Podloge za stručno usavršavanje učitelja osnovnih škola
za domenu
Računalno razmišljanje i programiranje

09_B
Decimalni razlomici, modul `decimal`
strojni oblik racionalnih brojeva, tip `float`

Uz dozvolu izdavača korišteni su sadržaji iz priručnika:

Leo Budin, Predrag Brođanac, Zlatka Markučič
Smiljana Perić, Dejan Škvorc, Magdalena Babić

Računalno razmišljanje i programiranje u Pythonu
Element, Zagreb, 2017

Decimalni razlomci

Računanje s razlomcima ponekad je nespretno i teško posebice kada nazivnici postaju veliki brojevi. Najveća poteškoća je određivanje najmanjeg zajedničkog nazivnika. Zbog toga se krajem XVI. stoljeća u Europi proširila uporaba decimalnih razlomaka.

U decimalnim se razlomcima jedinica dijeli na deset jednakih dijelova – na desetinke. Desetinka se dalje dijeli na deset jednakih dijelova – na stotinke. Daljnjim podjelama dobivamo tisućinke, desettisućinke, stotisućinke itd.

Zapisivanje decimalnih razlomaka mnogo je jednostavnije od zapisivanja običnih razlomaka jer nazivnike ne moramo pisati – oni su određeni položajem u zapisu. Tako ćemo umjesto:

$$3/10 + 1/100 + 0/1000 + 7/10000$$

jednostavno napisati

$$0.3107$$

Iako je normama i pravopisom standardnog hrvatskoga jezika za pisanje decimalnih brojeva predviđen zarez, u programiranju koristimo točku za označavanje početka zapisivanja decimalnih razlomaka.

Prva znamenka iza točke (3) je znamenka desetinke. Iza nje slijedi znamenka stotinke (1) i redom znamenka tisućinke (0) i desettisućinke (7). Broj možemo zapisati kao decimalni broj ili u obliku razlomka:

$$0.3107 = 3107/10000.$$

Kada zapisujemo mješovite brojeve, najprije pišemo cijeli dio broja, iza njega stavljamo točku i nakon nje zapisujemo decimalni razlomak.

Prema tome, mješovite brojeve zapisivat ćemo ovako:

$$3.5 = 3 + 5/10 = 3 + 1/2$$

$$4.125 = 4 + 125/1000 = 4 + 1/8$$

$$17.17 = 17 + 17/100$$

$$0.23 = 0 + 23/100 = 23/100$$

$$0.001 = 0 + 1/1000 = 1/1000$$

Podsjetimo se iz matematike na neka svojstva ovakvog zapisa brojeva:

- dodavanje odnosno brisanje nula na kraju decimalnog razlomka neće promijeniti njegovu vrijednost (2.3 = 2.300)
- decimalni razlomak bit će uvećan za faktor 10 ako decimalnu točku pomaknemo za jedno mjesto udesno ($0.017 \cdot 10 = 0.17$), odnosno smanjen za faktor 10 ako točku pomaknemo za jedno mjesto ulijevo
- zbrajanje i oduzimanje decimalnih razlomaka (decimalnih brojeva) obavljamo na jednak način kao i kod cijelih brojeva (pritom se mora paziti da brojeve napišemo jedan ispod drugog tako da decimalna točka bude na istom mjestu)
- množenje se obavlja tako da zanemarimo postojanje decimalne točke i brojeve pomnožimo kao cijele brojeve i zatim postavimo decimalnu točku tako da broj decimalnih mjesta u umnošku bude jednak zbroju decimalnih mjesta faktora.

Sva ta pravila računanja s decimalnim brojevima ugrađena su u modul `decimal` predviđen za računanje s decimalnim brojevima.

Modul `decimal`

Za prikaz i računanje s decimalnim brojevima u *Pythonu* postoji modul `decimal`. Taj modul omogućuje uvođenje novog tipa koji se naziva `Decimal`.

Decimalne brojeve možemo definirati na dva načina:

- od broja tipa `int` ili
- od decimalnog broja zapisanog kao string.

Pogledajmo najprije način zadavanja pomoću broja tipa `int`:

```
>>> from decimal import *                    #1
>>> a = Decimal(1)                           #2
>>> a                                        #3
Decimal('1')
>>> print(a)                                 #4
1
>>> b = Decimal(7)                           #5
>>> b                                        #6
Decimal('7')
>>> print(b)                                 #7
7
```

Naredbom (#1) importirali smo modul `decimal`. Nakon toga možemo definirati decimalne brojeve kao `Decimal(cijeli_broj)`. Naredbe (#2) i (#5) definiraju decimalne brojeve iz brojeva `1` i `7`. Naredbama (#3) i (#6) vidimo kako su oni pohranjeni u spremniku, a kako ih funkcija `print()` ispisuje pokazuju nam naredbe (#4) i (#7).

Nakon definiranja brojeva možemo pogledati kako na njih djeluju aritmetičke operacije:

```
>>> a + b
Decimal('8')
>>> a - b
Decimal('-6')
>>> a * b
Decimal('7')
>>> a // b
Decimal('0')
>>> a % b
Decimal('1')
>>> a / b
Decimal('0.14285714285714285714285714429')
```

Uz aritmetičke operacije koje smo razmatrali do sada i koje djeluju na jednak način kao i kod cijelih brojeva, operacija dijeljenja `/` koju smo upoznali kod razlomaka, kao rezultat će nam vratiti decimalni broj.

Nije teško izbrojiti da smo kao rezultat dobili decimalni broj s dvadeset i osam znamenaka.

Naime, modul `decimal` omogućuje prikaz rezultata operacija u obliku decimalnog broja do najviše dvadeset i osam decimalnih mjesta. U modulu `fractions` vidjeli smo da bi taj isti racionalni broj napisali jednostavno kao 1/7.

