Uvod u programski jezik Pajton - 3. deo

Uvod u merno-informacione sisteme

Rečnici

- Rečnici predtavljaju kolekciju podataka gde se podaci čuvaju u obliku ključ-vrednost
- Rečnici:
  - su promenljivi (elementi se mogu menjati)
  - ne mogu sadržati duplikate
- Podaci se čuvaju pomoću heš mapa tj. heš tabela
- Nemaju fiksnu dužinu
- Umesto preko indeksa, pristup elementima se vrši preko ključa

- Rečnici se navode u vitičastim zagradama {}
- Svi parovi se navode odvojeni zarezom
- Ključ i vrednost se odvajaju dvotačkom

```
1 recnik = {'a':1, 'b':2}
```

- Pristup elementima vrši se kao i kod liste, osim što se umesto indeksa navodi ključ
- U slučaju da ne postoji element sa traženim ključem dobija se greška *KeyError*

```
1 recnik = {'a':1, 'b':2}
2 print(recnik['a'])#1
3 print(recnik['c'])#KeyError
```

- Korišćenjem metode *get* moguće je izbeći generisanje greške *KeyError*
- Kao drugi parametar metodi *get*, može se proslediti podrazumevana povratna vrednost ako se element ne pronađe

```
recnik = {'a':1, 'b':2}
print(recnik.get('c'))#None
print(recnik.get(1,'Nema elementa'))#Nema elementa
```

- Dodavanje elemenata u rečnik

```
recnik = {1:'test', 2:'tekst'}
recnik[17]='proba'
print(recnik)#{1: 'test', 2: 'tekst', 17: 'proba'}
```

- Od verzije Pajtona 3.7 rečnici su uređena vrsta podataka (kako se dodaju podaci u rečnik tako će biti i ispisivani, ovo ne važi za starije verzije)

- Rečnik mora imati **jedinstven** ključ
- Ako se dodaju ključu koji već postoji dodeli nova vrednost, veza sa starom se prekida

```
1 recnik = {'a':1, 'b':2, 'b':3}
2 print(recnik)#{'a': 1, 'b': 3}
```

- Ključ se ne može menjati, pa zbog toga mora biti nepromenljivi tip (broj, string, torka (ne sme sadržati promenljive tipove)) u suprotnom dobija se greška *TypeError: unhashable type*

```
1 recnik = {'a':1, 3:2, (3,2,'string'):3}
2 print(recnik)#{'a': 1, 3: 2, (3, 2, 'string'): 3}
3 recnik[[1,2]]=5#TypeError: unhashable type: 'list'
```

- Vrednosti rečnika ne moraju biti jedinstvene

```
1 recnik = {'a':1, 'b':1, 'c':1}
2 print(recnik)#{'a': 1, 'b': 1, 'c': 1}
```

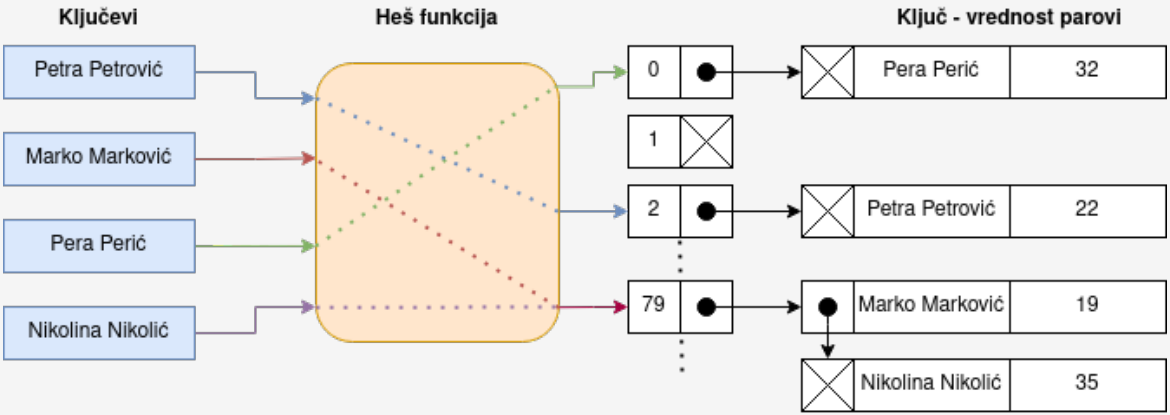
- Kao vrednost može se čuvati bilo koji tip

```
1 recnik = {1:1, 2:'dva', 3:[1,'dva'], 4:{1:1,2:'dva'}}
2 print(recnik)#{1:1,2:'dva',3:[1,'dva'],4:{1:1,2:'dva'}}
```

