2.9 — Rječnik.
Skup objekata indeksiranih imenovanim indeksima (tzv. ključevima), umjesto cijelih brojeva (kao kod nizova podataka), zove se rječnik. U rječniku uvijek razlikujemo informaciju u paru: ključ i pripadnu vrijednost.

Ključevi i vrijednosti u rječniku mogu biti različitih tipova. N-torku para u rječniku zove se član (eng. *item*) i često piše kao *ključ/vrijednost* (eng. *key/value*). O rječniku se može razmišljati kao o asocijativnom polju koje se ne indeksira brojem, nego najčešće stringom.

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 označuje parom praznih vitičastih zagrada. Tvorbu rječnika moguće je izvesti i s pomoću ugrađene funkcije *dict()*:

Zaslon 2.26 — Rječnik.
```
>>> rj1={u'Cesarić':'Skrivena bol',u'Pupačić':'More'}
>>> rj1[u'Pupačić']
'More'
>>> dict(imenica=u'priča', prijedlog='o', im2='Vukovaru')
{'imenica': u'pri\u010da', 'im2': 'Vukovaru', 'prijedlog': 'o'}
>>> dict([[12,'akumulator'],['baterija',4.5]])
{'baterija': 4.5, 12: 'akumulator'}
```

Zaslon 2.27 — Rječnik-ponavljanje ključeva.
```
>>> D={'auto':5,'motor':8,'prozor':'novi','motor':10}
>>> D
{'prozor': 'novi', 'auto': 5, 'motor': 10}
```

Zaslon 2.28 — Rječnik u rječniku.
```
>>> rj={'miro':{'ri':55,'kk':788},'stef':'bure'}
>>> rj
{'miro': {'kk': 788, 'ri': 55}, 'stef': 'bure'}
>>> rj.keys()
['miro', 'stef']
>>> rj.values()
[{'kk': 788, 'ri': 55}, 'bure']
>>> rj['miro']
{'kk': 788, 'ri': 55}
>>> rj['miro']['kk']
```

```
788
>>> rj['jozo']='super'
>>> rj
{'jozo': 'super', 'miro': {'kk': 788, 'ri': 55}, 'stef': 'bure'}
```

Može se uočiti da se dodavanje novog para u postojeći rječnik izvelo na način **rj['jozo']='super'**.

2.7 Skup

Skup je neporedan niz jedincatih (neponavljajućih) elemenata. Elementi moraju biti jednoznačni, engl. *hashable*. Tvorbu skupa moguće je izvesti pomoću ugrađene funkcije *set()*:

Zaslon 2.29 — Skup.

```
>>> set((2,7,4,"haha"))
set(['haha', 2, 4, 7])
```

Da skup ne može sadržavati duplikate, može se provjeriti primjerom:

Zaslon 2.30 — Skup-dupliciranje.

```
>>> set((2,7,4,"haha",7,"haha",357))
set(['haha', 2, 4, 357, 7])
```

Dodavanje novog člana u postojeći skup može se izvesti odabirom metodu *add()*:

Zaslon 2.31 — Skup-dodavanje.

```
>>> S=set((4,7,8))
>>> id(S)
156273548
>>> S.add('rtz')
>>> id(S)
156273548
```

Može se uočiti da je skup promjenjiv objekt, jer je nakon dodavanja novog člana memorijska adresa ostala nepromijenjena.

3 — Programski tijek

Upravljanje tijekom programskog izvođenja je redoslijed po kojem se programski kôd izvršava. Ovo upravljanje u Python programima, kao i u drugim programskim jezicima, temelji se na *uvjetnim naredbama*, *petljama* i *pozivima funkcija*.

3.1 Programska petlja

Dvije su osnovne programske petlje u Pythonu: *while* i *for* petlja.

3.1.1 Naredba `while`

Naredba `while` u Pythonu izvršava naredbu ili blok naredbi niti jedan, jedan ili više puta, ovisno o ispunjenju upravljanog uvjetnog izraza. Ovo je njena sintaksa:

Definicija 3.1 — While.

```
while izraz:
      naredba(e)
```

`While` naredba također može uključivati naredbu `break` unutar tijela petlje ili naredbu `else` u vrlo rijetkim slučajima.

