

21 Kasım 2017

# Bilgisayar Bilimi

Kur1 Ders Notları

Veri Yapıları

Ahmet Tuna POTUR

<b>İçindekiler</b>	<b>i</b>
<b>1. Veri Yapıları</b>	<b>1</b>
1.1. <i>None</i> Veri Tipi	1
1.2. <i>Bool</i> Veri Tipi	1
1.2.1. <i>bool()</i> fonksiyonu	1
1.3. <i>bin</i> ve <i>hex</i> Veri Tipi	1
1.3.1. <i>bin()</i> ve <i>hex()</i> Fonksiyonları	1
1.4. <i>Liste (list)</i> Veri Tipi	1
1.4.1. <i>list()</i> Fonksiyonu ve Listeleri Oluşturmak	2
1.4.2. İndeks(index) Liste Elemanlarına Erişmek	2
1.4.3. Liste Elemanlarının Değerlerini Değiştirme	3
1.4.4. İndeks Sınırlarına Çıkmak	4
1.4.5. <i>len()</i> Fonksiyonu	4
1.4.6. <i>in</i> Operatörü	4
1.4.7. Dilimleme	5
1.4.8. <i>range()</i> Fonksiyonu	7
1.4.9. İç İççe Listeler	8
1.4.10. Liste Metotları	9
1.4.10.1. <i>append()</i> Metodu	9
1.4.10.2. <i>pop()</i> Metodu	9
1.4.10.3. <i>sort()</i> Metodu ve <i>sorted()</i> Fonksiyonu	10
1.4.10.3.1. <i>sort()</i> Metodu	10
1.4.10.3.2. <i>sorted()</i> Fonksiyonu	11
1.4.10.3.3. Karakter Dizisi Elemanlarının Alfabetik Sıralanması	11
1.4.10.4. <i>extend()</i> Metodu	12
1.4.10.5. <i>insert()</i> Metodu	13
1.4.10.6. <i>remove()</i> Metodu	14
1.4.10.7. <i>index()</i> Metodu	15
1.4.10.8. <i>count()</i> Metodu	16
1.4.10.9. <i>clear()</i> Metodu	16
1.4.10.10. <i>copy()</i> Metodu	16
1.4.10.11. <i>reverse()</i> Metodu	17
1.4.10.11.1. <i>reverse()</i> Metodu	17

1.4.10.11.2. reversed() Fonksiyonu .....	17
1.4.11. Liste Verileri Üzerinde Doğrudan Çalışmak .....	18
1.4.12. Liste İşlemleri.....	18
1.5. Demetler (tuple) Veri Tipi.....	20
1.6. Karakter Dizileri (string) Veri Tipi .....	20
1.6.1. Karakter Dizilerini Oluşturma .....	21
1.6.2. Özel Karakterler .....	22
1.6.3. Formatlama .....	22
1.7. print() Fonksiyonu Hakkında Detaylı Bilgi .....	22
1.8. Sözlük Veri Tipi .....	23
1.9. Kümeler (Sets) Veri Tipi .....	23
<b>2. Kullanışlı Python Fonksiyonları .....</b>	<b>23</b>
2.1. dir() Fonksiyonu .....	23
2.2. help() Fonksiyonu .....	23
2.3. del() Fonksiyonu .....	23
2.4. abs() Fonksiyonu .....	23
2.5. round() Fonksiyonu .....	23
2.6. max() ve min() Fonksiyonu .....	23
2.7. sum() Fonksiyonu .....	23
<b>3. Koşul Durumları.....</b>	<b>23</b>



## 1. Veri Yapıları

### 1.1. None Veri Tipi

```
>>> boş = None
>>> print(baş)
None
>>> type(baş)
<class 'NoneType'>
```

```
>>> boş = 5
>>> print(baş)
5
>>> type(baş)
<class 'int'>
```

Eğer bir değişkenin değerini sonradan belirlemek isterseniz bu değişken **None** (atanmamış anlamında) değerine eşitleyebilirsiniz. **None** tipiyle oluşturulan değişkeni sonradan değer atayıp kullanabiliriz.

### 1.2. Bool Veri Tipi

```
>>> doğru = True
>>> type(doğru)
<class 'bool'>
```

```
>>> yanlış = False
>>> type(yanlış)
<class 'bool'>
```

İngilizcede Boolean olarak geçen **bool** veri tipi sadece **True** ve **False** olarak iki değerden oluşur.

#### 1.2.1. bool() fonksiyonu

```
>>> bool(1)
True
>>> bool(375)
True
>>> bool(-57)
True
>>> a = 13
>>> bool(a)
True
>>> bool(0)
False
>>> z = 0
>>> bool(z)
False
```

```
>>> bool(3.14)
True
>>> bool(-12.25)
True
>>> bool(-98.67)
True
>>> b = 5.41
>>> bool(b)
True
>>> bool(0.0)
False
>>> y = 0.0
>>> bool(y)
False
```

**bool()** fonksiyonu değer olarak aldığı sayı verisini **bool** değere dönüştürür. Python'da **0** dışındaki tüm sayı değerleri **True** değerine karşılık gelir, **0** sayı değeri **False** değerine karşılık gelir. Üstteki **bool()** fonksiyonu örnekleri incelendiğinde **bool(0)** ve **bool(0.0)** tip dönüşümleri **False** değerini döner, bunun dışında tüm sayısal dönüşümler **True** değere dönüşür.

```
>>> 17 < 2043
True
>>> "Tuna" == "Tuna"
True
```

```
>>> 17 > 2043
False
>>> "Ahmet" == "Tuna"
False
```

**bool** veri tipi karşılaştırma operatöründen sonra ortaya çıkan sonuç değeridir. Karşılaştırma operatörlerini sonucun yanlış çıktığı durumda **False**, doğru çıktığı durumda **True** değerini döner. **bool** veri tipi ileride göreceğimiz koşullu durumlar ve döngüler konularında kullanılır.