Poređenje liste i rečnika

- Kako bi se skladištili ili očitali podaci neophodno im je pristupiti
- Lista pristupa podacima po indeksu, gde se za određeni indeks pristupa elementu
- Rečnici podacima pristupaju tako što se vrednost ključa konvertuje u broj pomoću heš funkcije
- U slučaju kolizije (kao izlaz heš funkcije dobija se ista vrednost za dva različita ključa) podaci se međusobno uvezuju
- Na osnovu generisanog broja pristupa se odgovarajućem elementu u memoriji

Postupak čuvanja rečnika



Poređenje liste i rečnika

- Sa stanovišta pristupa odgovarajućem elementu ne postoji osetna razlika između liste i rečnika
- Osetne razlike se dobijaju prilikom pretrage
- Pretraga liste vrši se prolaskom kroz sve elemente liste
- Ako lista ima n elemenata, u najgorem slučaju neophodno je kroz sve elemente (n pristupa)
- Kada se vrši pretraga rečnika, ključ se prosleđuje heš funkciji i pristupa se jasno definisanoj lokaciji

```
1  from timeit import default_timer
2
3  lista = []
4  recnik = {}
5
6  for i in range(10_000_000):
7      lista.append(i)
8      recnik[i]=i*3
9
10
11 def pretraga(kolekcija, rec):
12     if rec in kolekcija:
13         return True
14     else:
15         return False
```

Ime metode	Opis
clear()	Briše sve elemente iz rečnika.
copy()	Vraća kopiju rečnika.
fromkeys(key, value)	Pravi rečnik za prosleđene ključeve i dodeljuje im vrednost
get(value, string)	Vrši pretragu rečnika na osnovu vrednosti. Ako ne postoji vrednost u rečniku vraća string.
items()	Vraća listu, gde se svaki par ključ vrednost stavlja u torku.
keys()	Vraća listu ključeva iz rečnik.
pop(key)	Briše element sa prosleđenim ključem.
popitem()	Briše poslednji uneti element u rečnik.
setdefault(key, value)	Vraća vrednost za prosleđeni ključ. Ako ključ ne postoji onda ga dodaje sa prosleđenom vrednošću.
update({key:value})	Dodaje u rečnik odgovarajući ključ-vrednost par.
values()	Vraća listu svih vrednosti iz rečnika.

```
recnik = {'a':1, 2:3, 4:[1,2,5]}

recnik_2 = recnik.copy()
print(recnik_2)#{'a': 1, 2: 3, 4: [1, 2, 5]}
recnik_2.clear()
print(recnik_2)#{}
recnik_2 = dict.fromkeys([1,'b',3],0)
print(recnik_2)#{1: 0, 'b': 0, 3: 0}
print(recnik.get(2))#3
print(recnik.get(3))#None
print(recnik.items())#dict_items([('a', 1), (2, 3), (4, [1,
    2, 5])])
print(recnik.keys())#dict_keys(['a', 2, 4])
```

```
recnik.pop('a')
print(recnik)#{2: 3, 4: [1, 2, 5]}
recnik.popitem()
print(recnik)#{2: 3}
recnik.setdefault('a',4)
print(recnik)#{2: 3, 'a': 4}
recnik.setdefault(2,1)
print(recnik)#{2: 3, 'a': 4}
recnik.update({1:1})
print(recnik)#{2: 3, 'a': 4, 1: 1}
print(recnik.values())#dict_values([3, 4, 1])
```

Skupovi

- Predstavlja skup jedinstvenih podataka
- Elementi se mogu brisati i dodavati u skup
- Elementi skupova su nepromenljivi tipovi
- Skupovi nisu indeksirani
- Skupovi nisu uređeni (ne možemo znati u kom rasporedu se čuvaju i prikazuju)
- Elementima skupa se ne može pristupiti ni preko indeksa ni preko ključa
- Skupovi ne mogu sadržati duplikate (svi duplikati se automatski brišu)

Definicija i pristup elementima skupa

- Skup se definiše tako što se elementi navode u {}, odvojeni zarezom
- Ako se želi definisati prazan skup onda je neophodno pozvati konstruktor *set()*, bez prosleđenih parametara

```
1 skup = set()#set()
2 skup = {1,2,3}#{1, 2, 3}
3 skup = set('popokatepetl')#{'e','o','k','p','t','l','a'}
4 skup = set([1,'dva',3])#{3, 1, 'dva'}
```

- Elementima skupa je moguće pristupiti samo iteracijom kroz *for* petlju

```
1 skup={1, 2, 3}
2 for i in skup:
3     print(i)
```

