FIZ220_EST_UygulamaNotlari_02_Listeler_Sozlukler

March 13, 2020

1 Uygulama Notları: 2

1.1 FİZ220 - Bilgisayar Programlama II | 13/03/2020

Listeler ve Sözlükler (+ demetler, kümeler) * Listeler * Listeye eleman ekleme + append + insert + extend + Liste Birleştirme * Liste elemanlarına ulaşma * Listeden eleman çıkarma + clear + pop + remove + del * Listelerle ilgili faydalı komutlar / metotlar + len + index + in + count + sort + reverse + copy * Sözlükler * Sözlük tanımlama / özellik ekleme * Sözlükten özellik silme * Sözlüğün anahtarlarını (özelliklerini) öğrenme * Listeler ile sözlükler birbirlerini çok severlerse... * Bilerek bahsedilmeyenler * Bir listenin verilen -ardışık olmayan- indislerdeki elemanlarına ulaşmak * Bir listeden belli bir değere sahip olan tüm elemanları çıkartmak * Demetler (Tuple) nerede, niye bahsedilmedi? * Bir de kümeler (set) varmış, onları niye görmüyoruz?

Dr. Emre S. Taşcı, emre.tasci@hacettepe.edu.tr
Fizik Mühendisliği Bölümü
Hacettepe Üniversitesi

2 Listeler, Demetler ve Sözlükler

Python'da GNU Octave'da görmüş olduğumuz dizilerin yaklaşık karşılığı olarak **liste** (*list*) ve **demet** (*tuple*) önerilebilir; hücre dizilerinin yaklaşık karşılığı ise **sözlük** (*dictionary*) olmaktadır. Ama bu birebir karşılık değildir: Python'da bir liste/demet karma veri tiplerinden oluşabilir. Sözlüklerin gücü ise, anahtar (*key*) indislemeden gelmektedir.

2.1 Listeler

Python listeleri sıralıdır, ilk elemanın indisi 0 olup, göreceğimiz üzere bütün sıralamalar 0'dan başlar. Listeleri tanımlamak için köşeli parantez içine yazarız.

[1]:
```
liste = [1,"iki","3"]
print(liste)
print(liste[1])
```

```
[1, 'iki', '3']
iki
```

Liste elemanlarını istediğimiz gibi değiştirebiliriz:

[2]:
```
liste[1] = "bir"
print(liste[1])
```

```
bir
```

...ama tanımlı olmayan bir elemanı indisle ekleyemeyiz:

[3]:
```
liste[3] = "üç"
```

```
␣
↪---------------------------------------------------------------------------

       IndexError                                Traceback (most recent call␣
↪last)

       <ipython-input-3-fe6a5363db8d> in <module>
   ----> 1 liste[3] = "üç"


       IndexError: list assignment index out of range
```

2.1.1 Listeye eleman ekleme

Bir listeye farklı yollardan eleman ekleyebiliriz: * **append:** `append()` komutu listenin sonuna belirttiğimiz elemanı ekler:

[4]:
```
liste.append("üç")
print(liste)
```

```
[1, 'bir', '3', 'üç']
```

- **insert:** `insert()` komutu ile listenin istediğimiz bir yerine (aradan) bir eleman ekleriz:

[5]:
```
liste.insert(1,"beş")
print(liste)
```

```
[1, 'beş', 'bir', '3', 'üç']
```

Görüldüğü üzere, 1 numaralı indise "beş" değeri eklenip, sağındaki değerler bir yana kaydırılmıştır.

- **extend:** Bazen listeye birden fazla değişken eklemek gerekir. Bu gibi durumlarda `extend()` komutu ile işimizi görürüz:

[6]:
```
liste.extend(['yedi',8])
print(liste)
```

```
[1, 'beş', 'bir', '3', 'üç', 'yedi', 8]
```

- **Liste Birleştirme:** Son olarak, bazen elimizdeki iki listeyi birleştirip, yeni bir liste oluşturmak işimize en uygun çözüm olabilir. Bu gibi durumlarda da liste birleştirme işlemcisi olan + sembolünü kullanırız:

[7]:
```
liste_1 = ["sıfır",1,2,3]
liste_2 = ["dört",5,2]
liste_son = liste_1 + liste_2

print(liste_1)
print(liste_2)
print(liste_son)
```

```
['sıfır', 1, 2, 3]
['dört', 5, 2]
['sıfır', 1, 2, 3, 'dört', 5, 2]
```

2.1.2 Liste elemanlarına ulaşma

Listenin bir kısım elemanına ulaşmak için, yukarıdaki örneklerde yaptığımız gibi ilgili elemanın indisini belirtebilir, ya da birden fazla elemana ulaşmak istiyorsak birden fazla indisi aralık veya tek tek girebiliriz:

