Uvod u programski jezik Pajton - 2. deo

Uvod u merno-informacione sisteme

Liste

- Liste predstavljaju nizove objekata
- Liste:
  - su promenljive (elementi se mogu menjati)
  - su indeksirane odn. uređene (svaki element ima svoj indeks indeks prvog elemnta je 0)
  - mogu sadržati duplikate
- Liste čuvaju reference na objekte zbog čega se čuvani elementi mogu kombinovati
- I liste mogu biti članovi liste
- Nemaju fiksnu dužinu

Definicija i pristup elementima liste

- Liste se definišu pomoću uglastih zagrada []

```
1 a = [] #prazna lista
2 b = [1, 2, 5]
3 c = ['Pajton', 3, 2.72, 1+1j, [3, 'txt']]
```

- Slično kao i kod stringova moguće je pristupiti odgovarajućem indeksu liste
- Može se pristupati elementima sa negativnim indeksom

```
1 lista = ['Pajton', 3, 2.72, 1+1j, [3, 'txt']]
2 print(lista[0]) #Pajton
3 print(lista[2]) #2.72
4 print(lista[-1]) #[3, 'txt']
5 print(lista[-1][1]) #txt
```

Izmena elemenata i pristup opsegu liste

- Moguće je elementu liste izmeniti vrednost

```
1 lista = [1, 2, 4, 4, 5]
2 lista[2]=3
3 print(lista) #[1, 2, 3, 4, 5]
```

- Ako se pokuša pristupiti nepostojećem elemetnu liste dobija se greška *IndexError*
- Kao i kod stringa moguće je pristupiti opsegu liste

```
1 lista = [1, 2, 3, 4, 5, 6]
2 print(lista[2:-2]) #[3, 4]
3 print(lista[::2]) #[1, 3, 5]
```

Iteriranje kroz elemente liste

- Iteriranje kroz elemente liste se vrši for petljom

```
1 #
2 lista = [1, 2, 3, 4, 5, 6]
3 for i in lista:
4     print(lista)
```

- Moguće je izvršiti pristup po indeksu

```
1 lista = [1, 2, 3, 4, 5, 6]
2 for i in range(len(lista)):
3     print('Element '+str(i)+' je: '+str(lista[i]))
4     #identican ispis pomocu f-stringa
5     #print(f'Element {i} je: {lista[i]}')
```

Metode za rad sa listama

Ime metode	Opis
append(el)	Dodaje prosleđeni element na kraj liste.
clear()	Briše sve elemente liste.
copy()	Pravi kopiju cele liste.
count(el)	Vraća broj elemenata koji su jednaki sa prosleđenom.
extend(list)	Proširuje postojeću listu prosleđenom.
index(el)	Vraća indeks prvog elementa jednakog prosleđenom.
insert(index,el)	Upisuje element na prosleđeni indeks.
pop(index)	Briše element na prosleđenoj poziciji.
remove(el)	Briše prvo pojavljivanje prosleđenog elementa u listi.
reverse()	Vraća listu u obrnutom redosledu.
sort()	Sortira listu, podrazumevano po abecednom redu.

```
1  lista = [1, 1, 2, 3]
2  lista.append(5) #[1, 1, 2, 3, 5]
3  lista1 = lista.copy() #lista1 = [1, 1, 2, 3, 5]
4  lista1.clear() #lista1=[]
5  lista.count(1) #2
6  lista.extend([4,5]) #[1, 1, 2, 3, 5, 4, 5]
7  lista.index(1) #0
8  lista.insert(0,0) #[0, 1, 1, 2, 3, 5, 4, 5]
9  lista.pop(2) #[0, 1, 2, 3, 5, 4, 5]
10 lista.remove(5) #[0, 1, 2, 3, 4, 5]
11 lista.reverse() #[5, 4, 3, 2, 1, 0]
12 lista.sort() #[0, 1, 2, 3, 4, 5]
13 lista = ['jedan', 'dva', 'tri']
14 lista.sort() #['dva', 'jedan', 'tri']
```

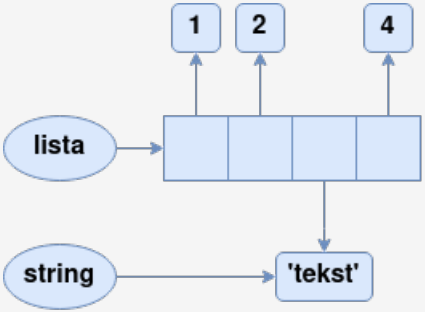
- Provera pripadnosti elementa kolekciji korišćenjem operatora in