Primjer petlje `while` :

Zaslon 3.1

```
>>> z=5
>>> i=0
>>> while i<z:
...     print 2+2
...     i+=1
...
4
4
4
4
```

```
4
>>>
```

Petlja *while* izvršavala se dokle god je uvjet $i < z$ bio zadovoljen. To znači za $i = 0,1,2,3,4$.

Program 3.1 Program broji samoglasnike u riječi.

```
1  samoglasnici=['a','e','i','o','u']
2  i=0
3  koliko = 0
4  stih="""Jer tko će ostati ako se svi odreknemo sebe i
5  pobjegnemo u svoj strah? Kome ostaviti grad?"""
6  while i<len(stih):
7      if stih[i] in samoglasnici:
8          koliko+=1
9      i+=1
10 print u'U stihu ima ',koliko,' samoglasnik.'
```

što kao rezultat daje:

Zaslon 3.2

```
U stihu ima  31  samoglasnik.
>>>
```

Prvo se izračuna izraz poznat kao uvjet petlje (eng. *loop condition*). Ako uvjet nije istinit (*false*), naredba `while` završava i niti jedna od naredbi u tijelu petlje se ne izvršava. Ako je pak uvjet petlje zadovoljen, naredba ili naredbe od kojih se sastoji tijela petlje se izvršavaju. Kada tijelo petlje završi s izvršavanjem, uvjet se ponovno izračunava, da se vidi treba li se izvršiti nova iteracija. Ovaj se proces nastavlja sve dok uvjet petlje ne postane neistinit, nakon čega `while` naredba završava. U slučaju gornjeg programa varijabla *i* je došla do konca stringa '*stih*' i za svako slovo koje je samoglasnik, povećala brojilo za 1. Tijelo petlje (s vidljivom uvlakom!) treba sadržavati kôd koji će u nekom trenutku učiniti petljin uvjet lažnim, ili petlja nikada neće završiti, osim ako se ne podigne iznimka (pogreška) ili tijelo petlje ne izvrši naredbu `break`. U ovom slučaju vrijednost varijable *i* raste do broja znakova stiha, a preko tog broja izraz '`i<len(stih)`' postaje `False`.

3.1.2 Naredba `for`

Naredba `for` u Pythonu ostvaruje također iterativno izvršavanje naredbe ili bloka naredbi, a upravlja se preko iteracijskog izraza. Sintaksa za naredbu `for` je:

Definicija 3.2 — For.

```
for varijabla in nizu:
      naredba(e)
```

Treba primjetiti da je ključna riječ `in` dio sintakse naredbe `for`. Ona ne odgovara operatoru `in` koji se koristi pri ispitivanju članova u nizovima (je li neki član element niza ili nije). Ključna riječ 'in' pridružuje varijabli jedno za drugim sve članove (elemente) niza. Ako je niz lista, onda varijabla poprima sve elemente te liste, a ako je niz string, onda varijabla poprima sve znakove tog niza. Pod pojmom znaka nisu uključena samo slova, nego i razmaci, znakovi interpunkcije i posebni znakovi. Naredba `for` također može uključivati naredbu `break` i u rijetkim slučajima odlomak `else`.

Tipičan primjer uporabe naredbe `for` je:

Zaslon 3.3
```
>>> for i in ['i', 'pa', 'te', 'ni', 'niti']:
...     print i
...
i
pa
te
ni
niti
>>>
```

Iako se često unutar petlje nalaze složenije obradbe ili grananja tijeka programa (npr. preko naredbe *if*), kao što to pokazuje ovaj jednostavan primjer:

Program 3.2 Ispisuju se slova zadane riječi koja su leksikografski ispred slova 'm'.

```
1 print "ASCII kod slova 'm' je:", ord('m')
2 for x in 'Pobratimstvo lica u svemiru':
3     if x <= 'm':
4         print "'%s':%d" % (x,ord(x)),
```

što daje:

Zaslon 3.4
```
ASCII kod slova "m" je: 109
'P':80 'b':98 'a':97 'i':105 'm':109 ' ':32 'l':108 'i':105
'c':99 'a':97 ' ':32 ' ':32 'e':101 'm':109 'i':105
>>>
```

Iteracijski niz u ovom slučaju je string, pa varijabla `x` u svakom prolazu, iteraciji, poprima vrijednost pojedinog člana tog niza, u ovom slučaju pojedinačnog slova. U ovom primjeru unutar petlje izvršava se samo jedna naredba, naredba *if* koja uključuje *print* u slučaju istinite provjere. Primjetite da su sva ispisana slova leksikografski ispred slova '*m*' jer im je ASCII (brojni) ekvivalent manji. I praznina (eng. *space*) ima svoj brojni ekvivalent, broj 32. Sva velika slova abecede ispred su (imaju manje brojeve) od malih slova abecede.

Naredba ili naredbe od kojih se sastoji tijelo petlje izvršavaju se jedna po jedna za svaki član unutar iteratora (osim ako petlja završi uslijed podizanja iznimke ili izvršenja naredbe `break` ili `return`). Na mjestu varijable (u ovom slučaju s imenom 'x') može biti više njih, jer je dopušteno višestruka pridružba koju dakako moraju podržati i članovi niza. U slučaju npr. triju varijabli (*x,y,z*) i elementi liste moraju biti n-torke s tri člana. Po istom načelu dohvaćaju se i elementi rječnika, koristeći funkciju *items()* koja iz rječnika dohvaća parove (ključ i vrijednost):

Zaslon 3.5
```
>>> for k,v in {'ana':3, 'marko':5, 'miki':4}.items():
...     print k,v
...
marko 5
ana 3
miki 4
>>>
```

Kad se naredba petlje završi, upravljačke varijable (u ovom slučaju *k,v*) ostaju povezane na posljednju vrijednost na koju ih je naredba petlje povezala.

Zaslon 3.6
```
>>> print k,v
miki 4
>>>
```

3.2 Sažete liste

Česta svrha `for` petlje je provjeravanje svakog člana unutar niza i tvorba nove liste dodajući rezultate izraza izračunatog na jednom ili svim članovima koji se provjeravaju. Oblik izraza, zvan *sažeta lista* (eng. *list comprehension*) omogućuje točno i neposredno izvršavanje ove zadaće. Kako je sažeta lista izraz (a ne blok naredbi), može se ga koristiti izravno (npr. kao stvarni argument u funkciji poziva, zatim u naredbi `return` ili kao podizraz za neki drugi izraz). *Sažeta lista* ima sljedeću sintaksu:

```
[izraz  for  cilj  in  iter-clanovi  lc-odlomci]
```

gdje su `cilj` i `iter-clanovi` identični kao i u običnoj `for` naredbi. Izraz se može ograditi i okruglim, oblim zagradama, pa će u tom slučaju predstavljati n-terac, a ne listu. `lc-odlomci` je niz od nula ili više odlomaka, od kojih je svaki sličan obliku:

```
for cilj in iter-clanovi
   if lc-odlomci
```

`Izraz` unutar svake `if` klauzule ima istu sintaksu kao `izraz` u običnoj `if` naredbi.

Sažeta lista je ekvivalentna petlji `for` koja gradi istu listu ponavljajućim pozivima `append` metode rezultantne liste. Na primjer:

```
result1 = [x+1 for x in neki_niz]
```

jednak je kao sljedeća `for` petlja:

```
result2 = []
for x in neki_niz
    result2.append (x+1)
```

Sažeta lista koji koristi `if` odlomak izgleda ovako:

Zaslon 3.7
```
>>> [x*2 for x in range(10) if x>4.5 ]
[10, 12, 14, 16, 18]
```

a to je identično `for` petlji koja sadrži naredbu `if`:

Zaslon 3.8
```
>>> lis=[]
>>> for x in range(10):
...     if x>4.5:
...         lis.append(x*2)
...
>>> lis
```

```
[10, 12, 14, 16, 18]
>>>
```

