# Algoritmi sa stringovima i listama

## Konverzione funkcije: str, int, float, i list

Funkcije str, int, float, i list se koriste za konverziju jednog tipa podataka u drugi tip.

### str

Često želimo da konvertujemo broj u string sa ciljem da iskoristimo string metode da broj razbijemo na delove. Built-in funkcija str se koristi da neki podatak pretvorimo (konvertujemo) u string. Evo nekoliko primera:

| Naredba | Rezultat |
|---|---|
| str(37) | '37' |
| str(3.14) | '3.14' |
| str([1,2,3]) | '[1,2,3]' |

### int i float

Funkcija int konvertuje podatak u ceo broj (integer). Funkcija float konvertuje podatak u decimalan broj ( floating point) . Evo nekoliko primera.

| Naredba | Rezultat |
|---|---|
| int('37') | 37 |
| float('3.14') | 3.14 |
| int(3.14) | 3 |

Pri konverovanju decimalnog broja u ceo broj, funkcija int odbacuje sve posle decimalne tačke.

### list

Funkcija list uzima podatak koji može biti konvetovan u listu, pa od njega napravi listu. Evo dva primera kako se to radi.

```
list(range(5))
```

```
[0,1,2,3,4]
```

```
list('abc')
```

```
['a', 'b', 'c']
```

# Primeri

## Primer 1

Ispisati sve palindromski brojevi od 1 do 10000. Palindromski broj je broj koji je jednak kada se čita bilo sleva udesno bilo unazad, kao na primer brojevi 1221 ili 64546.

```
for i in range(1,10001):
    s = str(i)
    if s==s[::-1]:    #moze i obrtanje stringa
        print(s)
```

Ovde smo iskoristili str funkciju da broj i konvertujemo u string kako bi mogli da ga segmentiramo .

## Primer 2

Napisati program koji ispisuje koliko ima godina osoba rođena 1 Januara, 1981 (u 2018 godini).

```
datum_rodjenja = '1 Januar, 1981'
godina_rodjenja = int(datum_rodjenja[-4:])
print('Vi imate', 2018-godina_rodjenja, 'godina')
```

Godina je u poslednja 4 znaka (character-a) u varijabli datum_rodjenja. Koristimo int funkciju da konvertujemo ova 4 znaka u ceo broj kako bi mogli da sa njim vršimo matematičku operaciju oduzimanja.

## Primer 3

Napišite program koji za učitani broj num sabira njegove cifre. Na primer, ako je učitani broj 47, program treba da odgovori sa 11 (4+7). Počnimo sa prostim primerom kada je broj dvocifren.

```
s = str(num)
odgovor = int(s[0]) + int(s[1])
```

Ideja je da konvertujemo num u string tako da možemo da indeksiramo pojedinačno cifre . Zatim svaku cifru konverujemo nazad u broj int za potrebe sumiranja. Evo sada verzije koja radi za brojeve sa proizviljnim brojem cifara:

```
s = str(num)
```

```
odgovor = 0
for i in range(len(s)):
    odgovor = odgovor + int(s[i])
```

**Ceo gornji program se može napisati u sam jednoj liniji korišćenjem tehnike za sastavljanje (comprehension) liste.**

```
odgovor = sum([int(c) for c in str(num)])
```

**Primer 4**

Da razbijemo decimalni broj num, na njegov celobrojni i decimalni deo, možemo upotrebiti sledeće naredbe:

```
celi_deo = int(num)
deci_deo = num – int(num)  # ili dec_deo = num - celi_deo
```

Na primer ako je num jednako 12.345, tada je celi_deo 12 a deci_deo 12.345-12=0.345.