Rezultat aritmetičke operacije biti će decimalni broj i onda kada je samo jedan od operanada decimalni broj, a drugi operand može biti tipa `int` pa možemo napisati sljedeći niz naredbi:

```
>>> jedan = Decimal(1)                                          #8
>>> for i in range(1, 21):                                      #9
        print('1/{:<2} = {}'.format(i, jedan / i))

1/1  = 1
1/2  = 0.5
1/3  = 0.33333333333333333333333333333
1/4  = 0.25
1/5  = 0.2
1/6  = 0.16666666666666666666666666667
1/7  = 0.14285714285714285714285714429
1/8  = 0.125
1/9  = 0.11111111111111111111111111111
1/10 = 0.1
1/11 = 0.090909090909090909090909090091
1/12 = 0.083333333333333333333333333333
1/13 = 0.076923076923076923076923076923
1/14 = 0.071428571428571428571428571429
1/15 = 0.066666666666666666666666666667
1/16 = 0.0625
1/17 = 0.058823529411764705882352941176
1/18 = 0.055555555555555555555555555556
1/19 = 0.052631578947368421052631578947
1/20 = 0.05
```

Naredbom (#8) pripisali smo varijabli `jedan` decimalni broj `1` tako da će rezultat dijeljenja `jedan/i` u naredbi (#9) biti decimalni broj. Zanimljivo je u ispisu ustanoviti kako je od tih dvadeset razlomaka samo njih osam prikazano decimalnim brojem s konačnim brojem decimalnih mjesta. Ostali razlomci imaju punih dvadeset i osam decimalnih mjesta koliko je najviše moguće u modulu `decimal`. Oni zapravo imaju beskonačno mnogo decimalnih mjesta samo ih modul `decimal` ne može sve prikazati.

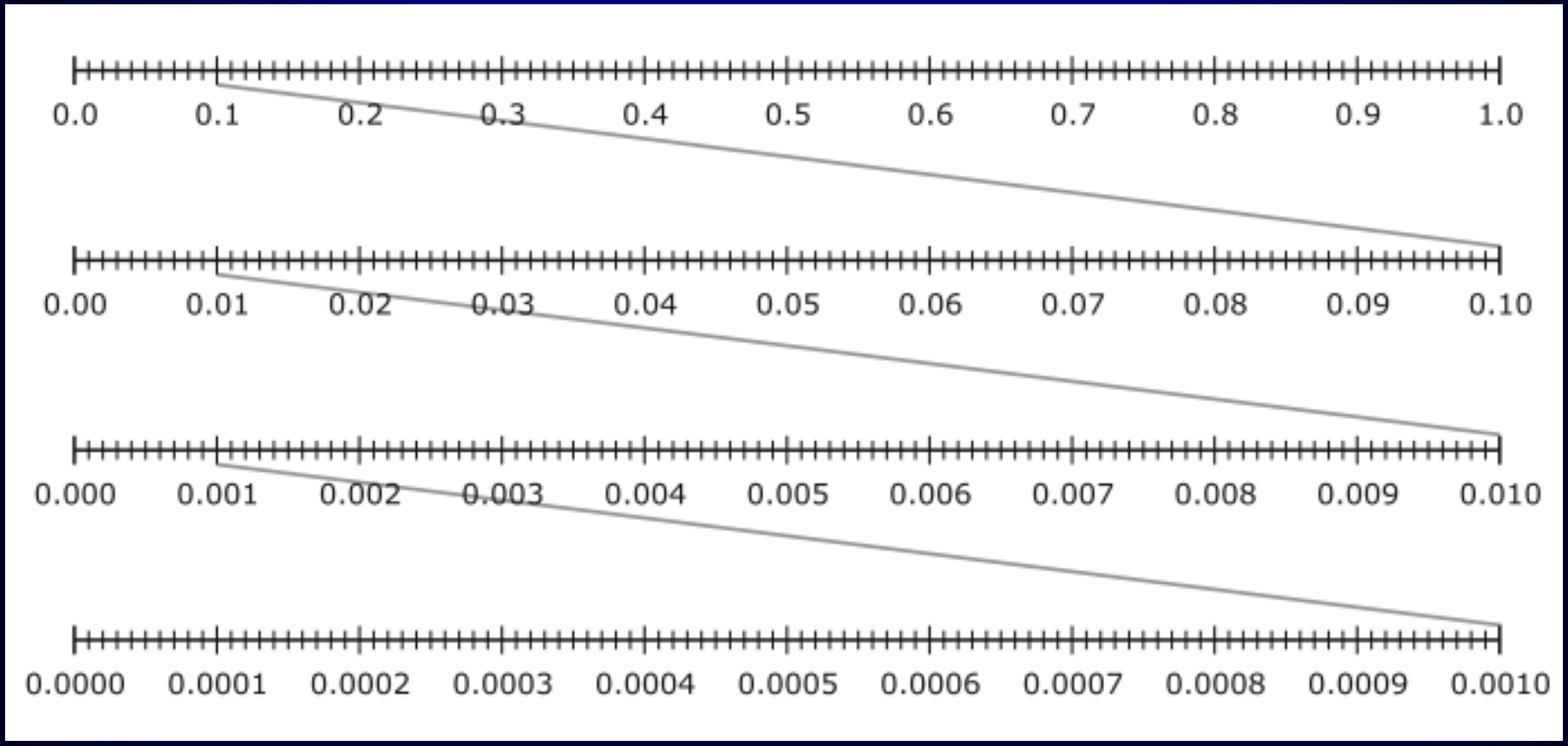
Nadalje, još možemo primijetiti da se zadnja znamenka može razlikovati od prethodnih znamenaka koje se ponavljaju u grupama. Tako se pri decimalnom zapisu razlomka 1/6 ponavljaju znamenke 6, a zadnja u zapisu je 7, dok je kod zapisa 1/12 zadnja znamenka 3 jednaka onima koje su ponavljaju. Vidjet ćemo kasnije da je to zbog pravila zaokruživanja zadnje znamenke.