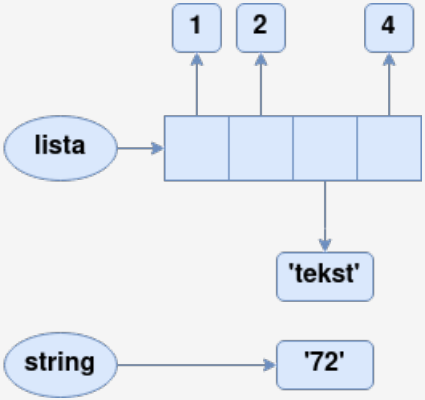
```
1 lista = [1, 2, 'a', 'b']
2 print('a' in lista) #True
3 print('c' in lista) #False
```

- Adresama svake promenljive moguće je pristupiti pozivom funkcije id()

```
1  string = 'tekst'
2  a = [1, 2, string, 4]
3
4  print(f'Adresa stringa {hex(id(string))}')
5  print(f'Adresa liste {hex(id(a))}')
6  for i in range(len(a)):
7      print(f'Adresa {i}. elementa {hex(id(a[i]))}')
8
9  string = '72'
10 print(f'Adresa stringa {hex(id(string))}')
11 print(a) #[1, 2, 'tekst', 4]
```

- Zašto se vrednost u listi nije promenila?

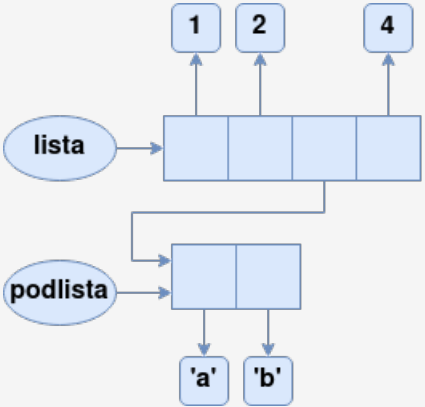


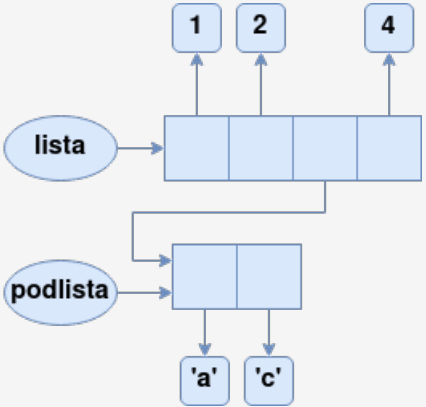


- String je nepromenljiv tip
- Šta bi se desilo da je element liste lista?

```python
podlista = ['a', 'b']
lista = [1, 2, podlista, 4]

print(f'Adresa podliste {hex(id(podlista))}')
print(f'Adresa liste {hex(id(lista))}')

for i in range(len(lista)):
    print(f'Adresa {i}. elementa {hex(id(lista[i]))}')

podlista[1] = 'c'
print(lista) #[1, 2, ['a', 'c'], 4]
```

- Zašto se vrednost u listi promenila?

