Programska potpora komunikacijskim sustavima

Izv. prof. dr. sc. Marin Šilić

Programski jezik Python – Tipovi Podataka

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Sadržaj predavanja

- Liste
- N-torke
- Skupovi
- Mape

Liste

Liste

- Lista – apstraktni tip podatka koji podrazumijeva uređenu kolekciju podataka
- Razlika između liste i polja
  - Broj elemenata liste je promjenjiv (nema fiksnu duljinu)
  - Dodavanje i brisanje elemenata liste tijekom izvođenja
- Elementi liste mogu biti različitog tipa
- Liste su slične ArrayList razredu u Javi i C#-u

Liste

- Stvaranje liste

```
lista = [0, 1, 2, 3, 4]
lista = list()
```

- Dohvaćanje elemenata liste
  - Dohvaćanje jednog elementa

```
lista[indeks]
```

Liste

- Negativni indeksi

```
lista[indeks]
```

- Prvi element liste ima indekse
  - `0`
  - `-(duljina)`
- Zadnji element liste ima indekse
  - `-1`
  - `(duljina-1)`

Liste

- Dohvaćanje elemenata liste
  - Dohvaćanje raspona elementa
  - Kao rezultat vraća novu listu

```
raspon = lista[indeks_pocetka: indeks_kraja]
```

- Vraća elemente od `indeks_pocetka` (uključujući) do `indeks_kraja` (isključujući)
  - Ako se `indeks_početka` izostavi – `indeks_početka` je 0
  - Ako se `indeks_kraja` izostavi – `indeks_kraja` je *duljina*
  - Oba indeksa se mogu izostaviti – vraća kopiju cijelog polja

Liste

- Zadatak
  - Definirati novu listu sa elementima različitog tipa
  - Dohvatiti elemente liste sa pozitivnim i negativnim indeksima
  - Dohvatiti raspon elemenata

Liste

- Dodavanje elemenata u listu
  - Konkatenacija više lista
  - Stvara novu listu koja sadrži redom elemente više lista

```
nova_lista = lista1 + lista2 + lista3
```

- Metoda `append`
  - Dodaje jedan element na kraj liste (ne stvara novu listu)

```
nova_lista.append(100)
```

- Metoda `extend`
  - Dodaje elemente liste na kraj druge liste (ne stvara novu listu)

```
nova_lista.extend([200, 300])
```

- Metoda `insert`
  - Ubacuje element na određenu poziciju u listi (ne stvara novu listu)

```
nova_lista.insert(1, 543)
```

Liste

- Zadatak
  - Napraviti praznu listu
  - Dodati elemente pomoću metoda `append`, `extend` i `insert`
  - Konkatenirati listu sa nekom drugom listom
  - Sličnosti/razlike `extend` i `+`

```
l1 = [1, 2, 3]
l2 = [10, 20, 30]
l1 + l2
l1.extend(l2)
```

Liste

- Broj elemenata liste
  - Ugrađena funkcija `len`, nije metoda objekta!
  - `len([1,3,5,7,7])`
- Broj pojavljivanja elementa u listi
  - Metoda `count`
  - `[1,3,5,7,7].count(7)`
- Ispitivanje prisutnosti elementa u listi
  - Operator `in`
  - `4 in [1,3,5,7,7]`
- Indeks prvog pojavljivanja elementa u listi
  - Metoda `index`
  - `[1,3,5,7,7].index(7)`

Liste

- Uklanjanje elemenata iz liste
  - Prema indeksu – operator `del` (nije metoda objekta)
    - Uklanja n-ti element liste, ne stvara novu listu

```
del lista[3]
```

  - Prema vrijednosti elementa – `remove` metoda
    - Uklanja prvo pojavljivanje elementa u listi, ne stvara novu listu

```
[1,3,5,7,7].remove(5)
```

  - Prema indeksu ili prvi element s kraja liste – metoda `pop`
    - Uklanja zadnji ili n-ti element, ne stvara novu listu

```
[1,3,5,7,7].pop()
[1,3,5,7,7].pop(2)
```

Liste

- Zadatak
  - Napisati funkciju koja vraća listu prvih 10 brojeva koristeći petlju while i funkciju `append`

```
i=0
numbers = []
while i<10:
    ...
```

  - Napisati funkciju `rem(x, l)` koja briše element s vrijednosti `x` iz liste (koristiti `index` + `del`)
    - `def rem(x, l):` ...
    - Funkcija vraća promijenjenu listu
    - Provjeriti tipove argumenata `(x → int, l → list)`!