Decimalne brojeve možemo na drugi način zadati tako da ih napišemo u obliku stringa. Pogledajmo:

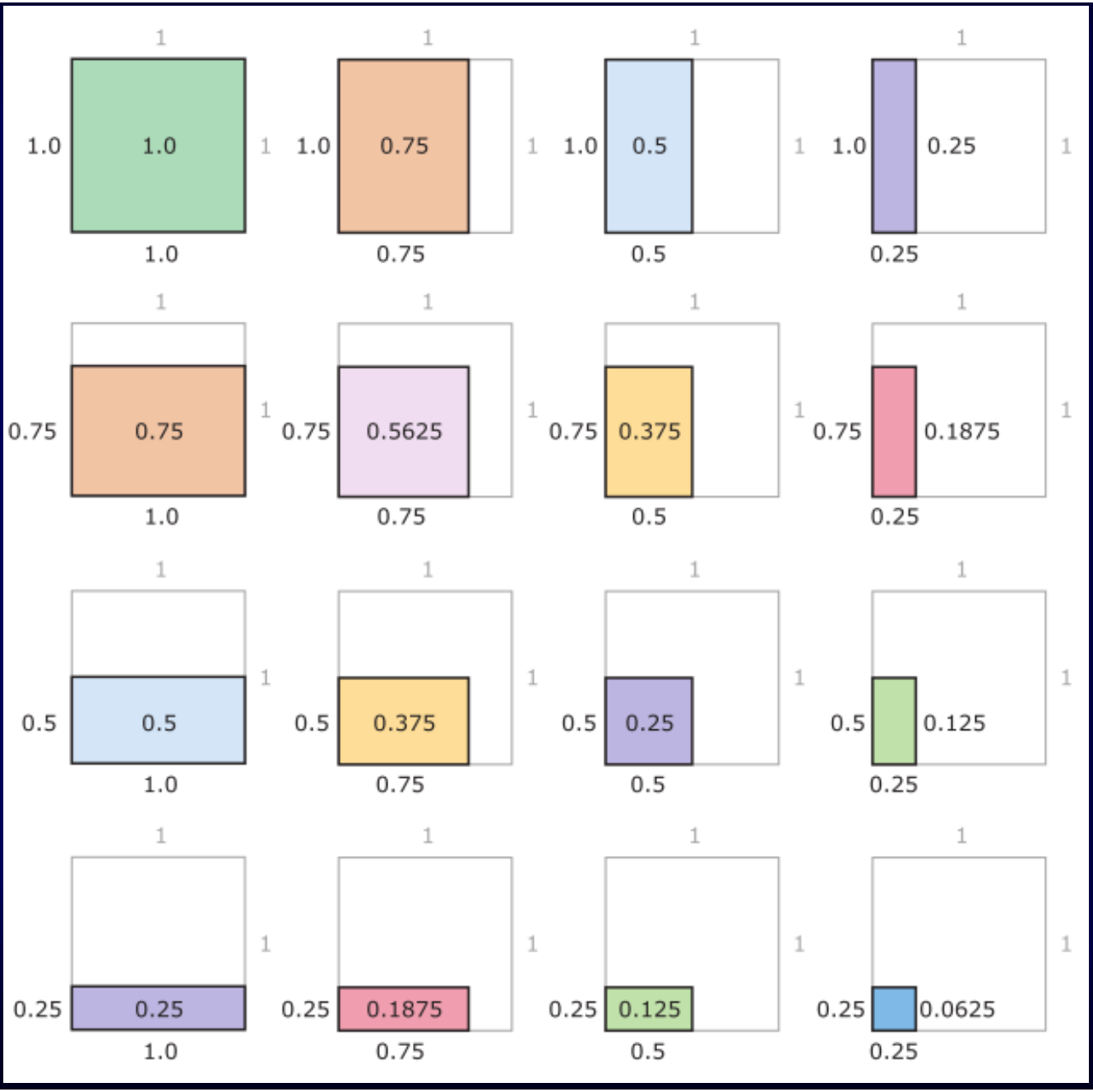
```
>>> from decimal import Decimal
>>> a = Decimal('1.25')                                    #10
>>> a                                                      #11
Decimal('1.25')
>>> print(a)
1.25
>>> b = input('Upiši decimalni broj: ')                    #12
Upiši decimalni broj: 2.75
>>> b
'2.75'
>>> c = Decimal(b)                                         #13
>>> c
Decimal('2.75')
>>> d = Decimal(input('Upisati decimalni broj: '))         #14
Upisati decimalni broj: 3.78
>>> d
Decimal('3.78')
```

Sada možemo naredbom (#10) definirati decimalni broj, a naredbom (#11) vidjeti kako je on pohranjen u varijabli `a` te kako se ispisuje naredbom `print()`. Funkcijom `input()` u naredbi (#12) unijeli smo u varijablu `b` string koji treba dodatnom naredbom (#13) prikazati kao decimalni broj. Već smo kod drugih tipova ustanovili da pretvorbu možemo obaviti istom naredbom u kojoj se obavlja upis tako da naredba oblika (#14) rješava problem.

Decimalni razlomci smješteni su na brojevnom pravcu kao i svi ostali racionalni brojevi. Na slici 9.6. prikazan je položaj decimalnih brojeva i to onih na razmaku jedne desetinke u intervalu [0.0, 1.0] i zatim s promijenjenim mjerilima onih na razmaku jedne stotinke u intervalu [0.0, 0.1], te na razmaku tisućinke u intervalu [0.0, 0.01] i desettisućinke u intervalu [0.0, 0.001].



Zanimljivo je pogledati i sliku 9.7 na kojoj su prikazani isti umnošci kao i na slici 9.5. s tim da su razlomci ovdje zamijenjeni odgovarajućim decimalnim zapisom.



sl. 9.7.

