Listeler

Listeler diğer programlama dillerinden tanıdığımız dizi yapısına benzer. Birden fazla ve türde veriyi içlerinde barındırabilirler.

In [31]:

```
notlar = [54, 45, 12.3, True, "Python"]
print(notlar)
```

```
[54, 45, 12.3, True, 'Python']
```

Listelerde index değeri kullanılarak liste içerisindeki elemanlara ulaşılabilir. Ayrıca ':' operatörü ile belirli aralıktaki değerlere bir arada ulaşma imkanımız da vardır.

In [32]:

```
print(notlar[1])
```

```
45
```

In [33]:

```
print(notlar[-1])
```

```
Python
```

In [34]:

```
print(notlar[-3:])
```

```
[12.3, True, 'Python']
```

Listelere append fonksiyonu yardımıyla veri eklenebilir ayrıca index değeri belirtilerek liste içerisinde bulunan değerler de değiştirilebilir.

In [35]:

```
notlar[1] = 100
print(notlar)
```

```
[54, 100, 12.3, True, 'Python']
```

In [36]:

```
notlar.append("Yeni Değer")
print(notlar)
```

```
[54, 100, 12.3, True, 'Python', 'Yeni Değer']
```

Listelere veri eklemek için kullanılabilicek farklı bir metod da insert metodudur. Insert metodunun append metodundan farkı eklenecek verinin hangi index numarasına tekabül edeceğini belirliyebilmemizdir.

In [37]:

```
print("Eski hali =>{}".format(notlar))

notlar.insert(3,"Daha yeni değer")

print("Yeni hali =>{}".format(notlar))
```

```
Eski hali =>[54, 100, 12.3, True, 'Python', 'Yeni Değer']
Yeni hali =>[54, 100, 12.3, 'Daha yeni değer', True, 'Python', 'Yeni Değer']
```

Listeler sıklıkla döngüler ile birlikte kullanılırlar. Python dilinde bir çok veri yapısı karakter veri tipleri de dahil olmak üzere itere edilebilirdir.Bir çok dilde buluanan foreach döngü yapısına benzetmek mümkündür.

In [38]:

```
for index in notlar:
    print(index)
```

```
54
100
12.3
Daha yeni değer
True
Python
Yeni Değer
```

Split ve List

Split metodu bir String ifadeyi tekrar eden bir karaktere göre bölüp liste olarak ele almamızı sağlar.

In [39]:

```
notlar = "12,45,66,12,76,99,1"
print(notlar)
```

```
12,45,66,12,76,99,1
```

In [40]:

```
notlar =  notlar.split(',')
print(notlar)
```

```
['12', '45', '66', '12', '76', '99', '1']
```

Her zaman verilen bir string ifade yukarıda verilmiş olan örnekte olduğu gibi düzenli bir biçimde virgüllerle ayrılmamıştır. Bu durumda list fonksiyonu kullanılır. Bu fonksiyon her karakterin listenin ayrı bir elemanı olduğunu varsayar.

In [41]:

```
notlar = "1245661276991"
notlar = list(notlar)
print(notlar)
```

```
['1', '2', '4', '5', '6', '6', '1', '2', '7', '6', '9', '9', '1']
```

İç İçe Liste

Listeler içinde başka bir liste barındırabilir.

In [42]:

```
liste = ["Yönetim", "Bilişim", "Sistemleri",[54,45,62], 65, 22, 33,3.6]
print(len(liste))
```

```
8
```

Çıktıyı incelediğimizde liste içerisinde yer alan listenin elemanları toplamıyla değil tek bir öğe olarak sayılmıştır. Liste içerisinde bulunan başka bir listeye ulaşmak için;

In [43]:

```
liste[3][1]
```

Out[43]:

```
45
```

komutu kullanılabilir.

Liste Manipülasyonları

Listelerden eleman çıkarma işlemi için pop, remove, ve del method ve fonksiyonları bulunmaktadır.

- **pop Listeden eğer parametre belirtilmez ise son elemanı siler ve sildiği değeri geri döndürür.**

In [44]:

```
liste = ["Yönetim", "Bilişim", "Sistemleri",[54,45,62], 65, 22, 33,3.6]
silinen_deger = liste.pop()
print(silinen_deger)
```

```
3.6
```

In [45]:

```
silinen_deger = liste.pop(1)
print(silinen_deger)
```

```
Bilişim
```

- **remove Metodunun pop metodundan farkı index değeriyle değilde doğrudan silinecek olan objeyi referans vermemizdir. Silinen değeri/objeyi değil listenin yeni halini döndürür.**

In [46]:

```
print(liste)
liste.remove("Sistemleri")
print(liste)
```

```
['Yönetim', 'Sistemleri', [54, 45, 62], 65, 22, 33]
['Yönetim', [54, 45, 62], 65, 22, 33]
```