Liste

- Plitko i duboko kopiranje
  - Pridruživanje liste

```
l1 = [0, 1, 2]
l2 = l1
```

  - Obje varijable pokazuju na istu listu (istu mem. adresu)

```
print id(l1), id(l2)
```

  - Promjena se vidi u obje varijable

```
l1[0] = 10
print l2
```

Liste

- Plitko i duboko kopiranje
  - Kopiranje liste (plitko)

```
l1 = [0, 1, 2]
l2 = l1[:]
```

  - Sada imena pokazuju na različite liste

```
l1[0] = 10
print l1, l2
    [10, 1, 2], [0, 1, 2]
```

  - Napomena: ako lista sadrži podliste onda je potrebno duboko kopiranje

```
import copy
copy.deepcopy(x)
```

Liste

- Kopiranje na raspon
  - Kopiranje na parcijalni raspon

```
l = [0,1,2,3,4,5]
l[1:5] = [10]
print l
[0, 10, 5]
```

  - Kopiranje na cijeli raspon (mijenjanje in-place)

```
l[:] = [100]
print l
[100]
```

N-torke

N-torke

- Nepromjenjiva lista
  - Brže izvođenje operacija nego kod listi
  - Nema uklanjanja, dodavanja ili zamjene elemenata
  - Ne podržava metode `append`, `extend`, `insert`, `remove`, `pop` i `del`
- Podržava ostale metode kao i nad listama
  - Jednak način dohvaćanja elemenata
  - Podržava konkatenaciju (jer stvara novu n-torku)
  - Podržava `index`, `count`, `len` i `in`

```
ntorka = ('prvi', 'drugi', 3, 4)
```

N-torke

- Konverzija između n-torke i liste
  - Pomoću razreda `list` i `tuple`
  - Stvaranje novog objekta razreda uz predaju postojeće liste/ntorke

```
ntorka = tuple([1,2,3])
lista = list(ntorka)
```

N-torke

- Pridruživanje više vrijednosti pomoću n-torki

```
(x, y, z) = (1,3,5)
```

- Varijable n-torke s lijeve strane pridruživanja poprimaju vrijednosti elemenata n-torke s desne strane
  - Korisno za vraćanje više vrijednosti iz funkcija!

N-torke

- Pridruživanje više vrijednosti pomoću n-torki

```
(x, y, z) = (1,3,5)
```

- Varijable n-torke s lijeve strane pridruživanja poprimaju vrijednosti elemenata n-torke s desne strane
  - Korisno za vraćanje više vrijednosti iz funkcija!

N-torke

- Zadatak
- Napisati funkciju `head_tail(l, n)` koja vraća prvih `n` i krajnjih `n` elemenata liste

```
def head_tail(l, n):
    ...
```

Skupovi

Skupovi

- Neuređeni skupovi jedinstvenih elemenata
  - Nema duplikata!
  - Elementi skupa mogu biti različitog tipa
  - Moguće mijenjati sadržaj skupa
  - Elementi skupa mogu biti samo nepromjenjivi objekti
    - Dopušteno: konstante, n-torke, ...
    - Nije dopušteno: liste, ..
  - Redoslijed nije definiran unaprijed

```
s = {1, (2, 3), '4', 5, 5 }
```

Skupovi

- Zadatak
  - Stvoriti nekoliko skupova
  - Pokušati stvoriti skup koji sadrži listu
  - Ispitati tip nekog skupa (funkcija `type`)