Približni prikaz decimalnog broja smanjivanjem broja decimalnih mjesta, funkcija `round()`

U mnogim primjenama nije potrebno računati s brojevima koji imaju mnogo decimalnih mjesta. Brojevi se zbog toga prikazuju s manjim brojem decimalnih mjesta, a ostala se mjesta odbacuju. Pri tome se nastoji načiniti što je moguće manju pogrešku.

Uobičajena pravila kojih se treba držati pri odbacivanju znamenaka:

1) ako je prva odbačena znamenka manja od 5, zadnja zadržana znamenka ostaje nepromijenjena
2) ako je prva odbačena znamenka veća od 5, tada se zadnja zadržana znamenka povećava za jedan
3) ako je prva odbačena znamenka jednaka 5 i iza nje u odbačenim dijelu postoje još neke znamenke, tada se zadnja zadržana znamenka povećava za jedan
4) ako je odbačena znamenka jednaka 5 i iza nje nema znamenaka (sve su odbačene znamenke jednake nuli) tada se zadnja zadržana znamenka ne mijenja ako je ona parna odnosno povećava za jedan ako je neparna.

Postupak odbacivanja znamenaka po tim pravilima nazivamo **zaokruživanjem broja** na zadani broj znamenaka.

U *Pythonu* postoji funkcija koja po tim pravilima obavlja zaokruživanje decimalnih razlomaka na zadani broj mjesta. Ta se funkcija poziva na sljedeći način `round(decimalni_broj, broj_mjesta)`. Pogledajmo kako ona djeluje:

```
>>> round(Decimal('1.342'), 2)                                    #1
Decimal('1.34')
>>> round(Decimal('1.346'), 2)                                    #2
Decimal('1.35')
>>> round(Decimal('1.34501'), 2)                                  #3
Decimal('1.35')
>>> round(Decimal('1.345'), 2)                                    #4
Decimal('1.34')
>>> round(Decimal('1.335'), 2)                                    #5
Decimal('1.34')
```

Naredba (#1) u skladu s prvim pravilom ostavlja znamenku 4 nepromijenjenom, dok će se u naredbi (#2) ona povećati na 5 jer je prva odbačena znamenka veća od 5. U naredbi (#3) je prva odbačena znamenka 5, ali se iza nje u odbačenom dijelu nalaze još neke znamenke te će se zadnja znamenka u skladu s pravilom **3)** povećati na 5. U naredbi (#4) odbacuje se znamenka 5 iza koje više nema znamenaka. S obzirom na to da je zadnja prihvaćena znamenka parna (4), ona se neće mijenjati. U naredbi (#5) smo promijenili decimalni broj i zaokružujemo broj `Decimal('1.335')`. S obzirom na to da je zadnja prihvaćena znamenka neparna, ona će se povećati tako da ćemo opet dobit zaokruženi broj `Decimal('1.34')`. Drugim riječima brojevi `Decimal('1.345')` i `Decimal('1.335')` zaokružuju se na isti broj `Decimal('1.34')`.

Primjer:

Napišimo program u kojem ćemo u funkciji `main()` zadati listu od osam decimalnih brojeva. Program treba ispisati originalnu listu, listu čiju su elementi zaokruženi na dvije decimale, listu s elementima zaokruženima na jednu decimalu te zbrojeve elemenata tih lista.

Napisat ćemo funkciju `zaokružiti()` koja zaokružuje sve elemente liste na n *decimalnih mjesta. U funkciji `ispis()` ispisat ćemo sve elemente liste u petlji te zbroj elemenata liste.*

```
#Zaokruživanje decimalnih brojeva - program_9_12.py

from decimal import Decimal

def zaokružiti(br, n):
    return [round(x, n) for x in br]

def ispis(br):
    for x in br:
        print('{:<5}'.format(x), end=' ')
    print('   Zbroj brojeva je {}'.format(sum(br)))

def main():
    br = [Decimal('4.526'), Decimal('0.457'), Decimal('3.195'),
          Decimal('1.350'), Decimal('0.650'), Decimal('4.751'),
          Decimal('0.950'), Decimal('0.050')]                    #6
    ispis(br)                                                    #7
    ispis(zaokružiti(br, 2))                                     #8
    ispis(zaokružiti(br, 1))                                     #9

main()
```

U funkciji `main()` smo listom (#6) zadali osam decimalnih brojeva. Funkcija `zaokružiti()` vraća listu u kojoj zaokružuje sve elemente liste na `n` decimalnih mjesta. Druga funkcija `ispis()` ispisat će sve elemente liste u petlji `for` i ispisati zbroj elemenata liste. Funkcija `sum(lista)` zbraja sve vrijednosti elemenata liste.

Naredbama (#7), (#8) i (#9) ispisat ćemo redom zadanu listu `br` te liste u kojima su elementi zaokruženi na dva, odnosno jedno decimalno mjesto. Ispis izgleda ovako:

```
4.526 0.457 3.195 1.350 0.650 4.751 0.950 0.050    Zbroj brojeva je 15.929
4.53  0.46  3.20  1.35  0.65  4.75  0.95  0.05     Zbroj brojeva je 15.94
4.5   0.5   3.2   1.4   0.6   4.8   1.0   0.0      Zbroj brojeva je 16.0
```

Iz ispisa programa `program_9_12.py` možemo ustanoviti da su približne zaokružene vrijednosti djelovale i na točnost zbroja. Kada bismo točan zbroj `15.929` zaokružili na dva decimalna mjesta, dobili bismo rezultat `15.93` i zaokruživanjem na jedno, odnosno `16.0`. Vidimo da smo zaokruživanjem pribrojnika dobili pogrešno izračunane zbrojeve. Može se zaključiti da je svakako preporučljivo sva potrebna izračunavanja obaviti s točnim nezaokruženim brojevima i samo u ispisima pokazati onaj broj decimalnih mjesta koji želimo vidjeti.

