

Kolekcije podataka

Slajdovi za predmet Osnove programiranja

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2020.

Ciljevi

- upotreba lista (nizova) za reprezentaciju kolekcije povezanih podataka
- poznavanje funkcija za manipulaciju Python listama
- upotreba lista za upravljanje podacima
- upotreba lista i rečnika za organizaciju složenih podataka

Primer: jednostavna statistika

- mnogi programi rukuju velikom količinom sličnih podataka
 - reči u tekstu
 - spisak studenata
 - podaci iz eksperimenta
 - poslovni partneri
 - grafički objekti na ekranu
 - karte u špilu

Primer od ranije

- izračunavanje proseka za niz brojeva, korišćenjem praznog stringa kao sentinela

```
def main():  
    sum = 0.0  
    count = 0  
    xStr = input("Unesite broj (<Enter> za kraj) >> ")  
    while xStr != "":  
        x = eval(xStr)  
        sum = sum + x  
        count = count + 1  
        xStr = input("Unesite broj (<Enter> za kraj) >> ")  
    print("\nProsek je", sum / count)
```

Primer od ranije ₂

- unosimo niz brojeva, ali program ne pamti **sve** brojeve koje smo uneli
- pamti samo tekuću sumu unetih brojeva
- sada je potrebno izračunati prosek, **median** i **standardno odstupanje**

Median

- **median** je vrednost koja deli skup brojeva na dva jednako velika dela
- za brojeve 2, 4, 6, 9, 13 median je 6
 - ima dve vrednosti manje od 6 i dve veće od 6
 - broj 6 „deli“ ovaj niz na dva jednaka dela
 - broj 6 nalazi se u sredini sortiranog niza brojeva
- izračunavanje mediana:
 - unesi sve brojeve
 - sortiraj po veličini
 - izdvoj srednji broj (ne srednju vrednost!)

Standardno odstupanje

- **standardno odstupanje** je mera koliko podaci odstupaju od prosečne vrednosti
- ako su svi podaci „blizu“ prosečne vrednosti, odstupanje je malo
- ako su podaci „rasuti“, odstupanje je veliko
- izraz za standardno odstupanje:

$$s = \sqrt{\frac{\sum (\bar{x} - x_i)^2}{n - 1}}$$

- \bar{x} je prosek
- n je broj podataka
- x_i je i -ti podatak
- izraz $(\bar{x} - x_i)^2$ je kvadrat „odstupanja“ pojedinačnog podatka od proseka

Standardno odstopanje ₂

- za niz brojeva 2, 4, 6, 9, 13
- prosek je 6.8

$$(6.8 - 2)^2 + (6.8 - 4)^2 + (6.8 - 6)^2 + (6.8 - 9)^2 + (6.8 - 13)^2 = 149.6$$

$$s = \sqrt{\frac{149.6}{5 - 1}} = \sqrt{37.4} = 6.11$$

Standardno odstupanje ₃

- za izračunavanje standardnog odstupanja potreban je prosek
 - (koji ne možemo izračunati dok nemamo sve podatke)
 - ali i svaki pojedinačni podatak!
-
- treba nam način da zapamtimo sve podatke koji se unose

Lista

- treba nam rešenje za skladištenje čitave kolekcije brojeva
- ne možemo da koristimo posebne promenljive jer ne znamo unapred koliko će brojeva biti
- treba nam način da **ceo niz** brojeva tretiramo kao jednu promenljivu

Python lista

- Python [lista](#) je uređena sekvenca podataka
- na primer, sekvenca n brojeva može se nazvati S :

$$S = s_0, s_1, s_2, \dots, s_{n-1}$$

- pojedine vrednosti u listi možemo dobiti pomoću [indeksa](#)
- korišćenjem indeksa, možemo sažeto zapisati matematičke operacije nad svim elementima liste
- na primer, zbir svih elemenata liste:

$$\sum_{i=0}^{n-1} s_i$$

Python lista ₂

- neka je sekvenca smeštena u promenljivoj `s`
- petlja za sumiranje svih elemenata liste:

```
sum = 0
for i in range(n):
    sum = sum + s[i]
```

- skoro svi programski jezici poznaju ovakvu strukturu
- negde se zove `niz`

Python lista ₃

- **lista** (niz) je sekvenca vrednosti gde cela sekvenca ima svoj identifikator (npr. `s`)
- gde se elementima sekvence pristupa pomoću indeksa (npr. `s[i]`)
- u drugim programskim jezicima nizovi su obično **fiksne dužine** i **homogeni**
 - kada se niz inicijalizuje, mora se navesti max broj elemenata
 - svi elementi niza su istog tipa