Jedan složeniji primjer kod uporabe dviju `for` petlje i `if` naredbe:

Zaslon 3.9

```
>>> for x in (0,1,2,3):
...     for y in (0,1,2,3):
...         if x < y:
...             print (x,y,x+y)
...
(0, 1, 1)
(0, 2, 2)
(0, 3, 3)
(1, 2, 3)
(1, 3, 4)
(2, 3, 5)
>>>
```

može se zamijeniti sa sažetom listom:

Zaslon 3.10

```
>>> [(x, y, x + y) for x in (0,1,2,3) for y in (0,1,2,3) if x < y]
[(0, 1, 1), (0, 2, 2), (0, 3, 3), (1, 2, 3), (1, 3, 4), (2, 3, 5)]
>>>
```

Kao što ovi primjeri pokazuju, redoslijed naredbi `for` i `if` u *sažetoj listi* jednak je kao u ekvivalentnoj petlji, ali u sažetoj listi gnježdenje operacija ostaje implicitno.

3.3 Programski izbor - grananje tijeka

3.3.1 Naredba *if*

Često se neka naredba ili niz (blok) naredbi treba izvršiti samo u slučaju ako je neki uvjet zadovoljen. Ponekad je to izvršavanje ovisno o nekoliko međusobno povezanih uvjeta. Pythonova kombinirana naredba `if`, koja koristi proširenja `elif` ili `else` zaključak, služi za takvo uvjetno izvršavanje naredbi. Sintaksa izgleda ovako:

Definicija 3.3 — If.

```
if   uvjet(i):
       naredba(e)
elif uvjet(i):
     naredba(e)
elif   uvjet(i):
       naredba(e)
...
else   uvjet(i):
      naredba(e)
```

gdje `uvjet(i)` predstavlja jedan ili više uvjetnih izraza povezanih odnosnim (relacijskim) operatorima.

Proširenja naredbe '`if` ' s '`elif` ' i '`else`' su dopuštene u slučaju ispitivanja više različitih uvjeta. '`Elif`' je kratica od '*else-if*', što znači '*inače-ako*', čime se prvo ispitivanje uvjeta

proširuje na iduće. Ako niti jedan od uvjeta nije zadovoljen, onda se izvode naredbe iza '`else`' dijela, ako on postoji. Treba primjetiti kako, za razliku od nekih jezika, Python nema naredbu `switch`, pa se moraju koristiti `if`, `elif` ili `else` za sve uvjetne obrade.

Zaslon 3.11

```
>>> if 2==2:
...     print 3+2
...
5
>>> if 2==3:
...     print 3+2
...
>>>
```

Ovaj jednostavan primjer pokazuje da ako je uvjet istinit, izvršava se naredba *print*.

Kada u 'ako' ili 'inače' odlomku ima više naredbi (tj. programsko grananje se odnosi na čitav blok naredbi), naredbe se, kao što je već napomenuto, pišu na zasebnim linijama koje su uvučene, pomaknute udesno od linije početka. Blok naredbi završava kada se pozicioniranje linije vrati na ono od početka odlomka (ili još više ulijevo od toga).

Kada postoji samo jedna jednostavna naredba, onda se ona može pisati na istoj naredbenoj liniji neposredno iza dvotočja ('*:*') kojim se završava jedan ili više uvjetnih ispitivanja. Dakako, moguće je isto napisati i u idućem retku, ali se onda mora koristiti uvlaka.

Zaslon 3.12

```
>>> rijec="IHJJ"
>>> duljina=len(rijec) ## koliko znakova ima rijec
>>> if duljina==4: print 'Dugi smo 4 slova :-)!'
...
Dugi smo 4 slova :-)!
>>>
```

Primjetite kako se operator usporedbe jednakosti sastoji od dva znaka '=', a ne jednog (koji se koristi za pridružbu, vidi 5.1). Za uvjet u `if` ili `elif` odlomku može se koristiti bilo koji Python-ov izraz. Pritom se vrijednost izraza promatra u Bool-ovom kontekstu, gdje se svaka vrijednost uzima kao istinita, ili neistinita. Svaki broj koji nije nula ili niz znakova koji nije prazan, isto kao i n-terac, lista ili rječnik, izračunava se kao istinito (`True`). Nula (bilo kojeg brojevnog tipa), `None` i prazni stringovi, n-terci, liste i rječnici, izračunavaju se kao neistina (`False`). To znači, ako smo u gornjem primjeru za operator usporedbe uzeli znak '=', umjesto '==', rezultat provjere bi uvijek bio ispunjen jer se varijabli '*duljina*' pridružuje broj (*4*) koji je različit od ništice.