- **del Listeden belirli bir veriyi yada aralığı siler.**

In [47]:

```
print(liste) #Eski Hali
del liste[0]
print(liste) #Yeni Hali
```

```
['Yönetim', [54, 45, 62], 65, 22, 33]
[[54, 45, 62], 65, 22, 33]
```

count, index ve sort Metodları:

- **count Bir objenin bir liste içerisinde kaç defa tekrar ettiğini bulur.**

In [48]:

```
liste = ["Yönetim", "Bilişim", "Bilişim", "Sistemleri",[54,45,62], 65, 22, 33,3.6]
print(liste.count("Bilişim"))
```

```
2
```

- **index Bir objenin liste içerisinde hangi index numarasıyla yer aldığını bulur.**

In [49]:

```
print(liste.index("Sistemleri"))
```

3

- **sort Listeyi sıralamamızı sağlar**

In [2]:

```
liste_1 = ["Yönetim","Bilişim","Sistemleri"]
liste_1.sort()
print(liste_1)
```

```
['Bilişim', 'Sistemleri', 'Yönetim']
```

In [3]:

```
liste_2 = [15,12,3,45,2]
liste_2.sort()
liste_2.reverse() # Bir listeyi tersine çeviren metod
print(liste_2)
```

```
[45, 15, 12, 3, 2]
```

Listeleri Birleştirmek

Örnek: İki farklı şirket tek bir çatı altında birleşme kararı almış ve müşterilerini birleştirek yeni oluşturdukları veri tabanına yazmak istiorlar. Bunun için ilk önce listeleri birleştirmeleri gerekmektedir. çözüme alternatif olarak **extend** metodu da bu işi yapmaktadır.

In [7]:

```
a_sirket_musterileri = ["Ertuğrul","Faysal"]
print(a_sirket_musterileri)
```

```
['Ertuğrul', 'Faysal']
```

In [8]:

```
b_sirket_musterileri = ["Eda","Yasin","Fatma"]
print(b_sirket_musterileri)
```

```
['Eda', 'Yasin', 'Fatma']
```

In [9]:

```
yeni_musteriler = a_sirket_musterileri + b_sirket_musterileri
print(yeni_musteriler)
```

```
['Ertuğrul', 'Faysal', 'Eda', 'Yasin', 'Fatma']
```

Listeleri Kopyalamak

Örnek: Bir işletme müşteri isimlerinin bulunduğu listenin üzerinde pazarlama çalışmalarını yürütmek istemektedir, ortaya çıkabilicek bir aksilik için orjinal listenin yedeğini almak istemektedir

In [17]:

```
musteri_listesi = ["Ahmet", "Alper", "Büşra","Ceyhun","Deniz"]
yedek_musteri_listesi = musteri_listesi
```

Yapılan bir hata sonucu müşteri listesinin son üç kaydı kaybolmuştur. Yedek müşteri listesine dönülmek istenmektedir.

In [18]:

```
del musteri_listesi[2:] # == [-3:]
print(musteri_listesi)
```

```
['Ahmet', 'Alper']
```

In [23]:

```
print(yedek_musteri_listesi)
```

```
['Ahmet', 'Alper']
```

Ancak yedek listesine dönülmek istendiğinde ise son üç kaydın bu listeden de silindiği görülmektedir. Bunun sebebi oluşturduğumuz yedeğin aslında müşteri_listesini yanlızca referans göstermesidir çünkü Python üzerinde değişkenler değiştirilebilir(muable) ve değiştirilemez(immutable) olarak ikiye ayrılır. Yani Liste, küme ve sözlük yapılar değiştirilebilir iken tupple gibi veri yapıları değiştirilemez türdedir. Detaylı Bilgi için Bu durumu aşabilmek için listemizi kopyalarken baştan oluşturmamız gerekmektedir. Çözüme alternatif olarak copy metodu kullanılabilir.

In [24]:

```
id(yedek_musteri_listesi)
```

Out[24]:

```
1392546433672
```

In [26]:

```
id(musteri_listesi) ## İki değişkenimiz aslında ram'de aynı değişkenlermiş!
```

Out[26]:

```
1392546433672
```

In [27]:

```
musteri_listesi = ["Ahmet", "Alper", "Büşra","Ceyhun","Deniz"]
yedek_musteri_listesi = musteri_listesi[:] # Doğrudan referans değil, yorumlayıcı listeyi yeniden oluşturmaya zorlanıyor
del musteri_listesi[-3:]
print(yedek_musteri_listesi)
```

```
['Ahmet', 'Alper', 'Büşra', 'Ceyhun', 'Deniz']
```

In [29]:

```
id(musteri_listesi)
```

Out[29]:

```
1392547463240
```

In [28]:

```
id(yedek_musteri_listesi)
```

Out[28]:

```
1392547414088
```