**Primer 5**

Ako želimo da vidimo da li je neki broj prost, to možemo da uradimo proveravajući da li postoji neki njegov delioc različit od 1 i samog tog broja. Znamo da delioce možemo trazitu sledećom petljom:

```
for i in range(2,num):
```

Ovom petljiom bi se vršila provera za sve cele brojeve 2,3,…,num-1. Međutim, ispostavlja se da je dovoljno ispitati deljivost brojevima od 2 do kvadratnog korena broja num. Na primer, da proverimo da li je 111 prost broj, dovoljno je da proverimo da li je on deljiva sa brojevima od 2 do 10, jer je $\sqrt{111} \approx 10.5$. Zato možemo pokušati sa petljom:

```
for i in range(2,num**.5):
```

Međutim to bi prouzrokovalo grešku jer num**.5 može biti da nije ceo broj, a range funkcija traži cele brojeve. Korišćenjem funkcije int ispravljamo ovu grešku:

```
for i in range(2,int(num**.5)+1):
```

Dodajemo +1 na kraj jer range funkcija ne uključuje poslednju vrednost.

Primer 6
https://petlja.org/biblioteka/r/problemi/Zbirka/redni_broj_maksimuma

# Redni broj maksimuma

| vreme | memorija | ulaz | izlaz | test primeri |
|---|---|---|---|---|
| 1 s | 64 Mb | standardni ulaz | standardni izlaz | |

Na aukciji neke slike učestvuje $n$ kupaca. Svaki kupac ponudi izvesni iznos novca, pri tome svi kupci su ponudili različite iznose. Napisati program kojim se određuje redni broj kupca koji je ponudio najveći iznos.

## Ulaz

Prva linija standarnog ulaza sadrži prirodan broj $n$ ($1 \leq n \leq 1000$) koji predstavlja broj kupaca. U narednih $n$ linija nalazi se po jedan pozitivan realan broj, ti brojevi predstavljaju iznose novca koju su ponudio kupci redom.

## Izlaz

U prvoj liniji standarnog izlaza prikazati redni broj kupca koji je ponudio najveći iznos novca.

## Primer

### Ulaz

```
5
123.23
234.45
100.23
345.00
128.80
```

### Izlaz

```
4
```

RESENJE 1 (C++ u Python)

```
# broj ponuda

n = int(input())
# ucitavamo iznos prve ponude
iznos = float(input())
# iznos i redni broj najvece ponude medju svim do sada ucitanim
# ponudama - za sada je to iznos prve ponude
maxIznos = iznos
rbMax = 1
# obradjujemo preostale ponude
for i in range(1, n):
    # ucitavamo iznos naredne ponude
    iznos = float(input())
```

```
    # ako je veci od do sada najveceg iznosa, azuriramo najveci iznos
    # i njegov redni broj
    if iznos > maxIznos:
        maxIznos = iznos
        rbMax = i
# ispisujemo redni broj najvece ponude
print(rbMax)
```

```
RESENJE 2 (pure Python, comprehension)
# broj ponuda
n = int(input())
# ucitavamo iznos prve ponude
iznosi = [float(input()) for i in range(n)]
# odredjujemo i ispisujemo redni broj najvece ponude
rbMax = iznosi.index(max(iznosi)) + 1
print(rbMax)
```

# Bulovske varijable (Booleans)

Bulovske varijable u Python-u su varijable koje mogu imati samo jednu od dve vrednosti, True ili False. Evo dva primera setovanja Bulovskih varijabli:

```
game_over = True
highlight_text = False
```

Bulovske varijable mogu učiniti vaš program mnogo čitljivijim. One se često koriste kao zastavice (flag) ili da označe dve različite opcije. Često se koriste u uslovima zadatim u if naredbama i while petljama:

```
if game_over:
    print('Bye!')
```

Ove naredbe su ekvivalentne:

```
if game_over:   ⇔     if game_over==True:

while not game_over:  ⇔   while game_over==False:
```

### Napomena

Uslovni izrazi rezultitraju bulovskim vrednostima i mogu se pridruživati varijablama. Na primer, sledeće pridruživanje daje varijabli vrednost True jer zraz 6==6 ima vrednost True.

```
x = (6==6)
```

Već smo se ranije susreli sa bulovskim vrednostima. Metod isalpha kod stringa vraća True ako je svaki znak (character) u stringu slovo, a False u suprotnom slučaju.