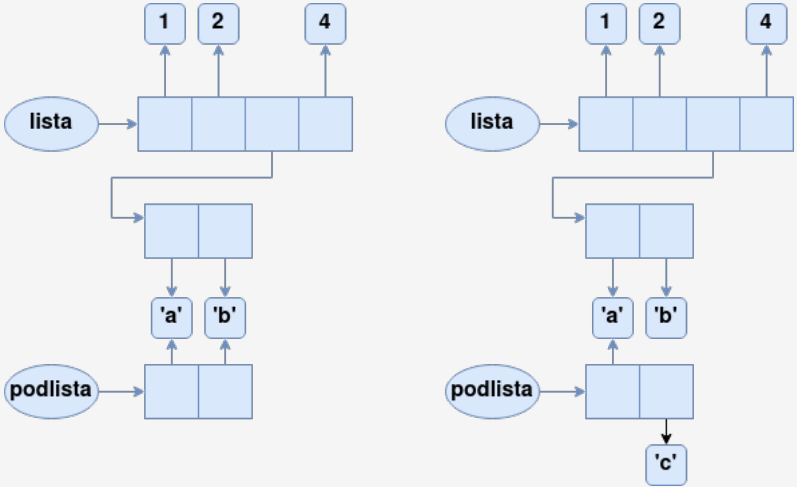




- Lista je promenljiv tip
- Kako izbeći ovakvo ponašanje? Rešenje: kopiranjem!

```python
podlista = ['a', 'b']
lista = [1, 2, podlista.copy(), 4]
#identican efekat ima i:
#lista = [1, 2, podlista[::], 4]

print(f'Adresa podliste {hex(id(podlista))}')
print(f'Adresa liste {hex(id(lista))}')

for i in range(len(lista)):
    print(f'Adresa {i}. elementa {hex(id(lista[i]))}')

podlista[1] = 'c'
print(lista) #[1, 2, ['a', 'b'], 4]
```

- Kopiranjem elemenata pravi se nezavisna kopija u memoriji



- Ovde se i dalje pravi plitka kopija
- Za složenije kolekcije podataka opet se može javiti problem
- Npr. da je u podlisti postojala lista i ako bi se ona modifikovala, rezultovala bi identičnim problemom.
- U prethodnom slučaju problem nije postojao jer su stringovi nepromenljivi tip podataka pa se svakom novom definicijom stringa pravi i novi objekat na drugoj memorijskoj lokaciji
- Kako bi se on prevazišao neophodno je praviti duboku kopiju

Torka (tuple)

- Torke predstavljaju nizove objekata
- Torke:
  - su **nepromenljive** (elementi se **ne mogu** menjati)
  - su indeksirane odn. uređene (svaki element ima svoj indeks, indeks prvog elemnta je 0)
  - mogu sadržati duplikate
- Kao i liste, torke čuvaju reference na objekte zbog čega se čuvani elementi mogu kombinovati
- Elementi torke mogu biti bilo koji objekti pa i one same
- Imaju fiksnu dužinu

Definicija i pristup elementima torke

- Torke se definišu pomoću zagrada () ili bez njih, a svi elementi se odvajaju zarezima
- Ako se želi definisati torka sa samo jednim elementom nakon elementa mora postojati zarez

```
1 a = () #prazna torka
2 b = 1, 2, 5
3 c = (1, 2, 5)
4 d = ('Podatak',)
```

- Moguće je pristupiti odgovarajućem indeksu torke
- Može se pristupati elementima sa negativnim indeksom

```
1 torka = ('Pajton', 3, 2.72, 1+1j, [3, 'txt'])
2 print(torka[0]) #Pajton
3 print(torka[-1]) #[3, 'txt']
4 print(torka[-1][1]) #txt
```

Izmena elemenata torke

- Pošto je torka nepromenjiv tip podataka nemoguće joj je promeniti vrednost
- U slučaju da se pokuša dodeliti vrednost dobiće se greška

```
1 a = (1, 2, 3)
2 a[1] = 'a'
3 #TypeError: 'tuple' object does not support item
      assignment
```

Metode za rad sa torkama

Ime metode	Opis
count(el)	Vraća broj elemenata koji su jednaki sa prosleđenom.
index(el)	Vraća indeks prvog elementa jednakog prosleđenom.

```
1 a = (1, 'a', 3, 3, 5)
2 a.index(3) #2
3 a.count(3) #2
```

Funkcije

- Kako bi se povećala preglednost koda, on se može dekomponovati u funkcije
- Funkcije predstavljaju logičke celine koje se kasnije mogu jednostavnije testirati i proširivati
- Definicija započinje naredbom *def* za kojom sledi ime funkcije kao i parametri koji se navode u zagradama (funkcija može biti i bez parametara)
- Telo funkcije definisano je blokom naredbi koji započinje operatorom :
- Svaka funkcija ima povratnu vrednost:
  - Ako se želi vratiti vrednost iz funkcije nakon naredbe *return* definišu se povratne vrednosti
  - Ako se ne otkuca *return* uz povratne vrednosti, funkcija vraća *None*
- Funkcija mora biti definisana pre poziva funkcije

```
1 def saberi(a,b):
2     return a+b
3
4 print(saberi(3,6)) #9
5
6 rez = oduzmi(34,4) #NameError: name 'oduzmi' is not defined
7
8 def oduzmi(a,b):
9     return a-b
```