Zaokruživanje prilikom ispisa decimalnih brojeva, formatiranje slovom `f`

U odjeljku 5.7. naučili smo kako se metodom `format` oblikuju ispisi koji sadrže brojeve tipa `int`. Pogledajmo tamo sve mogućnosti formatiranja. One vrijede i za decimalne brojeve. Jedina je razlika da se umjesto slova `d` ovdje rabi slovo `f` te da se nakon ukupnog broja znakova mora odvojeno točkom napisati i broj željenih decimalnih mjesta. Najbolje se to vidi u sljedećim primjerima:

```
>>> a = Decimal('15.23845')                                    #1
>>> print(a)
15.23845
>>> print('{:10.5f}|'.format(a))                               #2
  15.23845|
>>> print('{:<10.4f}|'.format(a))                              #3
15.2384   |
>>> print('{:<10.3f}|'.format(a))                              #4
15.238    |
>>> print('{:<10.2f}|'.format(a))                              #5
15.24     |
>>> print('{:<10.1f}|'.format(a))                              #6
15.2      |
>>> print('{:>10.1f}|'.format(a))                              #7
      15.2|
>>> print('{:^10.1f}|'.format(a))                              #9
   15.2   |
```

Naredbom (#1) pridružili smo varijabli `a` decimalni broj. Obična naredba `print()` ispisat će ga na početku reda. U naredbi (#2) oblikovan je izlazni string koji će ispisati decimalni broj. Oznaka `10.5f` određuje da je za broj predviđeno deset mjesta od kojih je pet za znamenke iza decimalne točke. Na kraju broja dodali smo vertikalnu crticu (utipkava se tipkama *(AltGr)+(w)*) koja nam označava kraj predviđenog polja za ispis broja. Naredba (#3) ispisuje broj `a` počevši od lijeve granice polja s četiriju znamenaka iza decimalne točke. Nakon toga slijede naredbe (#4) do (#6) koje ispisuju broj redom s tri, dva i jednim decimalnim mjestom. Konačno, naredba (#7) ispisat će broj smješten s desne strane polja od deset znakova i naredba (#8) u sredini polja.

Ovim načinom formatiranja ispisat će se i nule dodane na kraju decimalnog broja koje se uobičajeno ne ispisuju. Pogledajmo:

```
>>> b = Decimal('2.34')
>>> print('{:10.5f}|'.format(b))
   2.34000|
>>> print('{:<10.5f}|'.format(b))
2.34000   |
>>> print('{:^10.5f}|'.format(b))
 2.34000  |
```

Vidimo da je predviđen format za ispis `10.5f`, dakle u polju širine deset znakova s pet decimalnih mjesta iza decimalne točke. Stoga je pri ispisu broj `2.34` nadopunjen trima znamenkama 0.

Zaključno možemo ustanoviti da pri računanju s decimalnim brojevima treba sve aritmetičke operacije obavljati bez zaokruživanja međurezultata.

Međutim, ispis velikog broja decimalnih mjesta može biti besmislen. Zbog toga ćemo zaokruživanje obavljati prilikom ispisa i to stringom za oblikovanje koji ima završno slovo `f`. Važno je naglasiti da se to zaokruživanje također obavlja po svim pravilima zaokruživanja.

Zapisivanje vrlo velikih i vrlo malih brojeva

Zbrajanje međusobno jednakih pribrojnika

Želimo li izračunati zbroj od više međusobno jednakih pribrojnika, tada umjesto višekratnog zbrajanja primjenjujemo operaciju množenja tog pribrojnika s brojem njegovog ponavljanja u zbroju.

Tako ćemo pisati:

$$2 + 2 + 2 = 3 \cdot 2 = 6$$
$$10 + 10 + 10 + 10 = 4 \cdot 10 = 40.$$

Množenje međusobno jednakih faktora

Iz matematike znamo da se sličnim načinom razmišljanja množenje međusobno jednakih faktora može kraće zapisati operacijom **potenciranja** ovako:

$$2 \cdot 2 \cdot 2 = 2^3 = 8$$
$$10 \cdot 10 \cdot 10 \cdot 10 = 10^4 = 10\,000.$$

Brojeve 2 i 10 nazivamo **bazama** potencije, a brojeve 3 i 4 **eksponentima** potencije.

Ako za bazu uvedemo ime varijable x, a za eksponent varijablu n, potenciranje možemo napisati u obliku x^n. Posebno treba naglasiti da vrijedi sljedeće:

- potenciranje s eksponentom $n = 1$ daje vrijednost baze, tj. $x^1 = x$
- potenciranje sa $n = 0$ daje vrijednost 1, tj. $x^0 = 1$ (pri čemu mora biti $x \neq 0$)
- potenciranje s negativnim eksponentom označava razlomak $x^{-n} = 1/x^n$.

U *Pythonu* se operator potenciranja piše dvjema uzastopnim zvjezdicama `**`. Pogledajmo neke primjere potenciranja:

```
>>> from fractions import Fraction
>>> x = Fraction(2)                                    #2
>>> x ** 1
Fraction(2, 1)
>>> x ** 0
Fraction(1, 1)
>>> x ** 3                                             #3
Fraction(8, 1)
>>> x ** -3
Fraction(1, 8)
```

Nakon importiranja modula `fractions` odabrali smo naredbom (#2) za vrijednost baze x racionalni broj 2. Vidimo da potenciranje eksponentom 1 daje vrijednosti baze, a potenciranje eksponentom 0 daje rezultat 1. Naredbom (#3) izračunali smo vrijednost potencije s eksponentom 3, dok je potenciranje s jednakom negativnom vrijednošću eksponenta dalo recipročnu vrijednost.