# Kompozitni operatori

## Dodela i aritmetika

Operacije kao što je count=count+1 se pojavljuje veoma često pa Python za takve operacije ima skraćeno pisanje. Evo nekoliko primera:

| Naredba | Skraćeno |
|---|---|
| count=count+1 | count+=1 |
| total=total-5 | total-=5 |
| prod=prod*2 | prod*=2 |

Postoje takođe slične notacije za operatore /=, %=, //=,  **=.

## Skraćenice za operaciju pridruživanja (=)

Umesto:

```
a = 0
b = 0
c = 0
```

Možemo pisati:

```
a = b = c = 0
```

Još jedna skraćenica za pridruživanje

Recimo da imamo listu L sa tri elementa i da želimo da te elemente pridružimo imenima varijabli. To možemo učiniti ovako:

```
x = L[0]
y = L[1]
z = L[2]
```

Umesto toga možemo uraditi i ovako:

```
x,y,z = L
```

Slično, možemo pridružiti istovremeno tri ili više varijabli i ovako:

```
x,y,z = 1,2,3
```

Kao što smo ranije videli možemo zameniti vrednosti među varijablama na sledeći način:

```
x,y,z = y,z,x
```

## Skraćenice kod uslova

Evo nekih pogodnih skraćenica za uslove:

| Naredba | Skraćeno |
|---|---|
| if a==0 and b==0 and c==0: | if a==b==c==0: |
| if 1<a and a<b and b<5: | if 1<a<b<5: |

## Lazy evaluation

Recimo da pišemo program koji pretražuje u listi reči one čije je peto slovo ‘z’. Možemo pokušati na sledeći način:

```
for w in words:
    if w[4]=='z':
        print(w)
```

Ali ovim ćemo u nekim slučajevima ( kada su reči kraće od 5 slova) proizvesti grešku tipa „string index out of range“. Sledeća if naredna će međutim raditi bez greške i u takvim situacijama:

```
if len(w)>=5 and w[4]=='z':
```

Na prvi pogled izgleda da ćemo i sada dobiti grešku za reči kraće od 5 slova jer opet proveravamo w[4]. Razlog što se greška sada neće pojaviti je u „lenjom izračunavanju“ . Python počinje sa ispitivanje prvog dela uslova to jest da li je len(w)>=5. Ako taj uslov nije ispunjen ( ako je False), tada je sigurno da je celi uslov False, pa nema razloga da se bavimo drugim delom uslova . Zato Python neće ni proveravati drugi deo uslova (w[4]==’z’) , pa se greška neće pojaviti.

Lenjo izračunavanju se javlja i kod or uslova. U tom slučaju, Python proverava prvi deo or uslova, pa ako je on True ceo uslov će biti True i Python neće proveravati drugi de u or izrazu..

Primer: https://petlja.org/biblioteka/r/problemi/Zbirka/abecedno_ogledalo

# Abecedno ogledalo

| vreme | memorija | ulaz | izlaz | test primeri |
|---|---|---|---|---|
| 1 s | 64 Mb | standardni ulaz | standardni izlaz | |

Katarina je odlučila da svojoj drugarici pošalje šifrovanu poruku, koja sadrži samo slova engleske abecede, cifre i interpunkcijske znake. Svako slovo će šifrovati posebno na osnovu narednih pravila. Mala slova se šifruju velikim slovima tako što se slovo `a` šifruje slovom `Z`, slovo `b` šifruje slovom `Y`, `c` slovom `X` itd., sve do slova `y` koje se šifruje slovom `B` i `z` koje se šifruje slovom `A`. Velika slova se šifruju potpuno analogno - od `A` koje se šifruje sa `z` do `Z` koje se šifruje sa `a`. Ostali karakteri se ne menjaju.