Python lista ₄

- liste u Pythonu nemaju unapred određen broj elemenata
 - mogu da se „produžavaju“ i „skraćuju“ po potrebi
- liste u Pythonu su heterogene
 - jedna lista može sadržati elemente različitog tipa
- sadržaj liste se može menjati ([mutable](#))

Operacije nad listama

operator	značenje
<code><list> + <list></code>	konkatenacija
<code><list> * <int-expr></code>	ponavljanje
<code><list>[]</code>	indeksiranje
<code>len(<list>)</code>	dužina liste
<code><list>[:]</code>	isecanje
<code>for <var> in <list>:</code>	iteracija
<code><expr> in <list></code>	postojanje elementa

Operacije nad listama 2

- osim poslednje operacije, sve ostale su dostupne i za stringove
- operacija koja testira postojanje elementa:

```
>>> lst = [1,2,3,4]
```

```
>>> 3 in lst
```

```
True
```

- za razliku od stringova, liste se mogu menjati

```
>>> lst = [1,2,3,4]
```

```
>>> lst[2] = 7
```

```
>>> lst
```

```
[1, 2, 7, 4]
```


Operacije nad listama ₃

- sumiranje elemenata liste možemo ovako zapisati:

```
sum = 0
for x in s:
    sum = sum + x
```

- lista sa jednakim elementima:

```
>>> zeroes = [0] * 50
```

Operacije nad listama ₄

- liste se često popunjavaju jedan-po-jedan element
- funkcija `append` dodaje element na kraj liste

```
nums = []  
x = eval(input('Unesite broj: '))  
while x >= 0:  
    nums.append(x)  
    x = eval(input('Unesite broj: '))
```

- ovde se `nums` koristi kao akumulator!
- inicijalizuje se kao prazna lista
- u svakom ciklusu petlje dodaje se jedan element liste

Još operacija nad listama

funkcija	značenje
<code><list>.append(x)</code>	dodaj element <code>x</code> na kraj liste
<code><list>.sort()</code>	sortiraj listu
<code><list>.reverse()</code>	obrni redosled elemenata u listi
<code><list>.index(x)</code>	indeks prve pojave vrednosti <code>x</code> u listi
<code><list>.insert(i, x)</code>	ubaci <code>x</code> u listu na mesto indeksa <code>i</code>
<code><list>.count(x)</code>	broj pojavljivanja vrednosti <code>x</code> u listi
<code><list>.remove(x)</code>	ukloni prvu pojavu vrednosti <code>x</code> iz liste
<code><list>.pop(i)</code>	izbaci <code>i</code> -ti element iz liste i vrati njegovu vrednost

Još operacija nad listama 2

```
>>> lst = [3, 1, 4, 1, 5, 9]
>>> lst.append(2)
>>> lst
[3, 1, 4, 1, 5, 9, 2]
>>> lst.sort()
>>> lst
[1, 1, 2, 3, 4, 5, 9]
>>> lst.reverse()
>>> lst
[9, 5, 4, 3, 2, 1, 1]
>>> lst.index(4)
2
>>> lst.insert(4, "Hello")
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1, 1]
>>> lst.count(1)
2
>>> lst.remove(1)
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1]
>>> lst.pop(2)
4
>>> lst
[9, 5, 3, 'Hello', 2, 1]
```

Još operacija nad listama ₃

- većina ovih funkcija ne vraća ništa, već menja sadržaj liste
- lista se može produžavati dodavanjem novih vrednosti
- ili skraćivati izbacivanjem postojećih
- elementi ili čitavi isecci se mogu izbaciti pomoću operatora `del`

```
>>> myList=[34, 26, 0, 10]
```

```
>>> del myList[1]
```

```
>>> myList
```

```
[34, 0, 10]
```

```
>>> del myList[1:3]
```

```
>>> myList
```

```
[34]
```

Principi rada sa Python listama

- lista je sekvenca podataka koja se čuva kao jedan objekat
- elementi liste su dostupni putem indeksa
- podliste su dostupne putem isecanja
- liste se mogu menjati
 - elementi ili isečki se mogu zameniti pomoću dodele vrednosti
 - nije moguće kod stringova!
- liste se mogu produžavati i skraćivati po potrebi
- postoje gotove funkcije za manipulaciju listama

Statistika sa listama

- rešićemo naš statistički problem tako što ćemo čuvati sve podatke u listi
- napisaćemo funkcije koje računaju prosek, odstupanje i median za datu listu
- rešavamo problem korak-po-korak