Načinit ćemo isto s bazom 10, ali ćemo rezultate potenciranja ispisati funkcijom `print()`:

```
>>> x = Fraction(10)
>>> print(x ** 1)
10
>>> print(x ** 0)
1
>>> print(x ** 3)
1000
>>> print(x ** -3)
1/1000
```

Primjer:

Napišimo program koji će ispisati tablicu potencija brojeva 2 i 10 s eksponentima od 1 do 10.

```
#Program_9_13_b.py

from fractions import Fraction

b2 = Fraction(2)                                                    #7
b10 = Fraction(10)                                                  #8
for i in range(1, 11):
    print('{} **{:2} = {:4}'.format(b2, i, str(b2 ** i)), end=' ')  #9
    print('{} **{:2} = {:10}'.format(b10, i, str(b10 ** i)))        #10
```

Ispis:

```
2 ** 1 = 2        10 ** 1 = 10
2 ** 2 = 4        10 ** 2 = 100
2 ** 3 = 8        10 ** 3 = 1000
2 ** 4 = 16       10 ** 4 = 10000
2 ** 5 = 32       10 ** 5 = 100000
2 ** 6 = 64       10 ** 6 = 1000000
2 ** 7 = 128      10 ** 7 = 10000000
2 ** 8 = 256      10 ** 8 = 100000000
2 ** 9 = 512      10 ** 9 = 1000000000
2 **10 = 1024     10 **10 = 10000000000
```

Primjer:

Napišimo program koji će ispisati tablicu potencija brojeva 2 i 10 s negativnim eksponentima od –1 do –10.

U odnosu na prethodni program, napravit ćemo minimalne izmjene. Umjesto pozitivnog eksponenta (`i`), napisat ćemo negativni (`-i`) i prilagoditi formate ispisa.

```
#Program_9_13_c.py
from fractions import Fraction

b2 = Fraction(2)
b10 = Fraction(10)
for i in range(1, 11):
    print('{} **{:3} = {:6}'.format(b2, i, str(b2 ** -i)), end=' ')
    print('{} **{:3} = {:12}'.format(b10, i, str(b10 ** -i)))
```

Ispis:

```
2 ** -1 = 1/2       10 ** -1 = 1/10
2 ** -2 = 1/4       10 ** -2 = 1/100
2 ** -3 = 1/8       10 ** -3 = 1/1000
2 ** -4 = 1/16      10 ** -4 = 1/10000
2 ** -5 = 1/32      10 ** -5 = 1/100000
2 ** -6 = 1/64      10 ** -6 = 1/1000000
2 ** -7 = 1/128     10 ** -7 = 1/10000000
2 ** -8 = 1/256     10 ** -8 = 1/100000000
2 ** -9 = 1/512     10 ** -9 = 1/1000000000
2 **-10 = 1/1024    10 **-10 = 1/10000000000
```

Ako baze definiramo kao decimalne brojeve ispis za pozitivne eksponente neće se promijeniti, ali ćemo za negativne eksponente umjesto običnih razlomaka dobiti decimalne brojeve.

Primjer:

Napišimo program koji će ispisati tablicu potencija brojeva 2 i 10 s negativnim eksponentima od –1 do –10. Baze treba definirati kao decimalne brojeve.

```
from decimal import Decimal
b2 = Decimal(2)
b10 = Decimal(10)
for i in range(1,11):
    print('{} **{:3} = {:12f}'.format(b2, -i, b2 ** -i), end=' ')      #11
    print('{} **{:3} = {:12f}'.format(b10, -i, b10 ** -i))             #12
```

Ispis:

```
2 ** -1 =          0.5    10 ** -1 =          0.1
2 ** -2 =         0.25    10 ** -2 =         0.01
2 ** -3 =        0.125    10 ** -3 =        0.001
2 ** -4 =       0.0625    10 ** -4 =       0.0001
2 ** -5 =      0.03125    10 ** -5 =      0.00001
2 ** -6 =     0.015625    10 ** -6 =     0.000001
2 ** -7 =    0.0078125    10 ** -7 =    0.0000001
2 ** -8 =   0.00390625    10 ** -8 =   0.00000001
2 ** -9 =  0.001953125    10 ** -9 =  0.000000001
2 **-10 = 0.0009765625    10 **-10 = 0.0000000001
```

U naredbama (#11) i (#12) morali smo u stringu za oblikovanje teksta slovom `f` istaknuti da ispisujemo decimalne brojeve. Isto tako, vrijednost potencije ovdje ne moramo pretvoriti u string

Eksponencijalni način zapisivanja decimalnih brojeva, formatiranje slovom e

Zapisivanje vrlo velikih i vrlo malih brojeva je vrlo nepregledno i zbog toga se u mnogim primjenama rabi tzv. **eksponencijalni** ili **znanstveni način zapisivanja** takvih brojeva.

Svaki se broj može zapisati kao decimalni broj koji lijevo od decimalne točke ima samo jedno brojevno mjesto što znači da cijeli dio ima vrijednost između 1 i 9. Ostale znamenke broja zapisuju se na decimalnim mjestima iza decimalne točke. Takav broj treba pomnožiti odgovarajućom potencijom broja 10 i to – pozitivnim eksponentom ako je broj veći od 10 i negativnim eksponentom ako je broj manji od 1.

Tako možemo pisati:

$$126000.0 = 1.26 \cdot 10^{5}$$
$$89470000000.0 = 8.947 \cdot 10^{10}$$
$$0.000000126 = 1.26 \cdot 10^{-7}$$
$$0.00008947 = 8.947 \cdot 10^{-5}.$$

Znanstveni zapis dobivamo tako da decimalnu točku pomičemo ulijevo ili udesno tako dugo dok ispred nje ne bude samo jedna znamenka različita od nule. Pritom brojimo za koliko smo mjesta točku pomaknuli i taj broj zapišemo kao eksponent baze 10. Ako smo išli ulijevo eksponet će biti pozitivan, a ako smo išli udesno eksponent će biti negativan.