## Ulaz

Sa standardnog ulaza unosi se jedna linija teksta, završena karakterom tačka (karakterom `.`).

## Izlaz

Na standardni izlaz ispisati šifrovani tekst (bez karaktera tačka).

## Primer

### Ulaz

```
Zdravo svima.
```

### Izlaz

```
aWIZEL HERNZ
```

Resenje

IDEJA: Dakle, prvi zadatak je da čitamo karaktere jedan po jedan, dok ne dođemo do karaktera `'.'`. Jedna mogućnost bi bila da učitamo odjednom celu liniju. U jeziku Python jedan način je da se napravi petlja oblika

```
for c in input():
  if c == '.':
    break
```

U telu petlje potrebno je šifrovati svaki karakter. Prvo vršimo klasifikaciju na mala slova, velika slova i ostale karaktera. Ako je karakter klasifikovan kao malo slovo, vršimo njegovo šifrovanje i to tako što primetimo da će rastojanje između koda karaktera koji se šifruje od koda karaktera `'a'` biti jednako rastojanju između karaktera `'Z'` i rezultata. Na primer, ako je u pitanju karakter `'c'` njegovo rastojanje od karaktera `'a'` je dva, pa je

rezultat karakter čiji se kod dobija kada se broj dva oduzme od koda karaktera `'Z'` tj. dobija se karakter `'X'`. Dakle, mala slova se mogu šifrovati na osnovu veze `'Z' - (c - 'a')`, a velika slova na osnovu veze `'z' - (c-'A')`.

```
for c in input():
  if c == '.':
    break

  if c.islower():
      tc = chr(ord('Z') - (ord(c) - ord('a')))
  elif c.isupper():
      tc = chr(ord('z') - (ord(c) - ord('A')))
  else:
      tc = c
  print(tc, end='')
```

# Pisanje naredbe u više linija

Ponekad ćete pisati dugačke linije koda koji bi bio čitljiviji ako bi naredbu pisali u dva ili više redova. Da bi to ostvarili koristimo znak na kraju linije kao u sledećem primeru:

```
if 'a' in string or 'b' in string or 'c' in string \
   or 'd' in string or 'e' in string:
```

Vodite računa da na kraju linije iza znaka nema blanko znakova jer bi to vodilo ka sintaksnoj grešci.

Ako unosite listu, rečnik ili argumente funkcije, znak nije potreban. Na primer:

```
L = ['Joe', 'Bob', 'Sue', 'Jimmy', 'Todd', 'Frank',
     'Mike', 'John', 'Amy', 'Edgar', 'Sam']
```

# Naredba pass

Naredba pass nema nikakvo dejstvo ( ne čini ništa ). Verovali ili ne i takva naredba može biti korisna kako ćemo kasnije videti.

# Formatizovanje stringova

Prtpostavimo da pišemo program koji izračunava bakšiš od 25% na račun od 23.60. Kada izvršimo proračun dobijamo 5.9, ali bi želeli da rezultat bude 5.90, a ne 5.9. Evo kako to radimo:

```
a = 23.60 * 25/100
print('Bakšiš je {:.2f}'.format(a))
```

Ovde smo iskoristili format metod za string. Evo još jedan primer:

```
racun = 23.60
baksis = 23.60*.25
print('Baksis: {:.2f}, Ukupno: {:.2f}'.format(baksis, racun+baksis))
```

Način na koji format metod funkcioniše je da stavimo par zagrada {} tamo gde želimo da formatizujemo vrednost. Argumenti u format metodi su vrednosti koje želimo da formatizujemo, gde prvi argument odgovara prvom paru zagrada, drugi argument drugom paru zagrada itd. Unutar svakog para zagrada stavlja se kod formatizovanja kojim se određuje kako će argument biti formatizovan.