Funkcija za unos podataka: `getNumbers`

- unos brojeva pomoću sentinel petlje
- inicijalno prazna lista će biti akumulator
- vraćamo listu kao rezultat funkcije kada se unesu svi podaci
- upotreba funkcije:
`data = getNumbers()`

Funkcija za unos podataka: `getNumbers` ₂

```
def getNumbers():  
    nums = []          # počnemo sa praznom listom  
  
    # sentinel petlja za unos brojeva  
    xStr = input("Unesi broj (<Enter> za kraj) >> ")  
    while xStr != "":  
        x = eval(xStr)  
        nums.append(x)    # dodaj vrednost na kraj liste  
        xStr = input("Unesi broj (<Enter> za kraj) >> ")  
    return nums
```

Funkcija za izračunavanje proseka: mean

- ulaz: lista brojeva
- izlaz: prosečna vrednost elemenata liste

```
def mean(nums):  
    sum = 0.0  
    for num in nums:  
        sum = sum + num  
    return sum / len(nums)
```

Funkcija za izračunavanje odstupanja: stdDev

- da bismo izračunali standardno odstupanje, treba nam prosek
- da li da izračunavamo prosek unutar stdDev?
- da li da vrednost proseka prenesemo kao parametar u stdDev?

Funkcija za izračunavanje odstupanja: stdDev₂

- izračunavanje proseka više puta unutar stdDev nije efikasno
- pošto naš program ispisuje i prosek i odstupanje,
- prenećemo prosek kao parametar u stdDev

```
def stdDev(nums, avg):  
    sumDevSq = 0.0          # akumulator  
    for num in nums:  
        dev = avg - num  
        sumDevSq = sumDevSq + dev * dev  
    return sqrt(sumDevSq/(len(nums)-1))
```

Funkcija za izračunavanje mediana: `median`

- nemamo formulu za računanje mediana: treba nam algoritam
- 1 sortiramo brojeve u rastućem redosledu
 - 2 srednji element niza je median
 - 3 ako niz ima paran broj elemenata, median je prosek dva srednja elementa

```
sortiraj brojeve u rastućem redosledu
if neparan broj elemenata:
    median = srednji element
else:
    median = prosek dva srednja elementa
return median
```

Funkcija za izračunavanje mediane: `median2`

```
def median(nums):  
    nums.sort()  
    size = len(nums)  
    midPos = size // 2  
    if size % 2 == 0:  
        median = (nums[midPos] + nums[midPos-1]) / 2  
    else:  
        median = nums[midPos]  
    return median
```

Glavni program

```
def main():  
    print("Izračunava prosek, median i std. odstupanje.")  
  
    data = getNumbers()  
    avg = mean(data)  
    std = stdDev(data, avg)  
    med = median(data)  
  
    print("\nProsek je", avg)  
    print("Standardno odstupanje je", std)  
    print("Median je", med)
```

Sekvenca

- sekvenca je **immutable** lista
- ne možemo joj menjati sadržaj
- umesto uglastih zagrada [] koristimo obične ()

```
>>> lst = [1,2,3]
```

```
>>> lst[0] = 7
```

```
>>> lst
```

```
[7, 2, 3]
```

```
>>> sek = (1,2,3)
```

```
>>> sek[0] = 7
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item assignment
```


Sekvenca ₂

- Python efikasnije koristi sekvence od listi
- treba koristiti sekvence kad je to moguće
- string se ponaša kao sekvenca

Rečnik

- rečnik (dictionary), za razliku od liste, ne poznaje redosled elemenata
- elementi rečnika nisu dostupni putem indeksa, već putem ključa
- rečnik čuva parove ključ-vrednost

```
>>> lozinke = {"guido": "superprogrammer",  
              "turing": "genius", "bill": "monopoly"}  
>>> lozinke["guido"]  
'superprogrammer'  
>>> lozinke["bill"]  
'monopoly'
```

- šta je ovde ključ, a šta vrednost?

Rečnik ₂

- rečnici se mogu menjati

```
>>> lozinke["bill"] = "bluescreen"
```

```
>>> lozinke
```

```
{'turing': 'genius', 'bill': 'bluescreen', 'guido': \
'superprogrammer'}
```

- originalni redosled nije očuvan!