Ako se izraz za `if` odlomak izračuna kao istinit, naredbe koje slijede `if` odlomak će se izvršiti i cijela naredba `if` završava. Ako se izraz za `if` odlomak izračuna kao neistinit, onda se izvršavaju izrazi za iduće `elif` odlomke, ako su njihovi uvjeti ispunjeni. Za prvu `elif` klauzulu za koju je uvjet istinit, izvršavaju se sve narede koje je slijede i čitava naredba `if` time završava. U protivnom, ako niti jedan `elif` odlomak nije izvršen, jer uvjeti nisu bili zadovoljeni, onda se izvršavaju naredbe odlomka `else`, dakako, ako on postoji.

Program 3.3 Ispisuju broj slova pojedine fonologijske kategorije.

```
1 SONANTI = u'vjl\u01C9rmn\u01CC'
2 FRIKATIVI = u'fszšžh'
```

```
AFRIKATE = u'cčćđ\u01C6'
o=s=f=a=i=0
niz=u'Noćas se moje čelo žari , noćas se moje vjeđe pote'
for x in niz:
    if x in SONANTI:
        s+=1
    elif x in FRIKATIVI:
        f+=1
    elif x in AFRIKATE:
        a+=1
    else:
        i+=1
print 'sonanti=',s,', frikativi=',f,', afrikate=',a,', ostalo='
    ,i
```

što daje:

Zaslon 3.13

```
sonanti= 9 , frikativi= 5 , afrikate= 4 , ostalo= 31
>>>
```

Usporedimo sada sljedeći primjer s primjerom 3.4. Umjesto sa stringovima ovdje se radi s brojevima. Želi se iz zadane liste brojeva odrediti koji su parni, koji neparni ili primarni, te koji su svi ostali. Pritom su zadane tri liste brojeva s kojima će se uspoređivati oni iz zadane liste.

Program 3.4 Uspoređuje i ispisuje količine i vrijednosti parnih, neparnih i primarnih brojeva iz niza zadanih brojeva.

```
Neparni=[1,3,5,7,9,11,13,15,17,19,21]
Primarni=[2,3,5,7,11,13,17,19,23]
Parni=[2,4,6,8,10,12,14,16,18,20]
s=f=a=i=0
Su=[] # inicijalizacija liste neparnih brojeva
Fu=[] # inicijalizacija liste parnih brojeva
Au=[] # inicijalizacija liste primarnih brojeva
Iu=[] # inicijalizacija liste ostalih brojeva
Brojevi=[1,2,3,4,5,18,19,2j,1+5j,2.4] # lista zadanih brojeva

for x in Brojevi:
    if x in Neparni:
        s+=1
        Su.append(x)
    elif x in Parni:
        f+=1
        Fu.append(x)
    elif x in Primarni:
        a+=1
        Au.append(x)
    else:
        i+=1
```

```
23          Iu.append(x)
24  print 'Neparnih=',Su
25  print 'Parnih=',Fu
26  print 'Primarnih brojeva=',Au
27  print 'Ostalih=', Iu
28  print 'Neparnih=',s,', Parnih=',f,', Primarnih brojeva=',a,',
    Ostalih=',i
```

što daje:

Zaslon 3.14

```
Neparnih= [1, 3, 5, 19]
Parnih= [2, 4, 18]
Primarnih brojeva= []
Ostalih= [2j, (1+5j), 2.4]
Neparnih= 4 , Parnih= 3 , Primarnih brojeva= 0 , Ostalih= 3
```