Skupovi

- Stvaranje skupa
  - Prazni skup

```
skup = set()
```

  - Klasično, definiranjem elemenata
  - Pretvorbom iz liste ili n-torke pomoću razreda `set`

```
skup = set((1, (2, 3), '4', 5, 5 ))
```

  - Veličina skupa – funkcija `len`

Skupovi

- Dodavanje elemenata u skup
  - Ako element koji se dodaje već postoji – ne dodaje se
  - Metoda `add`
    - Argument je jedan element
  - Metoda `update`
    - Argument je skup, lista ili n-torka elemenata

```
skup = { 1, 2, 3, 4 }
skup.add( 4 )
skup.add( 5 )
skup.update( { 4, 6} )
skup.update( [4, 7] )
```

Skupovi

- Uklanjanje elemenata iz skupa
  - Metoda `discard`
    - Uklanja element iz skupa, ne izaziva iznimku ako element nije u skupu
  - Metoda `remove`
    - Uklanja element iz skupa, izaziva iznimku ako element nije u skupu
  - Metoda `pop`
    - Uklanja neki element iz skupa, izaziva iznimku ako je skup prazan

- `skup = { 1, '2', 3, 4 }`
- `skup.discard( 4 )`
- `skup.discard( 4 )`
- `skup.remove( '2' )`
- `skup.pop()`

Skupovi

- Izbacivanje duplikata iz liste

```
l = list(set(l))
```

Skupovi

- Zadatak
  - Napisati funkciju `dupli(l)` koja ispisuje broj duplih elemenata u listi `l`

```python
def dupli(l):
    # create an empty dictionary to store element counts
    counts = {}

    # loop over each element in the list and count occurrences
    for elem in l:
        if elem in counts:
            counts[elem] += 1
        else:
            counts[elem] = 1

    # count number of elements with counts greater than 1
    num_duplicates = sum(1 for count in counts.values() if count > 1)

    # print the number of duplicate elements
    print("Number of duplicates:", num_duplicates)
```

Skupovi

- Operacije nad skupovima
- Ispitivanje pripadnosti – operator `in`
- Unija skupova – metoda `union (|)`
- Presjek skupova – metoda `intersection (&)`
- Razlika skupova – metoda `difference (-)`
- Jednakost skupova – operator `==`
- Ispitivanje nadskupa – metoda `issuperset`
- Ispitivanje podskupa – metoda `issubset`

Skupovi

- Zadatak
  - Napisati funkciju `jaccard(a, b)` koja računa jaccardov indeks skupova `a` i `b`

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

```
jaccard([1,2,3], [3,4,5])
0.2
```

Mape

Mape

- Neuređen skup parova ključ-vrijednost
  - Ključevi su jedinstveni
  - Vrijednosti ključeva ne moraju biti jedinstveni
- Definiranje elementa mape uključuje dodavanje ključa i istovremeno definiranje vrijednosti za ključ
  - Vrijednost ključa može se obrisati ili promijeniti
  - Vrijednost ključa može se lako dohvatiti

Mape

- Stvaranje mape
  - Slično skupu, ali elementi su parovi ključ-vrijednost
  - Ključevi su jedinstveni, nema duplikata
  - Vrijednosti različitih ključeva mogu biti različitih i bilo kojih tipova
  - Ključevi mogu imati različite tipove
  - Ali samo neke – znakovni nizovi, brojevi i još neke


```
mapa = { 1: 'prvi', '2': 'drugi', 'treci': [3,2,1]}
```

Mape

- Stvaranje prazne mape

```
mapa = {}
mapa = dict()
```

Mape

- Dohvaćanje elemenata
  - Za zadani ključ dohvaća se vrijednost, slično listama

```
mapa = {'1000' : 1000}
v1 = mapa['1000']
v2 = mapa.get('1000')
v3 = mapa.get('1000', -1)
```

- Definicija, promjena i uklanjanje parova ključ vrijednost
  - Slično listama

```
mapa['10'] = 1001
del mapa['10']
```