## Formatizovanje celih brojeva

Za formatizovanje celih brojeva koristi se kod {:d}. Stavljanjem broja ispred d omogućava nam da cele brojeve poravnavamo udesno. Evo primera:

```
print('{:3d}'.format(2))
print('{:3d}'.format(25))
print('{:3d}'.format(138))

Izlaz:
  2
 25
138
```

Broj 3 ispred d označava da će za broj biti određena 3 mesta. Vrednost će biti štampana skroz udesno a preostala mesta na levo biće popunjena blako znacima. Ovakve stvari su korisne za lepše formatiziranje tabela sa brojevima.

Za pozicioniranje celih brojeva u centar (umesto udesno) koristite znak ^ , a za pozicioniranje ulevo znak <.

```
print('{:^5d}'.format(2))
print('{:^5d}'.format(222))
print('{:^5d}'.format(13834))

  2
 122
13834
```

Svaka od print naredbi dodeljuje 5 mesta za cele brojeve i pozicionira ih u centar tog dodeljenog prostora.

Stavljanjem zareza kod formatizacije dodaće celom broju zareze na mestima gde je to potrebno. Na primer da štampamo 1,000,000 možemo koristiti naredbu:

```
print('{:,d}'.format(1000000))
```

## Formatizovanje decimalnih brojeva

Za formatizovanje decimalnih brojeva koristimo kod {:f}. Da bi broj prikazali sa samo dve decimale koristimo kod {:.2f}. Promenom 2 u neki drugi broj menja se broj decimalnih mesta.

I decimalni brojevi se mogu pozicionirati udesno. Na primer {:8.2f} će rezervisati osam mesta od kojih je jedno za decimalnu tačku i dva mesta za deo posle decimalne tačke. Ako je vrednost koja se štampa 6.42, potrebna su samo četiri mesta pa će preostala mesta biti popunjena blanko znacima, a cela vrednost će biti pozicionirana udesno.

Znakovi ^ i < pozicioniraju i decimalne brojeve u centar i ulevo, slično kao i za cele brojeve.

## Formatizovanje stringova

Za formatizovanje stringova koristimo kod {:s}. Evo jednog primera koji centrira tekst:

```
print('{:^10s}'.format('Hi'))
print('{:^10s}'.format('there!'))
```

```
    Hi
  there!
```

Za pozicioniranje udesno koristimo znak > character:

```
print('{:>6s}'.format('Hi'))
print('{:>6s}'.format('There'))
```

```
    Hi
there!
```

Ima još puno stvari koje se mogu postići formatiziranjem.. Za sve opcije pri formatizaciji pogledajte Python documentation.

# Ugnježdene petlje

Možete dodavati petlje unutar drugih petlji. Za petlju koja se nalazi unutar druge petlje kažemo da je ugnježdena (nested), i možete, manje više, ugnjezditi petlje do proizvoljne dubine.

**Primer 1**

Štampanje tablice množenja, 10 × 10.

```
for i in range(1,11):
    for j in range(1,11):
        print('{:3d}'.format(i*j), end=' ')
    print()
```

Tablica množenja je dvodimenziona. Stoga koristimo dve for petlje, jedna za horizonatlni a druga za verikalni smer. Naredba print koristi formatizovanje udesno da bi tablica lepše izgledala. Na kraju print naredbe end=' ' nam omogućava da štampamo više podataka u istoj liniji. Kada završimo sa štampanjem jedne linije nerdbom print() se pomeramo na sledeću liniju.

**Primer 2**

Jedan od čestih matematičkih zadataka je da se nađu rešenja sistema jednačina. Ponekad želimo samo celobrojna rešenja, i to se može pokazati kao matematički problematično. Međutim, možemo napisati program koji traži ta rešenja metodom „brutalne sile“ (brute force). Ovde nalazimo rešenja (x,y) za sistem 2x+3y=4, x-y=7, gde su x i y između -50 i 50.

```
for x in range(-50,51):
        for y in range(-50,51):
        if 2*x+3*y==4 and x-y==7:
        print(x,y)
```

