Skupovi i rečnici

Skupovi i rečnici se koriste kada nam originalni redosled podataka na ulazu u program nije bitan, ali nam je potrebno da sve vrednosti unutar ovih tipova podataka budu različite.

Skupovi (sets)

U Pythonu 2 nije postojao specifičan način da se set kreira, nego su se setovi kreirali prosleđivanjem lista i stringova. U kasnijim verzijama je to promenjeno i uspostavljena je sintaksa inicijalizacije setova pomoću vitičastih zagrada. Kako su setovi takođe kolekcije, imaju nekih sličnosti sa listama u smislu da podržavaju iteraciju i dodavanje/brisanje elemenata. Setovi se mogu posmatrati i kao „ključevi" rečnika bez vrednosti, jer im je implementacija, kada se posmatra izvorni kod, veoma slična rečnicima. Takođe, redosled u setovima je proizvoljan, što znači da se ne možemo osloniti na to da će prosleđena lista (ili druga sekvenca) zadržati isti redosled.

Pošto su setovi neuređeni, a ni ne mapiraju „ključeve" na vrednosti, oni nisu ni sekvence ni mapirajući tipovi, već kategorija za sebe. Priroda setova je više u domenu matematike, pa tako setovi, tj. skupovi podržavaju uniju, presek, razliku i slično.

Elementi setova mogu biti samo nepromenjivi tipovi kao što su stringovi, celi brojevi, n-torke, pa čak i moduli. Liste i rečnici nisu dozvoljeni kao elementi setova i ako do toga dođe, nastaće `TypeError: unhashable type` izuzetak.

Ugnežđeni setovi nisu mogući, ali ako postoji potreba za tom funkcionalnošću, treba koristiti ugrađeni `frozenset` tip, koji se može koristiti kao element podatka tipa set.

Sintaksa

Setovi se mogu kreirati na dva načina:

- korišćenjem ugrađene funkcije `set([iterabilni objekat])` – na ovaj način možemo pretvoriti već postojeći iterabilni objekat u set ili kreirati prazan set;
- korišćenjem vitičastih zagrada, na sledeći način: `s = {1,2,3}` – napomena ovde je da se set ne može inicijalizovati kao `s = {}` (prazan set), jer je to sintaksa koja važi za rečnike.

Već pomenuta matematička svojstva setova možemo prikazati na sledeći način:

Primer kreiranja setova i korišćenja unije, preseka i razlike nad njima:

Radno okruženje

```
set_a = set(['a', 'b', 5, 2])
print("Set_a: {}".format(set_a))
```

```
set_b = {'a', 'c', 5, 4, None}
print("Set_b: {}".format(set_b))


print("Intersection: {}".format(set_a & set_b))

print("Union: {}".format(set_a | set_b))

print("Difference: {}".format(set_a - set_b))
```

U našem primeru smo prvo definisali dva seta. Prvi koristeći ugrađenu funkciju `set()` sa prosleđenom listom vrednosti, a drugi koristeći vitičaste zagrade. Nakon ispisa ova dva seta, možemo videti da se redosled promenio. Ako bismo ponovo pokrenuli program, dobili bismo drugačiji redosled. Važno je napomenuti da redosled nije fiksan već proizvoljan, a tačan algoritam za određivanje redosleda u setovima se može videti u njihovom izvornom kodu.

Na ovom primeru smo uvideli i kako se setovi ponašaju prilikom različitih operacija kao što su presek, unija, razlika i ostali koristeći aritmetičke i binarne operatore.

Menjanje setova

Iako ugnežđeni setovi jesu nepromenljivi (`immutable`), setovi generalno mogu da se menjaju, ali drugačije nego liste. Pojedinačne elemente u setovima ne možemo menjati, ali možemo dodavati nove i brisati stare. Novi elementi se dodaju metodom `.add()`, gde za argument prosleđujemo element koji želimo. Ako želimo odjednom da dodamo više elemenata, koristimo metodu `.update()`.