Mape

- Veličina mape = broj parova ključ-vrijednost – funkcija `len`
- Popis svih ključeva mape – metoda `keys`
- Popis svih vrijednosti – metoda `values`
- Postojanje ključa u mapi – operator `in`

```
len(mapa)
mapa.keys()    # python3: list(mapa.keys())
mapa.values() # python3: list(mapa.values())
'1000' in mapa
```

Mape

- Još ugrađenih funkcija

```
.items()
.pop(x)
.update(d)
```

Mape

- Zadatak
  - Definirati mapu:

```
CITIES = {
    'zg': 790000,
    'st': 167000,
    'ri': 128000,
    'os': 84000
}
```

  - Napisati funkcije
    - `add_city(cities, name, population)`
      - dodaje grad i broj stanovnika u mapu `cities`
    - `remove_city(cities, name)`
      - briše grad iz mape `cities`
    - `max_city(cities)`
      - Vraća najveći broj stanovnika (ne ime grada, nego broj stanovnika)
      - Ugrađene funkcije `min(list)`, `max(list)`
    - `min_city(cities)`
      - Vraća najmanji broj stanovnika

```python
CITIES = {
    'zg': 1,
    'st': 2
}


def add(cities: map, name: str, pop: int):
    cities[name] = pop
    return cities

def remove(cities : map, name : str):
    if name in cities:
        del cities[name]
        return cities

a = add(CITIES, "a", 1)
print(a)
b = remove(CITIES,"a1")
print(b)

print(max(CITIES.values()))
print(min(CITIES.values()))
```

Mape

- Zadatak
  - Napisati funkciju `max_city_name(cities)`
    - Vraća ime grada s najvećim broj stanovnika
  - Koristiti
    - `dict.values()`
    - `dict.keys()`
    - `list.index(element)`
    - `max(list)`
  - Napomena: `dict.keys()` i `dict.values()` vraćaju ključeve, odnosno vrijednosti, a njihove pozicije odgovaraju parovima ključ-vrijednost iz mape

List vs Map (Set)

List vs Map (Set)

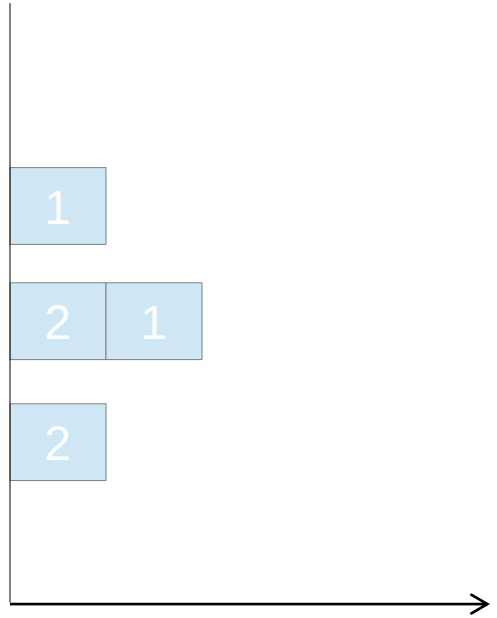
- List
  - Kada je potrebno čuvati redoslijed
  - Dohvaćanje preko cjelobrojnog indeksa
  - Prosječne vremenske složenosti
    - `append: O(1)`
    - `insert: O(n)`
    - `get/set: O(1)`
    - `delete: O(n)`
    - `x in s: O(n)`
- Dict
  - Ne osigurava redoslijed
  - Dohvaćanje preko proizvoljnog ključa
  - Prosječne (amortizirane) vremenske složenosti
    - `get/set: O(1)`
    - `delete: O(1)`
    - `x in s: O(1)`

Red kao List/Deque

Red kao List/Deque

- Red se može implementirati preko `list`
- Koriste se metode `insert/pop`

```
l = []
l.insert(0,1)
l.insert(0,2)
l.pop()
l.pop()
```



Red kao List/Deque

- Obična Python lista
  - Operacije uglavnom `O(n)`
- Ugrađena kolekcija `deque`
  - Vrlo učinkovita, npr. `pop` → `O(1)`

```
from collections import deque
q = deque()
q.append(1)
q.append(2)
```