ITU ACM Student Chapter Course Program

Introduction to Python

Week 4

Instructor
Serra Bozkurt

Assistants
Zafer Yıldız
Hüseyin Averbek

Week 4

PYTHON'DA VERİ YAPILARI

Python'da karşılaştığımız farklı problemlere daha etkili ve verimli çözümler üretebilmek için çeşitli veri yapıları kullanırız. Bunun nedeni, yazılan her programda verilerin depolanması, düzenlemesi gibi temel unsurların kullanılmasıdır. Bilgiler verimli bir şekilde saklandığında, kullanılmak istendiği zaman elde edilmesi çok daha kolaydır. Bu şekilde kod, ilgili veri yapısı kullanılmayan versiyonundan hızlı çalışacaktır.

İnceleyeceğimiz veri yapıları şunlardır: liste, dictionary (sözlük) ve tuple.

Listeler

Python'da list, yani liste, herhangi sayıda ve türde objeleri içinde bulunduran bir sandık vazifesi görür. Diğer dillerdeki listelerden en önemli farkı, bir listede birden fazla tip öğenin yanyana bulunabilmesi. Python içi bir kıyaslama yapılacak olursa da diğer veri tiplerinden (dict, set vs.) farkı, ögelerinin değiştirilebilir (mutable) olması ve sıralı olmasıdır.

Bir liste oluşturmak için, köşeli parantezler arasında, virgüllerle ayrılmış ifadeler sıralarız;

```
ilk_listemiz_rakamlar = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(type(ilk_listemiz_rakamlar))
```

```
<class 'list'>
```

Veya boş bir liste tanımlanıp içine daha sonradan elemanlar eklenebilir. Bunun iki yolu vardır;

```
bos_bir_liste = list()
print(type(bos_bir_liste))
```

```
<class 'list'>
```

Veya;

```
bos_ikinci_liste = []
print(type(bos_ikinci_liste))
print(bos_ikinci_liste)
```

```
<class 'list'>
[]
```

append()

Boş listelere eleman eklemek için append metodu kullanılır. Kullanımı list.append(eleman) şeklindedir.

```
bos_bir_liste = []
print(bos_bir_liste)

bos_bir_liste.append(1)
print(bos_bir_liste)
```

```
[]
[1]
```

Append tek seferde ancak tek eleman ekleyebilir. Ancak elemanın tipi önemli değildir.

```
bos_bir_liste = []
print(bos_bir_liste)

bos_bir_liste.append(1, 2)              # burada iki eleman eklemeye çalışıp␣
 ↪virgülle ayrıdık
print(bos_bir_liste)
```

```
[]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-4f5280b7389a> in <module>
      2 print(bos_bir_liste)
      3
----> 4 bos_bir_liste.append(1, 2)              # burada iki eleman eklemeye␣
 ↪çalışıp virgülle ayrıdık
      5 print(bos_bir_liste)

TypeError: append() takes exactly one argument (2 given)
```

```
bos_bir_liste = []
print(bos_bir_liste)

bos_bir_liste.append([1, 2])    # burada tek bir sayılar listesi ekledik
print(bos_bir_liste)

bos_bir_liste.append(6)         # burada tek bir sayı ekledik
print(bos_bir_liste)

# tek bir string eklemeye çalışalım
bos_bir_liste.append("Farklı veri tipleri tek listede bulunabilir ispatı")
print(bos_bir_liste)
```

```
[]
[[1, 2]]
[[1, 2], 6]
[[1, 2], 6, 'Farklı veri tipleri tek listede bulunabilir ispatı']
```

Örnekte görüldüğü gibi farklı veri tiplerine sahip ögeler tek bir listede toplanabildi. Bunun önemini tuple veri tipini görünce daha iyi anlayacağız.

Elemanlara erişmek

Stringlerdeki len(string) metodu stringin uzunluğunu döndürdüğü gibi, list objelerinde de len(Liste) listedeki eleman sayısını döndürür.