Za brisanje elemenata imamo više opcija:

- `remove(elem)` – ovoj metodi prosleđujemo element koji želimo da obrišemo; ako prosleđeni element ne postoji u setu, dobićemo grešku;
- `discard(elem)` – ovoj metodi takođe prosleđujemo element koji želimo da obrišemo, ali je razlika u tome da nećemo dobiti grešku ako prosleđeni element ne postoji;
- `pop()` – metoda bez argumenata koja će uvek izbaciti poslednji element; kako su setovi neuređeni, ako koristimo `pop()` metodu, moramo obratiti pažnju na to koji element se izbacuje;
- `clear()` – još jedna metoda bez argumenata; ovom metodom brišemo čitav sadržaj seta, ali nam set kao promenljiva ostaje u memoriji;
- `del` – ključna reč `del` nam omogućava brisanje promenljive iz programa.

Primer
Kod

`set_a = set(['a', 'b', 5, 2])`
 	
`print("Starting set: {}".format(set_a))`
 	
`set_a.add(None) # Dodavanje elementa None`

```
print("After usage of .add() method: {}".format(set_a))

set_a.remove(5) # Brisanje elementa koji ima vrednost 5

print("After usage of .remove() method: {}".format(set_a))


set_a.clear() # Uklanjanje svih elemenata iz seta

print(set_a) # Dobijamo prazan set
```

Rezultat

```
Starting set: {2, 'a', 5, 'b'}
After usage of .add() method: {2, 'a', 5, None, 'b'}
After usage of .remove() method: {2, 'a', None, 'b'}
set()
```

Tabela 14.1. Primer izmene, preseka i brisanja setova

Upotreba setova

Iako su setovi praktični za matematičke operacije, pomažu nam i u konvencionalnom programiranju. Kako oni ne dozvoljavaju dupliranje istih vrednosti, možemo ih koristiti za filtriranje listi i drugih promenljivih tipova podataka i nakon toga vratiti taj set u tip liste pozivom `list(set())`.

Kako liste možemo pretvoriti u setove, to nam omogućava da izvršimo uniju, presek, razliku i ostale matematičke operacije koje smo pomenuli upravo nad tim listama.

Primer odstranjivanja istih vrednosti iz liste:

Recimo da imamo listu godišta studenata: lst=[1993, 1994, 1995, 1994, 1993, 1996] i želimo da tu listu modifikujemo tako da sadrži samo jedinstvene elemente. Prvo ćemo listu pretvoriti u set naredbom set(lst), a potom taj set pretvoriti nazad u listu i sortirati je uzlazno:

```
lst=[1993, 1994, 1995, 1994, 1993, 1996]
set_lst = set(lst)
print(set_lst)

lst = list(set_lst)
print(lst)

lst.sort()
print(lst)
```

Pitanje

Setovi su:

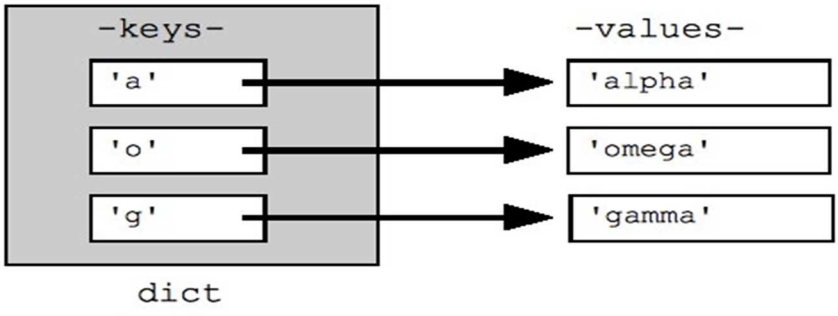
- **promenljivi tipovi podataka**
- uređeni tipovi podataka
- nepromenljivi tipovi podataka

Objašnjenje:

Setovi su promenljivi tipovi podataka. Tačnije, elementi se mogu dodavati ili brisati, ali jednom dodat element ne može biti promenjen, već samo obrisan.

Rečnici

Rečnici nisu deo sekvenci, već poseban tip podataka koji služi za mapiranje, ali to ne znači da rečnici ne mogu biti kolekcije koje sadrže druge tipove podataka. Python rečnici se još nazivaju i asocijativni nizovi ili heš tabele (`hash tables`). Tako možemo reći da je rečnik kontejnerski objekat koji se sastoji od ključ-vrednost elemenata, gde su ključevi jedinstvene vrednosti. Takođe, rečnici, kao ni setovi, ne podržavaju indeksiranje. Jedan primer strukture rečnika vidimo na slici 14.1.



