Rukovanje brojevima

Slajdovi za predmet Osnove programiranja

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2014.

Ciljevi

- razumevanje pojma tipova podataka
- poznavanje osnovnih numeričkih tipova podataka u Pythonu
- razumevanje osnovnih principa reprezentacije brojeva u računaru
- korišćenje Pythonove `math` biblioteke
- razumevanje akumulator šablona
- razumevanje i pisanje programa koji operišu numeričkim podacima

Numerički tipovi podataka

- informacije koje obrađuje računar su predstavljene u obliku podataka
- dve vrste brojeva:
  - celi brojevi – nemaju razlomljeni deo, npr. 5, 4, 3, 6
  - decimalni ostaci – brojevi između 0 i 1, npr. .25, .1, .05, .01

Numerički tipovi podataka 2

- celi i razlomljeni brojevi su različito predstavljeni unutar računara!
- kažemo da su celi i razlomljeni brojevi različitog tipa
- tip podatka određuje koje vrednosti promenljiva može imati i koje operacije možemo izvršiti nad njima

Numerički tipovi podataka 3

- celi brojevi se predstavljaju tipom podataka integer (kraće int)
- vrednosti tipa int mogu biti pozitivni i negativni celi brojevi i nula

Numerički tipovi podataka 4

- brojevi koji imaju razlomljeni deo predstavljaju se kao brojevi u pokretnom zarezu (float)
- kako ih razlikujemo?
  - numerički literal bez decimalne tačne predstavlja `int` vrednost
  - numerički literal sa decimalnom tačkom predstavlja `float` vrednost (makar sve decimale bile 0)

Numerički tipovi podataka 5

- Python ima posebnu funkciju type koja vraća tip date vrednosti

```
>>> type(3)
<class 'int'>
>>> type(3.1)
<class 'float'>
>>> type(3.0)
<class 'float'>
>>> myInt = 32
>>> type(myInt)
<class 'int'>
>>>
```

Numerički tipovi podataka 6

- zašto nam trebaju dve vrste brojeva?
  - brojači ne mogu biti razlomljeni
  - puno matematičkih algoritama su vrlo efikasni sa celim brojevima
  - tip `float` čuva samo aproksimaciju realnog broja
  - pošto `float` brojevi nisu egzaktni, treba koristiti `int` kad god je moguće

Numerički tipovi podataka 7

- operacije nad `int`-ovima proizvode `int`-ove, osim `/`
- operacije nad `float`-ovima proizvode `float`-ove

```
>>> 3.0+4.0
7.0
>>> 3+4
7
>>> 3.0*4.0
12.0
>>> 3*4
12
>>> 10.0/3.0
3.3333333333333335
>>> 10/3
3.3333333333333335
>>> 10 // 3
3
>>> 10.0 // 3.0
3.0
```

Numerički tipovi podataka 8

- celobrojno deljenje proizvodi ceo broj: `10//3 = 3`
- ostatak pri deljenju: `10%3 = 1`
- `a = (a//b)*b + a%b`

Paket `math`

- pored aritmetičkih operacija (+,-,*,/,//,**,%,abs) postoji još puno matematičkih funkcija u paketu `math`
- paket je skup korisnih funkcija

Korišćenje `math` paketa

- napišimo program koji izračunava korene kvadratne jednačine

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- jedini deo koji nam nedostaje je izračunavanje kvadratnog korena...
- za to postoji funkcija u paketu `math`

Korišćenje `math` paketa 2

- da bismo koristili paket, potreban nam je sledeći red u programu:
  `import math`
- `import`-i obično stoje na početku programa
- importovanje paketa čini funkcije iz tog paketa dostupnima u našem programu

Korišćenje `math` paketa 3

- funkciju `sqrt` pozivamo sa `math.sqrt(x)`
- „tačka-notacija“ kaže Pythonu da funkciju `sqrt` traži u paketu
- za izračunavanje korena možemo pisati:
  ```
  root = math.sqrt(b*b - 4*a*c)
  ```