[8]:
```
print("Bütün listemiz: ",liste_son)
print("4 indisli eleman: ",liste_son[4])
print("3, 4 ve 5 indisli elemanlar: ", liste_son[3:6])
print("3 indisli eleman ve sonrası: ",liste_son[3:])
print("Baştan 3 indisli elemana kadar olan elemanlar: ",liste_son[:3])
```

```
Bütün listemiz:  ['sıfır', 1, 2, 3, 'dört', 5, 2]
4 indisli eleman:  dört
3, 4 ve 5 indisli elemanlar:  [3, 'dört', 5]
3 indisli eleman ve sonrası:  [3, 'dört', 5, 2]
Baştan 3 indisli elemana kadar olan elemanlar:  ['sıfır', 1, 2]
```

Aralığı belirlerken -bir ihtimal GNU Octave'dan alışageldiğimiz üzere- `[3:5]` şeklinde değil de, `[3:6]` şeklinde yazdığımıza dikkat edin! Bunun sebebi, Python'da aralık belirtirken son değerin *kadar ve dahil* şeklinde değil de, ***kadar*** olarak yorumlanmasıdır. Aralıklarda son değer içerilmez. Bu bir tercih meselesidir ve pek çok dilde (örn: C, C++, Go, Haskell, Java, JavaScript, Perl, PHP, Visual Basic) bu şekilde alınmaktadır -- bkz. "sıralı tam liste" 8). Bu şekildeki tercihin birçok açıklaması vardır, sözü *pirimiz* Edgar W. Dijkstra'ya bırakırsak başlıca sebepler olarak: 1. Başlangıç ve bitiş parametrelerinin farkı doğrudan aralıktaki eleman sayısını vermesini, 2. Döngü kontrollerinde pratiklik ve tek işlem (sadece *küçük mü?* değerlendirmesi) sağlamasını, 3. (Ayrıca) Dizinin göstergecinin (*pointer*) doğrudan hafızadaki adresi olmasını ve bu adresin aynı zamanda ilk elemanın tutulduğu yerin başlangıcı olmasını,

sayabiliriz.

Ayrıca son iki örnekte görüldüğü üzere, başlangıç veya bitişin belirtilmediği durumlarda (sırası ile) 0 ve *eleman sayısı + 1* alınır.

Python'da -özellikle de listenin eleman sayısını bilmediğimizde- negatif indisleri kullanarak **sondan** da ilerleyebiliriz. '-1' indisi her zaman için dizinin sonuncu elemanına; '-2' indisi sondan bir önceki elemanına, vs.. işaret eder. Aralıkları da benzer şekilde kullanabiliriz:

[9]:
```
print("Bütün listemiz: ",liste_son)
print("\nSonuncu eleman: ",liste_son[6])
print("Sonuncu eleman: ",liste_son[-1])
print("Sondan bir önceki eleman: ",liste_son[-2])
print("Sondan iki ve üç önceki elemanlar: ",liste_son[-3:-1])
```

```
Bütün listemiz:  ['sıfır', 1, 2, 3, 'dört', 5, 2]

Sonuncu eleman:  2
Sonuncu eleman:  2
Sondan bir önceki eleman:  5
Sondan iki ve üç önceki elemanlar:  ['dört', 5]
```

Pozitif ve negatif indisleri birlikte de kullanabiliriz, örneğin, 3 indisli elemandan sondan ikinci elemana *kadar* (dahil değil!) olan elemanlara:

[10]:
```
print("3 indisli elemandan sondan ikinci elemana kadar olan elemanlar:␣
 ↪",liste_son[3:-2])
```

```
3 indisli elemandan sondan ikinci elemana kadar olan elemanlar:  [3, 'dört']
```

şeklinde ulaşabiliriz.

Listenin eleman sayısını bulmak içinse `len()` fonksiyonunu kullanırız:

[11]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2]
print("Bütün listemiz: ",liste_son)
print("Listemizdeki eleman sayısı: ",len(liste_son))
```

```
Bütün listemiz:  ['sıfır', 1, 2, 3, 'dört', 5, 2]
Listemizdeki eleman sayısı:  7
```

Son bir not olarak, aralıklarda başlangıç ve sondan başka, adım sayısını da belirtebiliriz (3. parametre olarak). Başlama ya da bitiş belirtilmezse, otomatik olarak -sırasıyla- ilk ve son elemanlar alınır. Örneğin:

[12]:
```
sayilar = [0,1,2,3,4,5,6,7,8,9,10]
print(sayilar[3:8:2]) # 3 indisten, 8'e kadar, 2 artarak gider
print(sayilar[:8:2]) # İlk elemandan 8'e kadar, 2 artarak gider
print(sayilar[1::2]) # 1 indisten, sona kadar, 2 artarak gider
print("")

# Aşağıdaki iki aralık kullanımının farkına dikkat edin!
print(sayilar[:10:2]) # Baştan 10'a *kadar* 2 artarak gider
```

```
print(sayilar[::2]) # Baştan *sona* kadar (son dahil!) 2 artarak gider
```

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

[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8, 10]
```