Nakon izvođenja programa može se uočiti da rješenje nije kako se očekivalo. Na primjer, u zadanoj listi *Brojevi* od deset brojeva primarnih je četiri (2,3,7,19), te se očekuje da oni budu ispisani na zaslonu. Naime, na zaslonu nije ispisan niti jedan primbroj. Zašto? Pa zato što if-elif petlja počinje s usporedbom prvo neparnih, pa onda parnih pa tek onda primarnih brojeva. Znači npr. za x jednak 3, prvo pita da li postoji broj tri u listi *Neparni*, a budući da postoji onda poveća brojač *s* za jedan i dopuni listu *Su* s elementom 3. Nakon toga izlazi iz if-elif petlje. Budući da je 3 i neparan broj i primaran, sada je pohranjen samo u listi neparnih brojeva *Su* i neće biti pohranjen i u listi *Au*.

Postoji još jedna zanimljivost koju treba predvidjeti. Ako umjesto inicijalizacije listi *Su*, *Fu*, *Au* i *Iu*, koje se nalaze u redcima 5-8 ovog primjera, zamjenimo s jednim retkom: $Su = Fu = Au = Iu = []$, na zaslonu će se prikazati:

Zaslon 3.15

```
Neparnih= [1, 2, 3, 4, 5, 18, 19, 2j, (1+5j), 2.4]
Parnih= [1, 2, 3, 4, 5, 18, 19, 2j, (1+5j), 2.4]
Primarnih brojeva= [1, 2, 3, 4, 5, 18, 19, 2j, (1+5j), 2.4]
Ostalih= [1, 2, 3, 4, 5, 18, 19, 2j, (1+5j), 2.4]
Neparnih= 4 , Parnih= 3 , Primarnih brojeva= 0 , Ostalih= 3
```

Može se odmah primjetiti da su brojači elemenata ostali isti što znači da se prolaz programa kroz petlje odvijao na jednak način kao prije, ali je zato broj elemenata u listama promijenjen. Možete li uočiti razlog? Doduše, s navedenom promjenom inicijaliziranih listi sve varijable Su, Fu, Au, Iu sadrže istu memorijsku adresu tj. pokazuju na istu listu u memoriji. Stoga kad se lista dopunjava (a lista je promjenjivi objekt) ona ostaje u memoriji na istoj adresi, pa time i svi pokazivači na tu adresu pokazuju i dalje na tu listu. To naravno ne vrijedi za nepromjenjive objekte, kao npr. cijele brojeve kod brojača $s = f = a = i = 0$.

Ako se zamijene naredbe *elif* s *if*, te *else* s *if x not in Neparni+Parni+Primarni*, dobit će se željeni rezultat:

Zaslon 3.16

```
Neparnih= [1, 3, 5, 19]
Parnih= [2, 4, 18]
```

```
Primarnih brojeva= [2, 3, 5, 19]
Ostalih= [2j, (1+5j), 2.4]
Neparnih= 4 , Parnih= 3 , Primarnih brojeva= 4 , Ostalih= 3
```

3.3.2 Naredba `break`

Upotreba naredbe `break` dopušta se jedino unutar tijela petlje. Kada se `break` izvrši, petlja završava. Ako je petlja smještena unutar drugih petlja, `break` završava jedino petlju u kojoj se break nalazi, te programsko upravljanje nastavlja izvan nje. `Break` naredba je obično unutar nekog odlomka u naredbi unutar tijela petlje, tako da se izvršava uvjetno. Jedno uobičajeno korištenje naredbe `break` je u implementaciji petlje koja odlučuje hoće li se izvršavati samo u sredini svake iteracije:

Program 3.5 Ispisuju broj slova unesenog stringa. Izlazi s tipkom 'Enter'.

```
1 while True:
2     x=raw_input(u'Brojim znakove (završetak samo Enter): ')
3     if x=='': break
4     print x, 'ima', len(x),'znakova'
```

što daje (uz interakciju s korisnikom koji upisuje niz po niz znakova ili završava s '*Enter*' tipkom):

Zaslon 3.17

```
Institut za jezik i jezikoslovlje ima 33 znakova
Zagreb ima 6 znakova
Hrvatska ima 8 znakova
>>>
```

3.3.3 Naredba `continue`

Naredbe `continue` dopušta se jedino unutar tijela petlje i po svom bezuvjetnom skoku slična je `break` naredbi, ali umjesto da iskače iz petlje, *continue* vraća tijek programa na početnu naredbu petlje, točnije na novu, iduću iteraciju.