Prenos argumenata

- Poziv funkcije se vrši tako što joj se proslede odgovarajući argumenti
- Argumenti se prenose po *referenci*
- Kod promenljivih tipova podataka svaka promena unutar funkcije, prosleđenih argumenata, odraziće se vrednosti izvan funkcije

```
1 def funkcija(a):
2     a[2]='c'
3
4 lista = ['a', 'b', 3]
5
6 print(lista) #['a', 'b', 3]
7 funkcija(lista)
8 print(lista) #['a', 'b', 'c']
```

- Ovo ne dolazi do izražaja kada je reč o promenljivim tipovima podataka

Podrazumevani parametri

- Ako se u definiciji funkcije definišu podrazumevani parametri, onda u pozivu funkcije nije neophodno navoditi argumente na tim mestima

```
1 def saberi(a, b=7):
2     return a+b
3
4 print(saberi(3)) #10
5 print(saberi(3,2)) #5
```

- Svi parametri nakon prvog koji ima podrazumevanu vrednost, moraju imati podrazumevanu vrednost

```
1 def saberi(a, b=7,c):
2     return a+b+c
3
4 saberi(3) #SyntaxError: non-default argument follows
      default argument
```

Prazna funkcija i redosled parametara

```
1 def saberi(a, b):
2
3 print(saberi(3,2)) #IndentationError: expected an
      indented block after function definition
```

- Rešenje je korišćenje naredbe pass
- Takođe, parametrima se može obrnuti redosled gde se prilikom prosleđivanja argumenata upisuju imena parametara

```
1 def saberi(a, b):
2     pass
3
4 saberi(b=3,a=2)
```

Moduli

- Predstavljaju .py fajlove sa pridodatim funkcijama i klasama
- Dekompozicija kako bi se povećala preglednost koda
- Elementi modula se mogu učitati korišćenjem naredbe *import*

```
1 import math
```

- Takođe, moguće je koristiti naredbu *from* uz naredbu *import* za učitavanje modula u celosti

```
1 from math import *
```

- Moguće je učitati jednu ili više funkcionalnosti iz postojećih modula

```
1 from math import sqrt, sin
```

Pristup elementima modula

- Pristup elementima zavisi od načina kako su oni učitani

```
1 import math
2 print(math.sqrt(2))
```

- Ako je naziv modula predug moguće je definisati alternativno ime

```
1 import math as m
2 print(m.sqrt(2))
```

```
1 from math import *
2 print(sqrt(2))
```

```
1 from math import sqrt
2 print(sqrt(2))
```

Opseg vidljivosti promenljivih

- built-in: predstavlja ugrađeni prostor, odnosno promenljive i literale koji su definisani na nivou jezika (list(), True, None…)
- globalni odnosno na nivou modula: predstavlja sve promenljive definisane unutar modula definisane izvan klasa i funkcija, dostupne u okviru samo tog modula
- lokalni nivo: predstavlja sve promenljive definisane unutar funkcije i vidljive su samo unutar nje
- Ime se uvek traži u lokalnom nivou, ako tu ne postoji traži se dalje na globalnom, pa na built-in nivou

- Zato je moguć pristup elementima koji se nalaze izvan funkcije

```
1 def zbir(a,b):
2     print(hex(id(c)))
3     return a+b+c
4
5 c = 3
6 print(zbir(1,2)) #6
7 print(hex(id(c))) #ista adresa kao i u funkciji
```

- Šta se događa ako se želi modifikovati vrednost promenljive u funkciji?

```
1 def zbir(a,b):
2     c=4
3     print(hex(id(c)))
4     return a+b+c
5
6 c = 3
7 print(zbir(1,2)) #6
8 print(hex(id(c))) #adresa nije ista kao i u funkciji
```

Opseg važenja promenljivih

- Ako se želi modifikovati neka promenljiva izvan lokalnog nivoa neophodno ju je proglasiti za globalnu
- Globalna promenljiva se dobija korišćenjem ključne reči global

```
1  def zbir(a,b):
2      global c
3      c=4
4      print(hex(id(c)))
5      return a+b+c
6
7  c = 3
8
9  print(zbir(1,2)) #7
10 print(hex(id(c))) #adresa je ista kao i u funkciji
```

Hvala na pažnji!