2.1.3 Listeden eleman çıkarma

Tıpkı eklemede olduğu üzere, listeden eleman çıkarmak için de birden fazla yöntem vardır:

- **clear:** `clear()` komutu ile listeyi boşaltırız:

[13]:
```
liste_gidici = ["A",1,2,"üç"]
print("Mevcut liste: ",liste_gidici)
liste_gidici.clear()
print("clear() çekilmiş liste: ",liste_gidici)
```

```
Mevcut liste:  ['A', 1, 2, 'üç']
clear() çekilmiş liste:  []
```

- **pop:** istediğimiz bir indise sahip elemanı `pop()` komutu ile uçurabiliriz:

[14]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2]
liste_son.pop(1)
print(liste_son)
```

```
['sıfır', 2, 3, 'dört', 5, 2]
```

indisler sözkonusu olduğundan, dilersek negatif indis de kullanabiliriz; örneğin sondan ikinci elemanı çıkarmak istersek:

[15]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2]
liste_son.pop(-2)
print(liste_son)
```

```
['sıfır', 1, 2, 3, 'dört', 2]
```

İndis belirtilmediği durumda ise doğrudan son eleman çıkartılır.

`pop()`la ilgili faydalı bir özellik de, çıkardığı elemanın değerini döndürmesidir, örneğin:

[16]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2]
cikan = liste_son.pop(1)
print(liste_son)
print("Listeden cikarttigimiz eleman: ",cikan)
```

```
['sıfır', 2, 3, 'dört', 5, 2]
Listeden cikarttigimiz eleman:  1
```

- **remove:** Listemizden belli bir değere sahip olan elemanı çıkartmak için `remove()` komutunu kullanırız. Örneğin, değeri "5" olan elemanı çıkartalım:

[17]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2]
liste_son.remove(5)
print(liste_son)
```

```
['sıfır', 1, 2, 3, 'dört', 2]
```

`remove()` komutunu kullanırken dikkat edilmesi gereken şey, eğer belirttiğimiz değerden birden fazla varsa, sadece başa en yakın olanı çıkartacaktır. Örneğin, listemizden "2" değerini çıkarmasını istersek:

[18]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2, 213]
liste_son.remove(2)
print(liste_son)
```

```
['sıfır', 1, 3, 'dört', 5, 2, 213]
```

- **del:** `deli pop()` gibi düşünebiliriz ama ona ek olarak, aralık boyunca da silme yapmamıza izin verir:

[19]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2, 213]

# Tek bir eleman silelim
del liste_son[3]
print(liste_son)
del liste_son[-2]
print(liste_son)

# Aralik silelim
del liste_son[1:4]
print(liste_son)
```

```
['sıfır', 1, 2, 'dört', 5, 2, 213]
['sıfır', 1, 2, 'dört', 5, 213]
['sıfır', 5, 213]
```

Aralık verirken de yine pozitif indislerle negatif indisleri beraber kullanabiliriz:

[20]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2, 213]
print(liste_son)
# 2 indisli elemandan sondan 3. elemana *kadar* silelim
del liste_son[2:-3]
print(liste_son)
```

```
['sıfır', 1, 2, 3, 'dört', 5, 2, 213]
['sıfır', 1, 5, 2, 213]
```

2.1.4 Listelerle ilgili faydalı komutlar / metotlar

`len, index, in, count, sort, reverse, copy`

Listelerin elemanlarına nasıl ulaşıp, ekleme-çıkartma yapabileceğimizi gördükten sonra, sıklıkla kullanılan komut ve metotları inceleyip, derleyelim:

- **len:** listenin eleman sayısını döndürür:

[21]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2, 213]
print("Listemiz: ",liste_son)
print("Listemizdeki eleman sayısı: ",len(liste_son))
```

```
Listemiz:  ['sıfır', 1, 2, 3, 'dört', 5, 2, 213]
Listemizdeki eleman sayısı:  8
```

- **index:** İlgili listenin verilen elemanının indisini döndürür:

[22]:
```
print("Listemizde 'dört' değerinin indisi: ",liste_son.index("dört"))
print("Listemizde '213' değerinin indisi: ",liste_son.index(213))
```

```
Listemizde 'dört' değerinin indisi:  4
Listemizde '213' değerinin indisi:  7
```

Eğer aranılan değer listede yoksa hata ('ValueError') uyarısı verecektir:

[23]:
```
print("Listemizde '777' değerinin indisi: ",liste_son.index(777))
```

```
␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call␣
↪last)

<ipython-input-23-3617a6e8bbe1> in <module>
----> 1 print("Listemizde '777' değerinin indisi: ",liste_son.index(777))


ValueError: 777 is not in list
```

Normal şartlarda hata alıp programın çalışmasının durması pek hoş değildir, bu nedenle ya bu hatayı yakalamaya çalışırız, ya da bir başka prosedür olan `in` yapısını kullanırız:

Hatayı yakalamak suretiyle icabına bakmak
(Bu yol biraz daha ileri bir teknik, şimdilik atlayabilirsiniz)

[24]:
```
indis = "yok"
deger = 777
try:
```

```
    indis = liste_son.index(deger)
except ValueError:
    print(deger," değeri listede bulunamadı.")
print(deger," değerinin bulunduğu indis: ",indis)

print("----------------------")

deger = 213
try:
    indis = liste_son.index(213)
except ValueError:
    print(deger," değeri listede bulunamadı.")
print(deger," değerinin bulunduğu indis: ",indis)
```

```
777  değeri listede bulunamadı.
777  değerinin bulunduğu indis:  yok
----------------------
213  değerinin bulunduğu indis:  7
```

in prosedürünün kullanımı ile işimizi görmek:

[25]:
```
print("dört" in liste_son)
print(777 in liste_son)
```

```
True
False
```

- **count:** Bir değerin listede kaç kere bulunduğunu verir:

[26]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2, 213, 2]
print("Listemiz: ",liste_son)
print("Listemizde 2 değerinden ",liste_son.count(2)," adet vardır.")
print("Listemizde 'dört' değerinden ",liste_son.count("dört")," adet vardır.")
print("Listemizde 219 değerinden ",liste_son.count(219)," adet vardır.")
```

```
Listemiz:  ['sıfır', 1, 2, 3, 'dört', 5, 2, 213, 2]
Listemizde 2 değerinden  3  adet vardır.
Listemizde 'dört' değerinden  1  adet vardır.
Listemizde 219 değerinden  0  adet vardır.
```

(yukarıdaki örneğimizde de göreceğiniz üzere, listede aranan değerin bulunmaması durumunda hata değil, 0 değerini alırız)

- **sort:** Listeyi sıralamakta kullanılır. Standard dışı bir sıralama için ek parametreleri kabul eder. Dikkat edilecek nokta, bunun bir işlem olmasıdır. Listeyi sıralayıp, o hale getirir, bir sonuç döndürmez, yani `print("Sıralı listemiz: ",liste_sayilar.sort())` komutunu çalıştırdığımızda ekrana listeye dair bir şey yazılmaz ama liste çevrilmiş olur; bu sebepten önce sıralama işlemini yapıp, sonra listeyi yazdırabiliriz (bkz. aşağıdaki örnek).

[27]:
```
liste_sayilar = [67, 1, 6, 2, 3, 5, 2, 213, 2]
print("Sayı listemiz: ",liste_sayilar)
liste_sayilar.sort() # listemizi siraladik
print("Sıralı sayı listemiz: ",liste_sayilar)

print("---------------")

liste_isimler = ["Berk","Ahmet","Cengiz","Ebru"]
print("İsim listemiz: ",liste_isimler)
liste_isimler.sort() # listemizi siraladik
print("Sıralı isim listemiz: ",liste_isimler)
```

```
Sayı listemiz:  [67, 1, 6, 2, 3, 5, 2, 213, 2]
Sıralı sayı listemiz:  [1, 2, 2, 2, 3, 5, 6, 67, 213]
---------------
İsim listemiz:  ['Berk', 'Ahmet', 'Cengiz', 'Ebru']
Sıralı isim listemiz:  ['Ahmet', 'Berk', 'Cengiz', 'Ebru']
```

Dikkat! Sort metodu karma veri tipleriyle (örneğin sayılar ve stringler) doğrudan çalışamamaktadır (bu iş için yukarıda bahsettiğimiz `key` ek kıstas parametresi/bilgisini kullanmak gerekir).

- **reverse:** Listemizin sırasını tersine çevirir -- bu metot da, tıpkı `sort()` gibi, doğrudan liste üzerine etki eder.

[28]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2, 213, 2]
print("Listemiz: ",liste_son)
liste_son.reverse()
print("Tersine çevrilmiş listemiz: ",liste_son)
```

```
Listemiz:  ['sıfır', 1, 2, 3, 'dört', 5, 2, 213, 2]
Tersine çevrilmiş listemiz:  [2, 213, 2, 5, 'dört', 3, 2, 1, 'sıfır']
```

- **Liste kopyalama:** Liste kopyalama işlemi, bir ihtimal `liste_2 = liste_1` şeklinde sanılabileceği kadar kolay değildir. Aşağıdaki örneğe bir göz atalım:

[29]:
```
liste_1 = [1,2,3]
liste_2 = liste_1
liste_1.append(4)
print(liste_2)
```

```
[1, 2, 3, 4]
```

Nasıl yani??? Biz, `liste_1`'e, `liste_2`'yi onun önceki halini eşitledikten sonra eleman eklemiştik -- nasıl oldu da sonradan eklediğimiz eleman `liste_2`'ye de eklendi???