Korišćenje math paketa 4

```
# quadratic.py
# Izračunava realne korene kvadratne jednačine
# Ilustruje korišćenje math paketa
# Pažnja: program puca ako nema realnih korena

import math  # čini funkcije iz paketa dostupnim

print("Izračunavanje korena kvadratne jednačine")
print()

a, b, c = eval(raw_input("Unesite koeficijente (a, b, c): "))

root = math.sqrt(b * b - 4 * a * c)
root1 = (-b + root) / (2 * a)
root2 = (-b - root) / (2 * a)

print()
print("Rešenja su:", root1, root2)
```

Korišćenje math paketa 5

```
$ python quadratic.py
Izračunavanje korena kvadratne jednačine

Unesite koeficijente (a, b, c): 3, 4, -1

Rešenja su: 0.215250437022 -1.54858377035
```

- Šta je ovo?

```
$ python quadratic.py
Izračunavanje korena kvadratne jednačine

Unesite koeficijente (a, b, c): 1, 2, 3

Traceback (most recent call last):
 File "/home/branko/pajton/quadratic.py", line 14, in -toplevel-
    root = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
```

Korišćenje `math` paketa 6

- ako je $a = 1, b = 2, c = 3$, pokušavamo da izračunamo koren negativnog broja
- umesto `math.sqrt` možemo da upotrebimo `**`
- kako iskoristiti `**` za izračunavanje korena?

Akumuliranje rezultata: faktorijel

- čekaš u redu sa još 5 ljudi; na koliko načina se može poređati 6 ljudi?
- permutacije 6 elemenata
- ukupan broj permutacija = 6! (6 faktorijel) = 720
- faktorijel se definiše ovako: $n! = n(n-1)(n-2)...(1)$
- prema tome $6! = 6 * 5 * 4 * 3 * 2 * 1 = 720$

Akumuliranje rezultata: faktorijel $_2$

- kako napisati program za računanje faktorijela?

1. uneti broj čiji faktorijel se računa
2. izračunati faktorijel
3. ispisati rezultat

Akumuliranje rezultata: faktorijel $_3$

- kako smo izračunali 6! ?

1. 6*5 = 30
2. uzeti tih 30, i pomnožiti 30 * 4 = 120
3. uzeti tih 120, i pomnožiti 120 * 3 = 360
4. uzeti tih 360, i pomnožiti 360 * 2 = 720
5. uzeti tih 720, i pomnožiti 720 * 1 = 720

Akumuliranje rezultata: faktorijel $_4$

- ponavljamo množenje i pamtimo međurezultat
- ovakvi algoritmi se zovu akumulatori, jer postepeno gradimo (akumuliramo) rezultat
- međurezultati i konačni rezultat se čuvaju u akumulatorskoj promenljivoj

Akumuliranje rezultata: faktorijel $_5$

- opšti oblik akumulatorskog algorima izgleda ovako:

1 inicijalizuj akumulatorsku promenljivu
2 ponavljaj narednu operaciju dok se ne dobije konačan rezultat
2a ažuriraj vrednost akumulatorske promenljive
3 rezultat je u akumulatorskoj promenljivoj

Akumuliranje rezultata: faktorijel $_6$

- trebaće nam petlja

```
fact = 1
for i in (6,5,4,3,2,1):
    fact = fact * i
```

- da probamo ovo ručno da izvršimo

Akumuliranje rezultata: faktorijel 7

- zašto smo inicijalizovali `fact` na 1?
  - u svakom ciklusu petlje prethodna vrednost `fact` se koristi za računanje sledeće
  - u prvom prolazu `fact` će imati vrednost zahvaljujući inicijalizaciji
- ako bismo izostavili inicijalizaciju šta bi se desilo?

Akumuliranje rezultata: faktorijel 8

- množenje je komutativno, možemo napisati program ovako:

```
fact = 1
for i in (2,3,4,5,6):
    fact = fact * i
```

- šta ako želimo faktorijel nekog drugog broja?