**Primer 3**

Pitagorina trojka je trojka brojeva (x,y,z) takvih da je $x^2+y^2=z^2$. Na primer (3,4,5) je Pitagorina trojka jer je $3^2+4^2=5^2$. Pitagorine trojke odgovaraju trouglovima čije su stranice celi brojevi (kao 3-4-5-trougao). Ovde je program koji pronalazi sve Pitagorine trojke (x,y,z) kada su x, y, z pozitivni brojevi manji od 100.

```
for x in range(1,100):
    for y in range(1,100):
        for z in range(1,100):
            if x**2+y**2==z**2:
```

```
                print(x,y,z)
```

Ako izvršite ovaj program, primetićete da postoje redundantna rešenja. Na primer pojavljuju se i rešenje (3,4,5) i (4,3,5) iako su ona zapravo jedno te isto. Da bi se oslobodili ove redundantnosti, promenimo drugu petlju da ide od x do 100. Na taj način kada je x jednako 4, na primer, prva vrednost za y će biti 4, a ne 1, pa zato nećemo dobiti redundantno rešenje (4,3,5). Takođe možemo promeniti i zadnju petlju da ide of y do 100.

Ako sada pogledate rešenja možete uočiti da su mnoga rešenja umnošci od drugih rešenja, kao što su, na primer, rešenja (6,8,9) i (9,12,15) umnošci od rešenja (3,4,5). Sledeći program pronalazi samo primitivne Pitagorine trojke, to jest one koje nisu unošci drugih trojki. To se postiže tako što kad god pronađemo novu trojku, proveravamo da li su x,y,i z deljivi sa istim brojem.

```
for x in range(1,100):
    for y in range(x,100):
        for z in range(y,100):
            if x**2+y**2==z**2:
                for i in range(2,x):
                    if x%i==0 and y%i==0 and z%i==0:
                        break
                else:
                    print((x,y,z), end=' ')
```

### Primer 4

Sastavljanje lista (list comprehensions) može da sadrži ugnježdene petlje. Sledeći primer kreira string Pyyttthhhhoooooonnnnnn:

```
'';.join([c*i for c in 'Python' for i in range(1,7)])
```

### Primer 5

У једном одељењу ученици су одлучили да у склопу новогодишње приредбе организују мало извлачење игре лото. Лото се игра тако што се у бубњу налази n куглица обележених бројевима од 1 до n и затим се из бубња извлачи одређени број куглица (извлачи се једна по једна куглица, а извучене куглице се не враћају поново у бубањ). Када се извуче тражени број куглица, бројеви који пишу на њима се поређају од најмањег до највећег. На пример, ако у бубњу има пет куглица и извлаче се три, могуће је да се прво извуче куглица 4, затим 1 и онда 2 - то извлачење се представља тројком бојева 1 2 4 (јер су након извлачења куглице поређане по величини). Да би повећали шансе за добитак, одлучили су да се увек из бубња извлаче само три куглице. Напиши програм који исписује које све могућности (комбинације) које на крају могу бити извучене (тј. све тројке бројева које их представљају), ако се зна да у бубњу има n различитих куглица, где је n број који се учитава са стандардног улаза ($3 \leq n \leq 9$). На стандардни излаз испиши све могућности, при чему су три броја у свакој комбинацији уређени од најмањег до највећег и раздвојени размаком, а комбинације се приказују лексикографским редом (што значи да би троцифрени бројеви који би се добили када би се размаци обрисали били поређани од најмањег до највећег).

*Примери*

**Примери**

| **Улаз:** | **Излаз:** | **Улаз:** | **Излаз:** |
|---|---|---|---|
| **5** | **1 2 3** | **4** | **1 2 3** |
| | **1 2 4** | | **1 2 4** |
| | **1 2 5** | | **1 3 4** |
| | **1 3 4** | | **2 3 4** |
| | **1 3 5** | | |
| | **1 4 5** | | |
| | **2 3 4** | | |
| | **2 3 5** | | |
| | **2 4 5** | | |
| | **3 4 5** | | |