Zaslon 3.18

```
>>> for i in ['Uvijek','neće','izgubiti','tko','se','bude','potruditi']:
...     if i[-2:]!='ti': continue
...     print i
...
izgubiti
potruditi
>>>
```

U ovom primjeru '*continue*' vraća programsko upravljanje na početnu naredbu petlje, ako izabrani element ne završava na '*ti*'. Ako završava, onda ga '`print i`' naredba ispisuje.

3.4 Ispis

3.4.1 Naredba *Print*

Naredba za ispis koristi ključnu riječ `print` nakon koje slijedi niti jedan ili više izraza odvojenih zarezima. `Print` je praktičan i jednostavan način ispisivanja vrijednosti u tekstovnom obliku.

`Print` ispisuje na zaslon računala svaki izraz `x` kao niz znakova što je isto kao kad bi se prethodno pozvala funkcija `str(x)` koja bi ekslicitno tip podatka, npr. broja pretvorila u string. `Print` implicitno izbacuje razmak između izraza, i također implicitno uključuje novi red ($\backslash n$) nakon posljednjeg izraza, osim u slučaju kad iza posljednjeg izraza slijedi zarez. Evo nekih primjera naredbe `print`:

Program 3.6 Ispis znakova i brojeva; literala i varijabli.

```
1 slovo = 'A'
2 print "Brojke:", 5, 3.14, 'i slova: ', slovo,'BC', 'DEF'
3 cbr = 100
4 print  "Ispis varijabli:",  cbr, slovo
```

što daje:

Zaslon 3.19
```
Brojke: 5 3.14 i slova:  A BC DEF
Ispis varijabli: 100 A
>>>
```

Odredište izlaza naredbe `print` obično je pridružen zaslonu računala, ali se isto tako može pridružiti drugom objektu (npr. datoteci) ili izlaznoj napravi. Format ispisa može se preciznije kontrolirati uporabom operatora % ili drugim tehnikama obradbe nizova znakova. U Python ver. 3.0 nadalje, *print* naredba postaje funkcija *print()* s pripadnim argumentima za ispis.

3.4.2 Formatiranje ispisa

Priprava ispisa (formatiranje) je postupak uređivanja stringa prije nego se on pokaže na zaslonu ili spremi u datoteku. Postoje dva načina priprave ispisa, jedan koristi formatirajući string operator (%), a drugi, noviji pristup, koristi metodu (funkciju) *format()*. U većini slučaja sintaksa je ista: umjesto % koristi se '{}' s dodatkom ':'. Na primjer, '

Zaslon 3.20
```
>>> u'%s  došao,  %s  pošao. ' % ('kako', 'tako')
u'kako  došao,  tako  pošao. '
>>> u'{0} došao, {1} pošao.'.format('kako', 'tako')
u'kako došao, tako pošao.'
>>> u'{1} došao, {0} pošao.'.format('kako', 'tako')
u'tako došao, kako pošao.'
>>> 'Ovo je {0} bilo je {1}'.format('sef',[2,4])
'Ovo je sef bilo je [2, 4]'
```

Kako se iz primjera vidi stari operator (%) zamijenjen je metodom u kojoj se pozicijski razmještaj može iskoristiti na više (logičnih) načina, pa se preporučuje novija metoda formatiranja zapisa.

Budući da je puno ispisa načinjeno na stari način, u programu 3.7 pokazano je formatiranje više različitih tipova podataka, uz opis svakog od njih:

Program 3.7 Primjena PRINT naredbe s formatiranjem.

```
1 #formatiranje ispisa
2 x='Dobar dan'
```