Bunun sebebi, dilin yapısından kaynaklanmaktadır. Değişken isimleri aslında göstergeç (*pointer*) denilen, değerin tutulduğu adresi saklamakla yükümlüdürler. Tek değerli değişkenlerde birebir ilişki olsa da, birden fazla değişkeni tutan derleme yapılarda, isim doğrudan adresi gösterir. Bu yüzden `liste_2 = liste_1` dediğimizde, `liste_1`'de tutulan adres bilgisi `liste_2`'ye atanır (ikisi de aynı yeri gösterir). Bunu da daha açık bir şekilde -bir nevi- adresleri gösteren `id()` komutu ile kontrol edebiliriz:

[30]:
```
print("liste_1'in  adresi: ",id(liste_1))
print("liste_2'nin adresi: ",id(liste_2))
```

```
liste_1'in  adresi:  140044461705288
liste_2'nin adresi:  140044461705288
```

Bundan ötürü `liste_1`'de değişiklik yapsak, `liste_2`'de de yapsak, ikisinin de etkilenmek üzere gösterdiği adres aynı olduğundan ikisi de değişir (yazılımda bu durumlar yumuşak kopya (*soft copy*) olarak da adlandırılır).

Eğer yapmak istediğimiz *gerçekten* birbirlerine o anda eşit, ama sonrasında bağımsız iki liste oluşturmaksa, bunu da `copy()` metodu ile elde ederiz:

[31]:
```
liste_1 = [1,2,3]
liste_2 = liste_1.copy()
liste_1.append(4)
print(liste_2)

print("liste_1'in  adresi: ",id(liste_1))
print("liste_2'nin adresi: ",id(liste_2))
```

```
[1, 2, 3]
liste_1'in  adresi:  140044461270344
liste_2'nin adresi:  140044461272328
```

Bir listenin bazı elemanlarını bir başka liste olarak atamak istediğimizi düşünelim:

[32]:
```
liste_1 = ['sıfır', 1, 2, 3, 'dört', 5, 2, 213, 2]
print("liste_1: ",liste_1)

# Bu listenin 2 indisli elemanından
# sondan 3. elemanına kadar olan elemanları
# liste_2 olarak atayalım:
liste_2 = liste_1[2:-3]
print("liste_2: ",liste_2)
```

```
liste_1:  ['sıfır', 1, 2, 3, 'dört', 5, 2, 213, 2]
liste_2:  [2, 3, 'dört', 5]
```

İkinci liste, birinci listenin bazı elemanlarını çağırıp eşitlediğimizden, birinci listeden bağımsız. Bunu, birinci listenin 4 indisli elemanını değiştirerek kontrol edebiliriz:

[33]:
```
print(liste_1[4])
liste_1[4] = "four"
print("liste_1: ",liste_1)
print("liste_2: ",liste_2)
print("----------")
print("liste_1'in  adresi: ",id(liste_1))
print("liste_2'nin adresi: ",id(liste_2))
```

```
dört
liste_1:  ['sıfır', 1, 2, 3, 'four', 5, 2, 213, 2]
liste_2:  [2, 3, 'dört', 5]
----------
liste_1'in  adresi:  140044490008392
liste_2'nin adresi:  140044461270344
```

Görüldüğü üzere, biz ikinci listenin ilgili elemanını birinci listeden almış olsak da, sonrasında birinci listede ilgili eleman değişse bile, listeler bağımsız olduğundan, ikinci liste bu değişiklikten etkilenmemekte.

Bu özelliği uça götürüp, ikinci listeyi, birinci listenin *tüm elemanlarına* (`liste_1[:]`) da eşitleyerek tanımlayabiliriz -- sonuç yine iki bağımsız liste olacaktır:

[34]:
```
liste_1 = [1,2,3]
liste_2 = liste_1[:]
liste_1.append(4)
print(liste_2)

print("liste_1'in  adresi: ",id(liste_1))
print("liste_2'nin adresi: ",id(liste_2))
```

```
[1, 2, 3]
liste_1'in  adresi:  140044461269512
liste_2'nin adresi:  140044461272200
```

2.2 Sözlükler

Sözlükler (*dictionary*), girişte de kısaca bahsedildiği üzere, GNU Octave'daki *hücre dizilerine* benzerler. Listelerden farkı, indislerin de tanımlanabilir olmasıdır. Bir sözlüğün indisini *özellik* veya *anahtar* (*key*) terimleri ile belirtiriz.