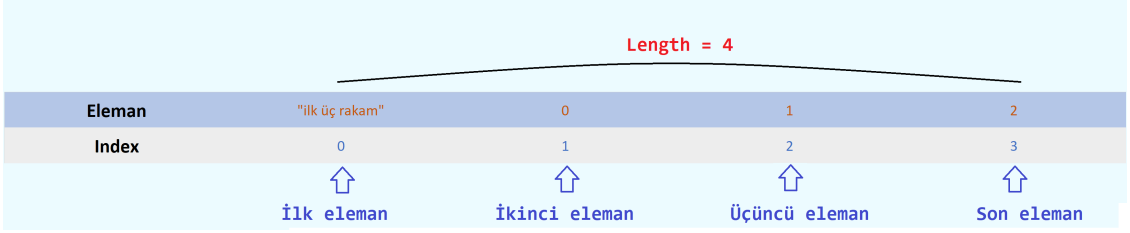
```
listem = ["ilk 3 rakam", 0, 1, 2]

eleman_sayisi = len(listem)

print(eleman_sayisi)
```

```
4
```

Listem[n]: n'inci sıradaki elamanı döndürür. Listem[n:k] ise n'inci sıradaki elemandan, k'ıncı sıradaki elemana kadar olan elemanlardan oluşan listeyi döndürür. Buna indexing metodu denir. **İlk elemanın sırası 0 kabul edilir**. Ve diğer elemanların da sıralaması ona göre yerleştirilir.




```
listem = ["ilk 3 rakam", 0, 1, 2]

print(listem[0])      # ilk elemanı çağırdık

print(listem[2])      # 0 başlangıçlı sıralamyla 2. elemanı çağırdık
```

```
ilk 3 rakam
1
```

Eğer listenin uzunluğundan daha büyük bir index'teki elemanı çağırmak istersek program hata verir.

```
listem = ["ilk 3 rakam", 0, 1, 2]

print(listem[5])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-40-38836dd2a82f> in <module>
      1 listem = ["ilk 3 rakam", 0, 1, 2]
      2
----> 3 print(listem[5])

IndexError: list index out of range
```

Belirli bir elemanın indexini bulmak içinse list.index(eleman) metodu kullanılır;

```
listem = ["ilk 3 rakam", 0, 1, 2]

ikinin_indexi = listem.index(2)
stringin_indexi = listem.index("ilk 3 rakam")

print(ikinin_indexi, stringin_indexi)
```

```
3 0
```

Listelerin sadece belirli indexler arasındaki eleemanları üzerinde işlem yapmak istersek listeyi bölebilir veya yalnızca o kısmı kullanabiliriz. Bunun için Listem[n, k] metodu kullanılır. Bu metodda alınan ilk eleman n'inci indexteki, son eleman ise k-1'inci indextekidir.

```
listem = []

for i in range(7):        # 0 dan 6 ya kadar olan saayıları
    listem.append(i)      # listeye ekliyoruz

print(listem)

print(listem[1:4])        # listenn belli bir bölümünü görüyoruz
```

```
[0, 1, 2, 3, 4, 5, 6]
[1, 2, 3]
```

Listeleri Değiştirmek

Listede belirli bir indexteki elemanı değiştirmek için basitçe o indexe yeni eleman atamak yeterli olur;

```
listem = ["ilk 3 rakam", 0, 1, 2]
print("Liste değişmeden önce: ", listem)

listem[0] = 3
print("Liste değişim sonrası: ", listem)
```

```
Liste değişmeden önce:  ['ilk 3 rakam', 0, 1, 2]
Liste değişim sonrası:  [3, 0, 1, 2]
```