Slika 14.1. Prikaz strukture rečnika[1]

Na ovoj ilustraciji uočavamo ključ-vrednost parove. Pa tako, na primer, vrednost po ključu a je alpha. Ovakav rečnik bi u Pythonu mogao biti implementiran na sledeći način:

```
d = {'a':'alpha', 'o':'omega', 'g':'gamma'}.
```

Sintaksa i kreiranje rečnika

Rečnike možemo kreirati na tri načina, što možemo videti na primeru:

Primer sintakse i kreiranja rečnika:

Radno okruženje

```
student_dict = {'Name':'Marco', 'Field':'Electrical engineering',
'Age':'20'}

values_dict = dict(first=1, second = 2)

cities_dict = {}
cities_dict['city1'] = 'New York'
```

[1] https://miro.medium.com/max/1038/1*oNuaSOlSozUchdxiOE1ieQ.jpeg

```
cities_dict['city2'] = 'London'
cities_dict['city3'] = 'Moscow'

print(student_dict)
print(values_dict)
print(cities_dict)
```

Sintaksa kreiranja rečnika predstavlja ključ-vrednost elemente odvojene zarezom koji su obuhvaćeni vitičastim zagradama. Ključ-vrednost parovi su elementi rečnika i oni su odvojeni dvotačkom.

Prvi način kreiranja(rečnik `'student_dict'`) je korišćenje eksplicitne notacije uz pomoć vitičaste zagrade unutar vitičastih zagrada. Elementi (ključ-vrednost parovi) su odvojeni zarezom. U slučaju prvog elementa rečnika `student_dict`, ključ je string `Name`, a vrednost tog ključa je string `Marco`.

Takođe, rečnici se mogu kreirati korišćenjem ugrađene `dict()` (rečnik values_dict) funkcije kojoj se prosleđuju imenovani argumenti koji postaju ključevi novog rečnika, a vrednosti tih argumenata su zapravo i vrednosti tih ključeva, pa je tako na primeru rečnika `values_dict` kreiranog pomoću `dict()` funkcije ključ prvog elementa zapravo imenovani argument `first`, a vrednost ključa prvog elementa broj `1`.

U trećem primeru (rečnik `cities_dict`), prvo je kreiran prazan rečnik koristeći notaciju `{}`. Ovde vidimo povezanost, ali i razliku između rečnika i setova. Iako za setove koristimo vitičaste zagrade, ako promenljivoj prosledimo `{}`, dobićemo rečnik, a ako joj prosledimo `{1,2,3}` – dobićemo set.

Rad sa rečnicima

Neka nam u ovom delu nastavne jedinice za primer služi prethodno definisani rečnik:

```
student_dict = {'Name':'Marco', 'Field':'Electrical engineering',
'Age':'20'}
```

Kako je rečnik promenljivi tip podatka, možemo mu menjati elemente, i to preciziranjem ključa i operatorom dodele:

```
student_dict['Age'] = 25
```

Ako prosledimo ključ koji ne postoji, dobićemo `KeyError`. Da bismo rešili ovaj problem, imamo metodu `.get(kljuc, [podrazumevana vrednost])` koja će nam ili dobaviti vrednost nad zadatim ključem ili vratiti zadatu podrazumevanu vrednost i tako izbeći `KeyError` grešku.

Primer	
Kod	**Ispis**
`print(student_dict['Gender'] )`	`KeyError`
`print(student_dict.get('Gender', 'Field doesnt exist') )`	`'Field doesnt exist'`

Tabela 14.2. Primer korišćenja drugog parametra get metode

Ako imamo listu (ili n-torku) vrednosti od kojih želimo da napravimo rečnik, možemo upotrebiti klasnu metodu `.fromkeys(seq, [podrazumevana_vrednost])` u kojoj zadajemo sekvencu i opcioni parametar čiju će vrednost imati svi ključevi. Ako se ne zada, podrazumevana vrednost je `None`. Jedan takav primer bi mogao da izgleda ovako:

Primer	
Kod	`recnik = {}.fromkeys(['name', 'last name', 'address', 'city'], None)` `print(recnik)`
Ispis	`{'name': None, 'last name': None, 'address': None, 'city': None}`

Tabela 14.3. Primer korišćenja podrazumevane vrednosti None

U ovom primeru, metodi `.fromkeys()` smo prosledili listu od koje želimo da generišemo rečnik, kao i vrednost None koja će postati zajednička vrednost za sve ključeve.