2.2.1 Sözlük tanımlama / özellik ekleme

Hemen örneğe geçip, bir `ogrenci` sözlüğü üzerinden, özellikleri tanımlayalım:

[35]:
```
# Bos bir sozluk tanimlayarak baslayalim:
ogrenci_1 = {}

# ad ve numara ozelliklerini ekleyelim:
ogrenci_1["ad"]="Ayşe Celik"
ogrenci_1["numara"] = 12345678

# ozellikleri ille tek tek girmek zorunda da degiliz
#ogrenci["matematik 123":"B1", "fizik 125":"A2", "giris yili":20191]

print("'ogrenci_1' sözlüğümüz: ",ogrenci_1)
print("Öğrencinin Adı: ",ogrenci_1["ad"])
```

```
print("Öğrencinin Numarası: ",ogrenci_1["numara"])
```

```
'ogrenci_1' sözlüğümüz:  {'ad': 'Ayşe Celik', 'numara': 12345678}
Öğrencinin Adı:  Ayşe Celik
Öğrencinin Numarası:  12345678
```

Özellikleri, başta sözlüğümüzü tanımlarken de girebiliriz:

[36]:
```
ogrenci_2 = {"ad":"Barış Ateş", "numara":21912123}
print("'ogrenci_2' sözlüğümüz: ",ogrenci_2)
print("Öğrencinin Adı: ",ogrenci_2["ad"])
print("Öğrencinin Numarası: ",ogrenci_2["numara"])
```

```
'ogrenci_2' sözlüğümüz:  {'ad': 'Barış Ateş', 'numara': 21912123}
Öğrencinin Adı:  Barış Ateş
Öğrencinin Numarası:  21912123
```

Dahası, `update()` metodunu kullanarak halihazırda mevcut bir sözlüğe, topluca özellik eklemesi de yapabiliriz:

[37]:
```
ogrenci_2 = {"ad":"Barış Ateş", "numara":21912123}
print("'ogrenci_2' sözlüğümüz: ",ogrenci_2)

print("----------")

ogrenci_2.update({"matematik 123":"B1", "fizik 125":"A2", "giris yili":20191})
print("'ogrenci_2' sözlüğümüz: ",ogrenci_2)
print()
print("Öğrencinin Adı: ",ogrenci_2["ad"])
print("Öğrencinin Numarası: ",ogrenci_2["numara"])
print("Öğrencinin giriş yılı: ",ogrenci_2["giris yili"])
print("Öğrencinin Matematik 123 notu: ",ogrenci_2["matematik 123"])
print("Öğrencinin Fizik 125 notu: ",ogrenci_2["fizik 125"])
```

```
'ogrenci_2' sözlüğümüz:  {'ad': 'Barış Ateş', 'numara': 21912123}
----------
'ogrenci_2' sözlüğümüz:  {'ad': 'Barış Ateş', 'numara': 21912123, 'matematik
123': 'B1', 'fizik 125': 'A2', 'giris yili': 20191}

Öğrencinin Adı:  Barış Ateş
Öğrencinin Numarası:  21912123
Öğrencinin giriş yılı:  20191
Öğrencinin Matematik 123 notu:  B1
Öğrencinin Fizik 125 notu:  A2
```

Bir özellik tanımlanırken, önceden tanımlanmışsa, yeni değerle güncellenir; halihazırda tanımlı değilse o özellik eklenir:

[38]:
```
ogrenci_2 = {'ad': 'Barış Ateş', 'numara': 21912123, 'matematik 123': 'B1',␣
↪'fizik 125': 'A2', 'giris yili': 20191}

# Daha onceden tanimli olan "ad" ozelligini guncelliyoruz:
ogrenci_2["ad"] = "Barış Cengiz Ateş"

# Daha onceden tanimlanmamis olan "kimya 155" ozelligini ekliyoruz:
ogrenci_2["kimya 155"] = "C1"

print("Öğrencinin Adı: ",ogrenci_2["ad"])
print("Öğrencinin Numarası: ",ogrenci_2["numara"])
print("Öğrencinin giriş yılı: ",ogrenci_2["giris yili"])
print("Öğrencinin Matematik 123 notu: ",ogrenci_2["matematik 123"])
print("Öğrencinin Fizik 125 notu: ",ogrenci_2["fizik 125"])
print("Öğrencinin Kimya 155 notu: ",ogrenci_2["kimya 155"])

print("---------------")

# update() metodu ile toplu guncelleme/ekleme yapabiliriz:
ogrenci_2.update({"kimya 155":"B2","fizik lab 101":"C3"})
print("Öğrencinin Adı: ",ogrenci_2["ad"])
print("Öğrencinin Numarası: ",ogrenci_2["numara"])
print("Öğrencinin giriş yılı: ",ogrenci_2["giris yili"])
print("Öğrencinin Matematik 123 notu: ",ogrenci_2["matematik 123"])
print("Öğrencinin Fizik 125 notu: ",ogrenci_2["fizik 125"])
print("Öğrencinin Fizik Lab 101 notu: ",ogrenci_2["fizik lab 101"])
print("Öğrencinin Kimya 155 notu: ",ogrenci_2["kimya 155"])
```

```
Öğrencinin Adı:  Barış Cengiz Ateş
Öğrencinin Numarası:  21912123
Öğrencinin giriş yılı:  20191
Öğrencinin Matematik 123 notu:  B1
Öğrencinin Fizik 125 notu:  A2
Öğrencinin Kimya 155 notu:  C1
---------------
Öğrencinin Adı:  Barış Cengiz Ateş
Öğrencinin Numarası:  21912123
Öğrencinin giriş yılı:  20191
Öğrencinin Matematik 123 notu:  B1
Öğrencinin Fizik 125 notu:  A2
Öğrencinin Fizik Lab 101 notu:  C3
Öğrencinin Kimya 155 notu:  B2
```