Listede belirli bir indexteki elemanı tamamen silmek içinse list.pop(index) metodu kullanılır;

```
listem = ["ilk 3 rakam", 0, 1, 2]
print("Liste değişmeden önce: ", listem)

listem.pop(0)
print("Liste değişim sonrası: ", listem)
```

```
Liste değişmeden önce:  ['ilk 3 rakam', 0, 1, 2]
Liste değişim sonrası:  [0, 1, 2]
```

Listelerde Döngüler

Döngülerin listelerde eleman arama veya belirli bir eleman aralığı üzerinde işlem yapma gibi çeşitli işlevleri vardır. For-in yapısı, liste elemanları üzerinde gezinmek için en basit yollardan biridir;

```
listem = ["ilk 3 rakam", 0, 1, 2]

for eleman in listem:          # sırayla listedeki tüm elemanlar üzerinde gezinir
    print(eleman)              # ve her birini tek tek bastırır
```

```
ilk 3 rakam
0
1
2
```

Eğer listede bir elemanın var olup olmadığını kontrol etmek istiyorsak, if ve in yapılarını da kullanabiliriz.

```
listem = ["ilk 3 rakam", 0, 1, 2]
aranan_eleman = 1

if aranan_eleman in listem:
    print("Listemde", aranan_eleman, "elemanı vardır!")
else:
    ("Maalesef listemde", aranan_eleman, "elemanı bulunmamaktadır!")
```

```
Listemde 1 elemanı vardır!
```

Tuplelar

Tuplelar birden fazla veri türünü bir arada bulundurabilen virgüllerle veya parantez ile gösterilen immutable(değiştirilemeyen) veri tipleridir. Listelerden farkı **immutable(değiştirilemez)** olmalarıdır.

Tuplelar listelerin aksine parantez ile gösterilirler. Bir liste oluşturmak için, parantezler arasında, virgüllerle ayrılmış ifadeler sıralarız;

```
tuple1 = ("Serra", "Zafer", "Hüseyin")
print(type(tuple1))
```

```
<class 'tuple'>
```

Veya boş bir liste tanımlanabilir. Bunun iki yolu vardır;

```
bos_bir_tuple = tuple()
print(type(bos_bir_tuple))
```

```
<class 'tuple'>
```

Veya;

```
bos_ikinci_tuple = ()
print(type(bos_ikinci_tuple))
print(bos_ikinci_tuple)
```

```
<class 'tuple'>
()
```

Tek elemanlı bir tuple tanımlanırken dikkatli olunmalıdır. Örneğin:

```
tek_elemanlı_tuple = ("Eleman1")

print(type(tek_elemanlı_tuple))
```

```
<class 'str'>
```

Bu durumun çözümü de elemandan sonra virgül konmasıdır:

```
tek_elemanlı_tuple = ("Eleman1",)

print(type(tek_elemanlı_tuple))
```

```
<class 'tuple'>
```

Tuple Elemanlarına Erişmek

Tuple elemanlarına erişmenin mantığı listelerdeki mantığın birebir aynısıdır.

```
sehir_listesi = ["Adana","Ankara","Mersin","İstanbul"]
print(sehir_listesi[2])

sehir_tuple = ("Adana","Ankara","Mersin","İstanbul")
print(sehir_tuple[2])
```

```
Mersin
Mersin
```

Belirli bir elemanın indexine ulaşmak için list.index(eleman) metodu kullanılır;

```
sehir_tuple = ("Adana","Ankara","Mersin","İstanbul")
print(sehir_tuple.index("Mersin"))
```

```
2
```

Fakat tuplelara sonradan eleman eklenemez ve tuplelardaki elemanlar değiştirilemez.