Ime metode	Opis
add(el)	Dodaje element u skup
clear()	Briše sve elemente skupa
copy()	Kopira skup u celosti
difference(skup)	Vraća razliku sa prosleđenim skupom
difference_update(skup)	Briše elemente koji se nalaze u oba skupa
discard(el)	Briše prosleđeni element iz skupa
intersection(skup)	Vraća presek sa prosleđenim skupom
intersection_update(skup)	Briše elemente koji se ne nalaze u oba skupa

```
skup={1,2,3,4,5}
skup_2={3,4,5,6}
skup_2.add(7)
print(skup_2)#{3, 4, 5, 6, 7}
skup_3=skup_2.copy()
skup_3.clear()
print(skup_3)#set()
print(skup_2.difference(skup))#{6, 7}
skup_2.difference_update(skup)
print(skup_2)#{6, 7}
skup.discard(2)
print(skup)#{1, 3, 4, 5}
skup_2={3,4,5,6,7}
print(skup_2.intersection(skup))#{3, 4, 5}
print(skup_2)#{3, 4, 5, 6, 7}
skup_2.intersection_update(skup)
print(skup_2)#{3, 4, 5}
```

Ime metode	Opis
isdisjoint(skup)	Vraća true ako postoji presek između skupova
issubset(skup)	Vraća true ako je skup podskup prosleđenog skupa
issuperset(skup)	Vraća true ako je skup nadskup prosleđenog skupa
pop()	Izbacuje element iz skupa (na slučajan način)
remove(el)	Briše prosleđeni element iz skupa
symmetric_difference(skup)	Elementi koji se nalaze u oba skupa ali ne i u preseku
symmetric_difference_update(s)	Vraća simetričnu razliku skupova
union(skup)	Određuje uniju skupova
update(skup)	Proširuje postojeći skup prosleđenim

```
skup={1,2,3,4,5}
skup_2={2,3,4,5,7}
print(skup.isdisjoint(skup_2))#False
print(skup.issubset(skup_2))#False
skup_2.remove(7)
print(skup.issuperset(skup_2))#True
skup.pop()
skup.remove(2)
print(skup)#{3, 4, 5}
print(skup.symmetric_difference(skup_2))#{2}
skup.symmetric_difference_update(skup_2)
print(skup)#{2}
print(skup.union(skup_2))#{2, 3, 4, 5}
skup.update(skup_2)
print(skup)#{2, 3, 4, 5}
```

Rad sa datotekama

- Pajton može da radi i sa binarnim i sa tekstualnim fajlovima
- Ako se želi otvoriti datoteku neophodno je koristiti metodu *open* kao i specificirati mod u kom se želi datoteka otvoriti

```
1  file = open('dokument.txt','r')
```

- Modovi u koji se mogu koristiti za rukovanje fajlom su:
  - 'r' - čitanje iz datoteke, ako datoteka ne postoji vraća grešku.
  - 'w' - pisanje u datoteka, pravi datoteka ako ne postoji.
  - 'a' - dodavanje na kraj datoteke, pravi datoteka ako ne postoji.
  - 'x' - pravi novi datoteka, ako već postoji vraća grešku.

- Čitanje iz fajla se vrši *read* metodom

```
1 f = open("file.txt", "r")
2 print(f.read())
```

- Prosleđivanjem parametra metodi *read* moguće je ograničiti koliko karaktera se želi učitati
- Za čitanje jedne linije teksta moguće je koristiti metodu *readline*

```
1 f = open("file.txt", "r")
2 print(f.readline())
```

- Za čitanje više linije teksta moguće je koristiti metodu *readlines*

- Upis u fajl se vrši *write* metodom
- Po završetku pisanja u datoteku neophodno je i zatvoriti datoteku pozivom metode *close*

```
1 f = open("file.txt", "w")
2 f.write('Tekst')
3 f.close()
```

- Ako se želi upisati više linija moguće pozivom metode *writelines*
- Moguće je istovremeno i čitati i pisati ali se tada uz mod mora dodati i znak +

```
1 f = open("file.txt", "r+")
2 print(f.readline())
3 f.write('Tekst\n')
4 f.close()
```

- Zgodan način za rad sa fajlovima pruža naredba *with*

```
1 with open("file.txt", "w") as f:
2     f.write('Tekst')
```

- Prilikom korišćenja *with* naredbe nije neophodno zatvarati datoteku, jer će to odraditi po završetku bloka naredbi unutar *with* naredbe

- Korišćenjem metode *tell* moguće je odrediti trenutnu poziciju kursora u datoteci

```
1 with open("file.txt", "w") as f:
2     f.write('Tekst')
3     print(f.tell())#5
```

- Pomoću metode *seek* moguće je promeniti poziciju kursora

```
1 with open("file.txt", "w+") as f:
2     f.write('Tekst')
3     print(f.tell())#5
4     f.seek(0)
5     print(f.tell())#0
6     print(f.readline())#Tekst
```

Hvala na pažnji!