2.2.2 Sözlükten özellik silme

Silmek istediğiniz özelliğin değerini `del` (`del sozluk["ozellik"]`) komutuyla silebilirsiniz:

[39]:
```
ogrenci_2 = {'ad': 'Barış Ateş', 'numara': 21912123, 'matematik 123': 'B1',␣
 ↪'fizik 125': 'A2', 'giris yili': 20191}
print("'ogrenci_2' sözlüğümüz: ",ogrenci_2)

# Matematik 123 not bilgisini silelim:
del ogrenci_2["matematik 123"]
print("'ogrenci_2' sözlüğümüz: ",ogrenci_2)
```

```
'ogrenci_2' sözlüğümüz:  {'ad': 'Barış Ateş', 'numara': 21912123, 'matematik
123': 'B1', 'fizik 125': 'A2', 'giris yili': 20191}
'ogrenci_2' sözlüğümüz:  {'ad': 'Barış Ateş', 'numara': 21912123, 'fizik 125':
'A2', 'giris yili': 20191}
```

2.2.3 Sözlüğün anahtarlarını (özelliklerini) öğrenme

Sözlük değişkeninin sahip olduğu özelliklerin listesini `list()` komutu ile elde ederiz:

[40]:
```
ogrenci_2 = {'ad': 'Barış Ateş', 'numara': 21912123, 'matematik 123': 'B1',␣
 ↪'fizik 125': 'A2', 'giris yili': 20191}
print("ogrenci_2 sözlüğünün anahtarları (özellikleri): ", list(ogrenci_2))
```

```
ogrenci_2 sözlüğünün anahtarları (özellikleri):  ['ad', 'numara', 'matematik
123', 'fizik 125', 'giris yili']
```

Adından da anlaşılacağı üzere, `list()` komutu bize anahtarları bir liste halinde verir (giriliş sırası ile). Döndürdüğü sonuç liste cinsinden olduğundan, bütün liste metotlarını ve indisle ulaşmasını kullanabiliriz:

Aynı anda hem anahtarı, hem de değeri almak içinse `items()` metodundan faydalanırız:

[41]:
```
ogrenci_2 = {'ad': 'Barış Ateş', 'numara': 21912123, 'matematik 123': 'B1',␣
 ↪'fizik 125': 'A2', 'giris yili': 20191}
ogrenci_2_anahtarlar = list(ogrenci_2)
print("ogrenci_2 sözlüğünün 2 ve 3 indisli anahtarları: ",␣
 ↪ogrenci_2_anahtarlar[2:4])

print(ogrenci_2.items())
```

```
ogrenci_2 sözlüğünün 2 ve 3 indisli anahtarları:  ['matematik 123', 'fizik 125']
dict_items([('ad', 'Barış Ateş'), ('numara', 21912123), ('matematik 123', 'B1'),
('fizik 125', 'A2'), ('giris yili', 20191)])
```

2.3 Listeler ile sözlükler birbirlerini çok severlerse...

Listeler ile sözlükleri birbirleri içinde geçişli olarak kullanmak mümkündür (*sözlük listeleri*, *liste listeleri*, *sözlük sözlükleri*, vs..). Bu bize veri derlerken müthiş bir esneklik sağlar. Örneğin, sözlük olarak tanımladığımız `ogrenci` sözlüklerimizi bir liste altında derleyelim:

[42]:
```
ogrenci_1 = {'ad': 'Ayşe Celik', 'numara': 12345678}
ogrenci_2 = {'ad': 'Barış Ateş', 'numara': 21912123, 'matematik 123': 'B1',␣
 ↪'fizik 125': 'A2', 'giris yili': 20191}

ogrenciler_listesi = [ogrenci_1, ogrenci_2]
print(ogrenciler_listesi)

print ("--------")
print ("0 indisli öğrencinin adı: ",ogrenciler_listesi[0]["ad"])
print ("1 indisli öğrencinin adı: ",ogrenciler_listesi[1]["ad"])
```

```
[{'ad': 'Ayşe Celik', 'numara': 12345678}, {'ad': 'Barış Ateş', 'numara':
21912123, 'matematik 123': 'B1', 'fizik 125': 'A2', 'giris yili': 20191}]
--------
0 indisli öğrencinin adı:  Ayşe Celik
1 indisli öğrencinin adı:  Barış Ateş
```

(normal uygulamalarda böyle 0., 1. diye tek tek çağırmayacağız tabii: bir döngü üzerinden güzelce işleyeceğiz derlenmiş bilgileri ama o kısım `for` döngüsünü öğrenince gelecek. ;)

3 Bilerek bahsedilmeyenler

Bir listenin verilen -ardışık olmayan- indislerdeki elemanlarına ulaşmak
GNU Octave'ın aksine (örn: `liste_son[2,4]`), maalesef bunu yapmanın doğrudan bir yolu yoktur.