U *Pythonu* postoji mogućnost za takvo ispisivanja decimalnih brojeva. U ispisanom stringu taj se način ispisa označava završnim slovom `e`.

```
>>> from decimal import Decimal
>>> x = Decimal('126000.0')
>>> '{0:<15.3e}|{0:15.3e}|{0:^15.3e}|'.format(x)                    #1
'1.260e+5       |      1.260e+5|   1.260e+5    |'
>>> '{0:<15.2e}|{0:15.2e}|{0:^15.2e}|'.format(x)                    #2
'1.26e+5        |       1.26e+5|    1.26e+5    |'
>>> '{0:<15.1e}|{0:15.1e}|{0:^15.1e}|'.format(x)                    #3
'1.3e+5         |        1.3e+5|    1.3e+5     |'
>>> y = Decimal('0.00008947')
>>> '{0:<15.3e}|{0:15.3e}|{0:^15.3e}|'.format(y)                    #4
'8.947e-5       |      8.947e-5|   8.947e-5    |'
>>> '{0:<15.2e}|{0:15.2e}|{0:^15.2e}|'.format(y)                    #5
'8.95e-5        |       8.95e-5|    8.95e-5    |'
>>> '{0:<15.1e}|{0:15.1e}|{0:^15.1e}|'.format(y)                    #6
'8.9e-5         |        8.9e-5|    8.9e-5     |'
```

U naredbi (#1) broj ćemo najprije ispisati lijevo u polju širine petnaest znakova i s trima decimalnim mjestima iza decimalne točke. Nakon toga broj će se ispisati smješten desno u polju širine petnaest znakova i na kraju u sredinu zadanog polja. U ispisu se ne pojavljuje baza 10 već iza slova `e` samo eksponent baze. Iza širine polja dolazi točka i iza nje broj koji određuje koliko će se decimalnih mjesta ispisati.

U naredbama (#1) i (#4) ispisujemo tri decimalne znamenke, u naredbama (#2) i (#5) dvije decimalne znamenke te u naredbama (#3) i (#6) jednu decimalnu znamenku. Vidimo da se prilikom odbacivanja decimalnih znamenaka i ovdje zadnja prihvaćena znamenka pravilno zaokružuje.

Strojni oblik pohranjivanja racionalnih brojeva, tip podataka `float` *i funkcija* `float()`

U prvom smo poglavlju spomenuli da su u spremniku računala sve informacije, svi podatci pohranjeni kao nizovi jedinica i nula.

Do sada smo naučili da se s pomoću tih nizova i jedinica mogu prikazati različiti tipovi odnosno klase podataka. Upoznali smo

- tip podataka `int` kojim se prikazuju cijeli brojevi
- tip podataka `str` koji služi za pohranjivanje niza znakova
- tip `bool` kojim možemo utvrđivati istinitost ili neistinitost logičkih sudova i izraza
- klasu `Fraction` iz modula `fractions` za prikaz i računanje s razlomcima
- klasu `Decimal` iz modula `decimal` za prikaz i računanje s decimalnim brojevima.

Sada ćemo upoznati još jedan tip podataka koji se zove `float`. Naziv je nastao skraćivanjem engleskog naziva *floating point number* koji bi se mogao prevesti na hrvatski kao *broj s pomičnom (plivajućom) točkom*. U odjeljku 9.4.2. smo naučili da se znanstveni zapis decimalnog broja dobiva tako da decimalnu točku pomičemo ulijevo ili udesno tako dugo dok ispred nje ne stoji samo jedna znamenka. Pritom moramo podešavati vrijednost eksponenta potencije od 10. Prema tome, decimalna točka "pliva" tako dugo dok se ne postavi na pravo mjesto.

Funkcija `float()`

Međutim, u računalu se brojevi pohranjuju u binarnom zapisu. U tom zapisu postoje samo znamenke 0 i 1 i brojevi se prikazuju kao binarni zapisi. Umjesto decimalnih razlomaka u računalu se pohranjuju binarni razlomci. Vidjeli smo u odjeljku 9.4.1. da ti razlomci nemaju desetinke, stotinke, tisućinke... nego polovinke, četvrtinke, osminke, šesnaestinke...

Funkcija `float()` će se s jedne strane pobrinuti da pohrani u računalu broj u binarnom obliku i da nama u ispisu na zaslonu monitora broj ispiše o obliku decimalnog broja.

Uporabu funkcije `float()` koja će obaviti pretvorbu stringa možemo pogledati u sljedećim primjerima:

```
>>> a = float(input('Upiši broj: '))                    #1
Upiši broj: 7
>>> a
7.0
>>> b = float(input('Upiši broj: '))
Upiši broj: 0.003
>>> b
0.003
>>> c = float(input('Upiši broj: '))
Upiši broj: 0.00001
>>> c
1e-05
```

```
>>> d = float(input('Upiši broj: '))
Upiši broj: 36.7e-10
>>> d
3.67e-09
>>> e = float(input('Upiši broj: '))
Upiši broj: 345.86e18
>>> e
3.4586e+20
>>> g = float(input('Upiši broj: '))                                    #2
Upiši broj: 34.6f
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    g = float(input('Upisati broj: '))  #6
ValueError: could not convert string to float: '34.6f'
```

U naredbi (#1) pretvorili smo string '7' u broj 7.0. Iz primjera je vidljivo da brojeve možemo utipkavati kao stringove različitih oblika: kao cijele brojeve, kao obične brojeve s decimalnom točkom te kao brojeve u znanstvenom zapisu. Ako upisujemo string koji se ne može prevesti u broj s pomičnom točkom, *Python* će dojaviti pogrešan upis kao što se to vidi kod upisa zatraženog naredbom (#2).

U interaktivnom sučelju ili u programu varijablama možemo pridruživati brojeve neposredno bez pretvorbe. Pogledajmo:

```
>>> a = 0.1
>>> b = 1.25
>>> c = 153.45
>>> a, b, c
(0.1, 1.25, 153.45)
>>> d, e, f = 2e17, 0.34e-10, 145.78e60
>>> d, e, f
(2e+17, 3.4e-11, 1.4578e+62)
```

Broj s pomičnom točkom možemo dobiti i kao rezultat aritmetičke operacije dijeljenja dvaju cijelih brojeva uporabom operatora `/`. Operator `/` smo već susreli kod racionalnih brojeva u modulu `fractions` i decimalnih brojeva u modulu `decimal`.