Rečnik ₃

- ključ može biti bilo koji **immutable** tip - broj, string
- vrednost može biti bilo kog tipa
- rečnici se retko javljaju u drugim programskim jezicima
- u Pythonu se često koriste
- efikasni su, mogu primiti stotine hiljada elemenata

Dodavanje elemenata u rečnik

- rečnik može biti prazan: {}
- dodavanje elementa u rečnik: upotrebi se novi ključ

```
>>> lozinke["newuser"] = "jasamnovi"  
>>> lozinke  
{'turing': 'genius', 'bill': 'bluescreen', 'guido': \br/>'superprogrammer', 'newuser': 'jasamnovi'}
```

Primer: učitavanje rečnika iz fajla

```
passwd = {}  
for line in open('passwords', 'r').readlines():  
    user, pass = string.split(line)  
    passwd[user] = pass
```

Operacije nad rečnikom

operacija	značenje
<code><dict>.has_key(<key>)</code>	vraća True ako postoji dati ključ
<code><dict>.keys()</code>	vraća listu ključeva
<code><dict>.values()</code>	vraća listu vrednosti
<code><dict>.items()</code>	vraća listu sekvenci (key, val)
<code>del <dict>[<key>]</code>	uklanja element iz rečnika
<code><dict>.clear()</code>	uklanja sve elemente iz rečnika

Primer: brojanje reči u tekstu

- program analizira tekst i prebrojava koliko se puta svaka reč pojavila u tekstu
- ignorišemo različite gramatičke oblike (posmatramo ih kao različite reči)
- možemo rešavati problem pomoću akumulatora
- samo nam treba akumulator za svaku reč koju pronađemo u tekstu
- ne znamo unapred koje ćemo reči pronaći
- nema smisla inicijalizovati akumulator za svaku moguću reč
- kako upotrebiti rečnik?

Primer: brojanje reči u tekstu ₂

- ključevi su stringovi sa rečima
- vrednosti su brojevi koliko se puta reč pojavila u tekstu
- kada naiđemo na reč u tekstu, inkrementiramo njen akumulator:

```
counts[word] = counts[word] + 1
```

Primer: brojanje reči u tekstu ₃

`counts[word] = counts[word] + 1`

- šta kada prvi put naiđemo na reč u tekstu?
- `counts[word]` ne postoji, dobićemo grešku
- \Rightarrow nova reč nema svoj element u rečniku
- kada naiđemo na novu reč, moramo dodati novi element u rečnik

Dodavanje novog elementa u rečnik

- varijanta 1: prvo proverimo da li reč postoji u rečniku

```
if counts.has_key(word):  
    counts[word] = counts[word] + 1  
else:  
    counts[word] = 1
```

- varijanta 2: pokušamo da inkrementiramo akumulator; ako pukne, dodamo novi element u rečnik

```
try:  
    counts[word] = counts[word] + 1  
except KeyError:  
    counts[word] = 1
```

Učitavanje teksta i podela na reči

```
fname = input("Ime fajla: ")

# učitaj fajl kao dugačak string
text = open(fname, "r").read()

# pretvori sva slova u mala
text = string.lower(text)

# zameni svaki znak interpunkcije razmakom
for ch in "!\"#$%&()*+,-./:;<=>?@[\\]^_`{|}":
    text = text.replace(ch, " ")

# podeli string na reči
words = string.split(text)
```

Brojanje parsiranih reči

```
counts = {}  
for w in words:  
    try:  
        counts[w] = counts[w] + 1  
    except KeyError:  
        counts[w] = 1
```

Ispis rezultata

```
# uzmi listu reči
uniqueWords = counts.keys()

# sortiraj listu reči
uniqueWords.sort()

# ispiši svaku reč i broj pojavljivanja
for w in uniqueWords:
    print(w, counts[w])
```

Sortiranje

```
uniqueWords.sort()
```

- reči će nam biti sortirane po abecedi
- kako da sortiramo reči po broju pojavljivanja?
- sort može da primi parametar – funkciju za poređenje

```
uniqueWords.sort(compareWords)
```

Sortiranje ₂

```
def compareWords((w1,c1), (w2,c2)):  
    if c1 > c2:  
        return -1  
    elif c1 == c2:  
        return cmp(w1, w2)  
    else:  
        return 1
```

- funkcija vraća -1 ako je prva reč „češća“ od druge
- vraća +1 ako je prva reč „ređa“ od druge
- ako se reči pojavljuju jednako često, tada ih uporedi po abecedi

Sortiranje ₃

```
uniqueWords.sort(compareWords)
```

- funkcija `compareWords` se koristi kao parametar
- nema zagrada – `compareWords` se **ne poziva!**
- ova funkcija, data kao parametar funkciji `sort`, biće pozvana unutar `sort`

Ispis rezultata

- ispisujemo prvih n najčešćih reči

```
for i in range(n):  
    print("{0:<10}{1:>6}".format(uniqueWords[i],  
                                counts[uniqueWords[i]]))
```