Ama boş geçmemek adına:

[43]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2]
# 2 ve 4 indisli elemanlara ulaşalım:

# 1. yöntem: map() + __getitem__ fonksiyon ve metotları ile:
liste_24_1 = list(map(liste_son.__getitem__,[2,4]))
print("1. yöntem (map() + __getitem__): ",liste_24_1)

# 2. yöntem: operator modülünden itemgetter() fonksiyonunu kullanarak:
from operator import itemgetter
liste_24_2 = list(itemgetter(*[2,4])(liste_son))
print("2. yöntem (operator.itemgetter): ",liste_24_2)
```

```
1. yöntem (map() + __getitem__):  [2, 'dört']
2. yöntem (operator.itemgetter):  [2, 'dört']
```

Neyse ki -pek çok diğer derdin yanında- bu dertten de bizi NumPy dizileri kurtarıyor:

[44]:
```
import numpy as np
liste_son_np = np.array(['sıfır', 1, 2, 3, 'dört', 5, 2])
print(liste_son_np[[2,4]])
```

```
['2' 'dört']
```

Bir listeden belli bir değere sahip olan tüm elemanları çıkartmak `remove()` komutunun sadece ilk bulduğu elemanı çıkardığını, gerisine karışmadığını gördük, ama bazen bütün ilgili değerleri çıkarmak isteyebiliriz. Bu durumda `filter()` fonksiyonu yardımımıza koşmakta:

[45]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2]
liste_son_filtre = list(filter(lambda x: x != 2, liste_son))
print(liste_son_filtre)
```

```
['sıfır', 1, 3, 'dört', 5]
```

(oradaki `lambda` yönergesi tek satırda, tek kullanımlık fonksiyon yazmamızı sağlayan bir yönerge)

Ya da `filter`'ı bir kenara bırakıp doğrudan `for`+`if` kombosu çekebiliriz:

[46]:
```
liste_son = ['sıfır', 1, 2, 3, 'dört', 5, 2]
liste_son_filtre = list(s for s in liste_son if s != 2)
print(liste_son_filtre)
```

```
['sıfır', 1, 3, 'dört', 5]
```

Demetler (*Tuple*) nerede, niye bahsedilmedi? Demetler pek çok açıdan listelere benzese de, listelerin aksine *değiştirilemezler* (*immutable*). Yani bir demete *(doğrudan)* yeni eleman ekleyemez, içinden eleman çıkaramaz, mevcut elemanı da değiştiremezsiniz. E o zaman ne anladık? diyebilirsiniz ama demetlerin kullanımının avantajlı olduğu yerler de vardır (referans tabloları ve kayıtlar gibi, sabit kalmasını istediğimiz verileri demetlerde toplarız). Demetleri tanımlarken normal parantezler kullanırız, ama elemanlarına ulaşırken tıpkı listelerdeki gibi köşeli parantezlerle ulaşırız örneğin:

[47]:
```
d = (0,1,"iki",3)
print("Demetimiz: ",d)
print("Demetimizin 1 indisli elemanı: ",d[1])
print("Demetimizin sondan 2. elemanı: ",d[-2])
print("Demetimizin 2 ve 3 indisli elemanları: ",d[2:4])
```

```
Demetimiz:  (0, 1, 'iki', 3)
Demetimizin 1 indisli elemanı:  1
Demetimizin sondan 2. elemanı:  iki
Demetimizin 2 ve 3 indisli elemanları:  ('iki', 3)
```

Kendi işlerimizde demetlere pek ihtiyacımız olmadığı için, bir de değiştirilemez oluşlarından ötürü bu aşamada çareden çok dert olacaklarından, demetlere girmeden teğet geçiyoruz.

Bir de kümeler (*set*) varmış, onları niye görmüyoruz? Kıvrık parantezlerle '{}' tanımlanan kümeler de listelere benzemekle beraber, en temel özelliği bir elemanın sadece bir kere yer almasıdır. Bu özellikleri listelerde basit bir döngüyle de sağlanabildiğinden dolayı -benim bildiğim kadarıyla- çok yaygın bir kullanımı yoktur. Kümelerin bir diğer ayırt edici yönü ise, elemanlarına tek tek ulaşımın mümkün olmamasıdır (yani indis kullanarak tek bir elemana ulaşamayız, ancak bütün üzerinden döngü kurabiliriz -- bir nevi *ya hep, ya hiç* 8).

[48]:
```
k = {0,1,"iki",1,"iki",2,4}
print("Kümemiz: ",k)
```

```
Kümemiz:  {0, 1, 2, 4, 'iki'}
```