Dakle, u osnovnom dijelu *Pythona* (bez importiranja nekih modula) postoje dva operatora dijeljenja:

- operator cjelobrojnog dijeljenja `//` kojim dobivamo količnik u obliku cijelog broja
- operator dijeljenja `/` kojim dobivamo količnik u obliku broja s pomičnom točkom.

```
>>> 6 // 2
3
>>> 6 / 2
3.0
>>> 7 // 2
3
>>> 7 / 2
3.5
```

U interaktivnom sučelju ili u programu varijablama možemo pridruživati brojeve neposredno bez pretvorbe. Pogledajmo:

```
>>> a = 0.1
>>> b = 1.25
>>> c = 153.45
>>> a, b, c
(0.1, 1.25, 153.45)
>>> d, e, f = 2e17, 0.34e-10, 145.78e60
>>> d, e, f
(2e+17, 3.4e-11, 1.4578e+62)
```

Broj s pomičnom točkom možemo dobiti i kao rezultat aritmetičke operacije dijeljenja dvaju cijelih brojeva uporabom operatora `/`. Operator `/` smo već susreli kod racionalnih brojeva u modulu `fractions` i decimalnih brojeva u modulu `decimal`.

Dakle, u osnovnom dijelu *Pythona* (bez importiranja nekih modula) postoje dva operatora dijeljenja:

- operator cjelobrojnog dijeljenja `//` kojim dobivamo količnik u obliku cijelog broja
- operator dijeljenja `/` kojim dobivamo količnik u obliku broja s pomičnom točkom.

```
>>> 6 // 2
3
>>> 6 / 2
3.0
>>> 7 // 2
3
>>> 7 / 2
3.5
```

Želimo li točno računati s decimalnim razlomcima, onako kako smo to naučili u matematici na raspolaganju nam je modul `decimal`. Razliku između točnog računanja s decimalnim razlomcima u tom modulu i računanja s brojevima tipa `float` pokazat ćemo sljedećim primjerom:

```
>>> from decimal import Decimal
>>> a = Decimal('0.1')                                          #3
>>> b = Decimal('0.3')                                          #4
>>> print('3 * {} - {} = {}'.format(a, b, 3 * a - b))           #5
3 * 0.1 - 0.3 = 0.0
>>> c = 0.1                                                     #6
>>> d = 0.3                                                     #7
>>> print('3 * {} - {} = {}'.format(c, d, 3 * c - d))           #8
3 * 0.1 - 0.3 = 5.551115123125783e-17
>>> print('3 * {} - {} = {:34.32f}'.format(c, d, 3 * c - d))    #9
3 * 0.1 - 0.3 = 0.00000000000000005551115123125783
```

Brojevi `a` i `b` su naredbama (#3) i (#4) definirani kao decimalni brojevi modula `decimal`. Naredbom (#5) uvjerili smo se da je rezultat operacije `3 * a - b` jednak nuli. Brojevi `c` i `d` su naredbama (#6) i (#7) definirani kao tip `float`. Izvođenjem naredbe (#8) vidimo da rezultat obavljanja aritmetičke operacije `3 * 0.1 - 0.3` ovaj puta nije nula već neki vrlo mali broj. Ako rezultat ispišemo u `f` formatu (#9) vidimo da se prva znamenka različita od nule nalazi na sedamnaestom mjestu iza decimalne točke. Prema tome, zaokruživanjem na manji broj decimalnih mjesta ili u ispisu s manjim brojem decimalnih mjesta ta mala pogreška neće biti uopće zamjetljiva.

U odjeljku 9.3.2. naredbama smo ispisali vrijednosti razlomaka u decimalnom obliku. Pritom smo morali uvesti modul `decimal` i uvesti decimalni prikaz broja 1. Ovdje to nije potrebno i jednostavno možemo pisati:

```
>>> for i in range(1,21):
        print('1/{0:2} = {1}'.format(i, 1/i))

1/ 1 = 1.0
1/ 2 = 0.5
1/ 3 = 0.3333333333333333
1/ 4 = 0.25
1/ 5 = 0.2
1/ 6 = 0.16666666666666666
1/ 7 = 0.14285714285714285
1/ 8 = 0.125
1/ 9 = 0.1111111111111111
1/10 = 0.1
1/11 = 0.09090909090909091
1/12 = 0.08333333333333333
1/13 = 0.07692307692307693
1/14 = 0.07142857142857142
1/15 = 0.06666666666666667
1/16 = 0.0625
1/17 = 0.058823529411764705
1/18 = 0.05555555555555555
1/19 = 0.05263157894736842
1/20 = 0.05
```

Modul `decimal` nam je omogućio računanje i ispis decimalnih brojeva s dvadeset i osam znamenaka. Ovdje možemo prebrojati sedamnaest ili osamnaest znamenki. Taj će nam broj znamenaka biti i više nego dovoljan pri rješavanju naših problema. Najčešće će nas zanimati samo nekoliko znamenaka iza decimalne točke.

Još jednom treba naglasiti da tip `decimal` koji smo upoznali omogućuje računanje s decimalnim brojevima na način na koji smo naučili u matematici, tj. da se osigura veća točnost. Sva se ta dobra svojstva postižu programskim funkcijama u modulu `decimal` i zbog toga se takvo računanje u računalu obavlja mnogo sporije od računanja s brojevima tipa `float` koje se obavlja neposredno u elektroničkim sklopovima za obavljanje aritmetičkih operacija računalnog procesora.

Mi ćemo od sada nadalje uglavnom za prikaz racionalnih brojeva rabiti tip `float` jer će nam biti mnogo jednostavnije pisati programe.

Module `fractions` i `decimal` ima smisla rabiti samo onda kada želimo provjeriti matematičku korektnost naših izračunavanja koja provodimo bez računala.