Brisanje elemenata u rečniku se vrši na dva načina. Prvi je `del` ključna reč, a drugi način podrazumeva da se koristi `.pop()`. Razlika između ova dva načina je u tome što `del` ključna reč ne vraća ništa, dok `.pop()` metoda vraća vrednost obrisanog ključ-vrednost para.

Iteracija kroz rečnik

Pošto su rečnici kolekcije, oni podržavaju iteraciju. Kada je reč o iteraciji rečnika, potrebno je znati da li želimo iterirati kroz ključ-vrednost parove (elemente rečnika), samo kroz ključeve ili samo kroz vrednosti.

Primer
Kod
`for i in student_dict:` `  print(i, end= ', ')`
`for i in student_dict.values():` `  print(i, end= ', ')`
`for key, value in student_dict.items():` `  print('{}:{}'.format(key, value), end = ', ')`
Ispis
`Name, Field, Age,`
`Marco, Electrical Engineering, 20,`
`Name:Marco, Field:Electrical Engineering, Age:20,`

Tabela 14.4. Primer iteracije kroz rečnik

U ovim primerima smo uveli dve nove metode i to `.values()` i `.items()`. Prva metoda nad rečnikom vraća kolekciju samo vrednosti iz ključ-vrednost parova, dok nam druga metoda, `.items()`, vraća uređenu n-torku sa dva elementa, gde je prvi element ključ, a drugi element n-torke vrednost na koju ključ pokazuje. Takođe postoji i metoda `.keys()`, koja funkcioniše identično kao i `.values()` metoda i vraća samo ključeve.

Sortiranje

Rečnici ne podržavaju sortiranje jer su neodređeni, a redosled elemenata u njima je proizvoljan. Ako nam je potrebno da sam rečnik održi redosled, možemo koristiti poseban tip podatka `OrderedDict` iz modula `'collections'`, a ako nam je samo trenutno potrebno da imamo sortirane elemente, možemo to postići pomoću lista i iteracija:

Sortiranje smo ovde izvršili u tri koraka. Prvo smo izvukli ključeve iz rečnika i smestili ih u listu. Nakon toga je bilo potrebno sortirati vrednosti u toj listi. Treći korak je da koristimo vrednosti te sortirane liste za pristup vrednostima koje se nalaze na tim ključevima u rečniku.

Primer sortiranja elemenata pomoću lista i iteracija:

Radno okruženje

```
student_dict = {'Name':'Marco', 'Field':'Electrical engineering',
'Age':'20'}

student_dict_keys = list(student_dict.keys())

student_dict_keys.sort()

print("Order of dict keys: {}".format(student_dict.keys()))

print("Order of dict keys in a sorted list: {}".format(student_dict_keys))


for i in student_dict_keys:
    print("Dict element: {}, on a key: {}".format(student_dict[i], i))
```

Rezime

- Ni skupovi ni rečnici ne zadržavaju originalni redosled podataka.
- Elementi setova ne mogu biti promenljivi tipovi podataka kao što su liste i rečnici.
- Setovi podržavaju matematičke metode kao što su: unija, presek i razlika.
- Setovi ne mogu sadržati iste elemente i ako prosledimo sekvencu koja između ostalog ima i duple vrednosti, jedna od te dve vrednosti će se odstraniti.
- Setovi se mogu kreirati ugrađenom funkcijom set() ili korišćenjem vitičastih zagrada ({}). Prazan set se ne može definisati vitičastim zagradama, jer je to način definisanja rečnika.
- Setovi ne podržavaju indeksiranje.
- Rečnici nisu deo sekvenci, već poseban tip podatka koji služi za mapiranje. Baziraju se na ključ:element parovima odvojenim zarezom.
- Rečnici ne podržavaju indeksiranje.
- Rečnici se mogu definisati pomoću ugrađene funkcije dict() ili pomoću vitičastih zagrada ({}).
- U rečniku ne mogu postojati dva ista ključa.
- Samo ključeve u rečniku dobavljamo metodom keys(), a samo vrednosti datog rečnika dobavljamo metodom values().