Ancak tuplelar silinebilirler:

```
sehir_tuple = ("Adana","Ankara","Mersin","İstanbul")

del sehir_tuple

print(sehir_tuple)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-5801757253ce> in <module>
      3 del sehir_tuple
      4
----> 5 print(sehir_tuple)

NameError: name 'sehir_tuple' is not defined
```

Tuplelarda Döngüler

Listelerde olduğu gibi tuplelar da döngülerde kullanılabilir.

```
tuple1 = ["ilk 3 rakam", 0, 1, 2]

for eleman in tuple1:          # sırayla tupledaki tüm elemanlar üzerinde gezinir
    print(eleman)              # ve her birini tek tek bastırır
```

```
ilk 3 rakam
0
1
2
```

Eğer listede bir elemanın var olup olmadığını kontrol etmek istiyorsak, if ve in yapılarını da kullanabiliriz.

```
tuple1 = ["ilk 3 rakam", 0, 1, 2]
aranan_eleman = 1

if aranan_eleman in tuple1:
    print("Tupleda", aranan_eleman, "elemanı vardır!")
else:
    ("Maalesef tupleda", aranan_eleman, "elemanı bulunmamaktadır!")
```

```
Tupleda 1 elemanı vardır!
```

Sözlükler

Sözlükler (dictionary) Python dilinin son derece işe yarar veri tiplerinden bir tanesidir. Sözlükler, şimdiye kadar gördüğümüz tüm veritiplerinden yapısı gereği farklıdır. Sözlüğün içindeki her bir eleman listelerin aksine indeks ile değil **anahtar (key) ve değer (value) çiftleri** olarak tutulur bu açıdan gerçek hayattaki sözlüklere benzerler.

Basit bir sözlük örneği:

```
dictionary = {"computer": "bilgisayar",

              "technical": "teknik", # "technical" anahtar, "teknik" değer.

              "university": "üniversite"}

print(dictionary)
```

```
{'computer': 'bilgisayar', 'technical': 'teknik', 'university': 'üniversite'}
```

Sözlük Değerlerine Erişmek Sözlüklerde bir *değeri* elde etmek için *anahtarları* kullanacağız:

```
dictionary = {"sıfır":0,
              "bir":1,
              "iki":2,
              "üç":3}

print(dictionary["bir"]) #"bir" anahtarına karşılık gelen değer yazdırılır.

print(dictionary["üç"]) #"üç" anahtarına karşılık gelen değer yazdırılır.
```

```
1
3
```

Sözlükte **olmayan bir anahtarın** değerine erişmeye çalıştığımızda ise **KeyError** alırız.

```
print(dictionary["dokuz"])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-19-5022e31974b3> in <module>
----> 1 print(dictionary["dokuz"])

KeyError: 'dokuz'
```

Boş bir sözlük oluşturmak

```
dictionary= dict()
type(dictionary)
```

```
dict
```

Sözlüğe Değer Eklemek

```
dictionary = {"bir":1,
              "iki":2,
              "üç":3}

dictionary["dört"] = 4

print(dictionary)
```

```
{'bir': 1, 'iki': 2, 'üç': 3, 'dört': 4}
```

İç İçe Sözlükler

Tıpkı listeler gibi sözlükler de iç içe oluşturulabilir.

```
dictionary = {"sebzeler":{"patates":"yaz","karnabahar": "kış"},
     "meyveler":{"kiraz":"yaz","portakal":"kış"}}

print(dictionary["sebzeler"]["patates"])

print(dictionary["meyveler"]["portakal"])
```

```
yaz
kış
```

Temel Sözlük Metodları

1- **values()** metodu sözlüğün değerlerini bir *liste* olarak döner.

```
dictionary = {"sıfır":0,
           "bir":1,
           "iki":2,
           "üç":3}

dictionary.values()
```

```
dict_values([0, 1, 2, 3])
```

2- **keys()** metodu sözlüğün anahtarlarını bir *liste* olarak döner.

```
dictionary.keys()
```

```
dict_keys(['sıfır', 'bir', 'iki', 'üç'])
```

3- **items()** metodu sözlüğün tüm anahtar-değer çiftlerini bir *liste* içindeki *demetler* olarak döner.

```
dictionary.items()
```

```
dict_items([('sıfır', 0), ('bir', 1), ('iki', 2), ('üç', 3)])
```