Akumuliranje rezultata: faktorijel $_9$

- šta vraća `range(n)`?
  0, 1, 2, 3, ..., n-1
- `range` može imati još parametara; `range(start, n)` vraća
  start, start+1, ..., n-1
- `range(start, n, step)` vraća
  start, start+step, ..., n-1

Akumuliranje rezultata: faktorijel $_{10}$

- pomoću `range` funkcije možemo napraviti našu `for` petlju na više načina
- možemo brojati od 2 do n:
  `range(2, n+1)` (zašto n+1?)
- možemo brojati od n do 2:
  `range(n, 1, -1)`

Akumuliranje rezultata: faktorijel 11

- kompletan program za računanje faktorijela

```
# factorial.py
# Program računa faktorijel datog broja
# Ilustruje petlju sa akumulatorom

n = eval(raw_input("Unesite prirodan broj: "))
fact = 1
for i in range(n,1,-1):
    fact = fact * i
print("Faktorijel od", n, "je", fact)
```

Granice int-a

- koliko je 100! ?

```
$ python factorial.py
Unesite prirodan broj: 100
Faktorijel od 100 je 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

- prilično veliki broj

Granice int-a 2

- novije verzije Pythona mogu da izračunaju 100!
- u starijim verzijama ovaj program će pući već za n=13
- `OverflowError: integer multiplication`

Granice `int`-a 3

- iako postoji beskonačno mnogo celih brojeva, samo konačan broj njih se može predstaviti pomoću `int`-a
- opseg mogućih vrednosti zavisi od broja bita koje CPU koristi za predstavljanje `int`-a
- tipično se za `int` koristi 32 bita

Granice `int`-a 3

- iako postoji beskonačno mnogo celih brojeva, samo konačan broj njih se može predstaviti pomoću `int`-a
- opseg mogućih vrednosti zavisi od broja bita koje CPU koristi za predstavljanje `int`-a
- tipično se za `int` koristi 32 bita
- to znači da ima 2^{32} mogućih vrednosti, sa 0 u sredini
- opseg je od -2^{31} do $2^{31} - 1$
  (ovo -1 je zbog nule koja takođe zauzima mesto)
- 100! je mnogo veće od 2^{31}; kako to radi?

Rukovanje velikim brojevima

- ako koristimo `float` umesto `int`-a?
- kada inicijalizujemo akumulator na 1.0 dobijamo

```
$ python factorial.py
Unesite prirodan broj: 15
Faktorijel od 15 je 1.307674368e+012
```

- više ne dobijamo potpuno tačan rezultat!

Rukovanje velikim brojevima 2

- vrlo veliki ili vrlo mali brojevi se izražavaju u eksponencijalnoj notaciji
- 1.307674368e+012 predstavlja broj $1.307674368 \cdot 10^{12}$
- ovim množenjem se decimalna tačka pomera za 12 mesta u desno
- ali broj ima 9 decimala
- 3 cifre su izgubljene!

Rukovanje velikim brojevima 3

- `float`-ovi su aproksimacije
- pomoću njih možemo prikazati veći opseg brojeva, ali sa manjom preciznošću
- (noviji) Python ima rešenje – prošireni `int`-ovi
- fleksibilnost na račun brzine i potrošnje memorije

Konverzije tipova

- kombinovanje `int`-a i `int`-a proizvodi `int`
- kombinovanje `float`-a i `float`-a proizvodi `float`
- šta se dešava kada ih pomešamo?

```
x = 5.0 + 2
```

- šta bi trebalo da bude rezultat?

Konverzije tipova 2

- morali bismo ili konvertovati 5.0 u 5 i obaviti `int`-sabiranje
- ili konvertovati 2 u 2.0 i obaviti `float`-sabiranje
- kovertovanje `float` u `int` (u opštem slučaju) nosi gubitak podataka
- `int`-ovi se lako konvertuju u `float` dodavanjem .0

Konverzije tipova 3

- u mešovitim izrazima Python će automatski konvertovati `int` $\rightarrow$ `float`
- nekada mi želimo da kontrolišemo konverziju tipa
  – eksplicitna tipizacija

Konverzije tipova 4

```
>>> float(22//5)
4.0
>>> int(4.5)
4
>>> int(3.9)
3
>>> round(3.9)
4
>>> round(3)
3
```