### 1.3. bin ve hex Veri Tipi

#### 1.3.1. bin() ve hex() Fonksiyonları

### 1.4. Liste (list) Veri Tipi

Liste, sıralı bir dizidir. Listeler bir küme gibi, elemanlardan (bazen öge veya terim de denir) oluşur. Elemanların sayısına **listenin uzunluğu** denir. Kümenin aksine sıralı ve aynı öğeler dizide farklı konumlarda birkaç kez bulunabilir. Örneğin, **Fibonacci dizisi**, her sayının kendine önceki sayı ile toplanması sonucu oluşan bir sayı dizisidir. İlk iki öge 1 ile 1'dir. Böylece 9 elemanlı [1,1,2,3,5,8,13,21,34] dizisi elde edilir. **[P,Y,T,H,O,N]** ilk harfi 'K' ve son harfi 'P' olan bir listedir.

Listeler çok yararlı bir veri tipidir, indekslenirler, parçalanırlar ve üzerinde değişik işlemler yapabildiğimiz fonksiyonları barındırırlar. Bir listede her veri tipinden elemanı saklayabiliriz. Listeler her zaman birden çok eleman taşıdığından listelerin isimlerini '**veriler**', '**insanlar**', '**çalışanlar**' gibi çoğul isim olarak vermek gerekir. Python'da kareli parantez '[' ] içerisinde listeleri barındırır. Listelerin içindeki elemanlar virgülle birbirinden ayrılır.

#### 1.4.1. list() Fonksiyonu ve Listeleri Oluşturmak

```
>>> # Boş Liste
>>> liste_1 = list()
>>> liste_2 = []
>>> # Sadece int sayılar listesi
>>> liste_3 = [1,1,2,3,5,8,13,21,34]
>>> # değişik tipte veriler listesi
>>> liste_4 = [5,3.14,"Tuna"]

>>> # Boş Liste
>>> print(liste_1, liste_2)
Boş Listeler : [] []
>>> liste_3
[1, 1, 2, 3, 5, 8, 13, 21, 34]
>>> liste_4
[5, 3.14, "Tuna"]
```

**list()** fonksiyonu liste tipinde değişkenleri oluşturmak ve karşılığı olan tipleri **list()** tipine çevirmek için kullanılır.

Listeleri değer vermeden boş olarak veya değer vererek dolu olarak oluşturabiliriz.

**liste\_1** listesi: **list()** fonksiyonuyla **liste\_1 = list()** atamasıyla **boş liste**.

**liste\_2** listesi: **[]** boş liste operatörüyle **liste\_2 = []** atamasıyla **boş liste**

**liste\_3** listesi: **liste\_3 = [1,1,2,3,4,8,13,21,34]** atamasıyla **int** değerlerden oluşan Fibonacci listesi

**liste\_4** listesi: **liste\_4 = [5,3.14,"Tuna"]** atamasıyla **değişik veri tiplerinden** oluşan bir liste

oluşturulmuştur.

```
>>> # type() fonksiyonu listenin tipini list olarak gösterir
>>> veriler = [17,3.14,"Ahmet"]
>>> type(veriler)
<class 'list'>
```

Liste tipinde değişkenler **type()** fonksiyonuyla incelenebilir.

```
>>> # string değer listeye çevriliyor
>>> selam = "Merhaba Dünya"
>>> liste_selam = list(selam)
>>> liste_selam
['M', 'e', 'r', 'h', 'a', 'b', 'a', ' ', 'D', 'ü', 'n', 'y', 'a']
```

Karakter dizisi(string) **list()** fonksiyonuyla liste verisine dönüştürülebilir. Karakter dizisi listeye çevrildiğinde karakter dizisi içinde bulunan tüm harfler tek tek liste elemanı haline dönüşür.

#### 1.4.2. Indeks(index) Liste Elemanlarına Erişmek

elemanlar	= [	3	5	3.14	1.44	"Tuna"	"Emel"	[10, 20, 30]	27.57	39	]
indeks	→	0	1	2	3	4	5	6	7	8	

```

n elemanlı bir liste
a[0]          1. terim
a[1]          2. terim
a[2]          3. terim
⋮
⋮
a[n-1]        (n). terim

```

**a[indeks]** listesinde, köşeli ayraç "[ ]" içerisindeki sayıya **indeks** denmektedir. [ ] içerisine girilen **indeks** değeri liste içerisinde ulaşılacak istenen değer için kullanılır. **0**, başlangıç indeksi olduğu için **n** elemanlı bir listenin eleman sayısı **n-1** olur.

```

>>> veriler = [1,2,3,4,5,6,7]
# indeks numarasıyla elemanı çağırmak. veriler[indeks] işlemi
>>> veriler[0]          >>> veriler[5]          >>> veriler[-1]
1                        6                        7
>>> veriler[1]          >>> veriler[6]          >>> veriler[-2]
2                        7                        6

```

**veriler = [1,2,3,4,5,6,7]** atanmasıyla 1'den 7'ye sayılardan oluşan bir liste oluşturduk. **veriler[indeks]** ifadesiyle liste elemanları içinde istenen elemana ulaşılır.

**veriler[0]** değeri listenin ilk değeri 1'i  
**veriler[1]** değeri serideki ikinci değeri 2'yi  
**veriler[5]** değeri serideki altıncı değeri 6'yı  
**veriler[6]** değeri serideki yedinci değeri 7'yi  
**veriler[-1]** değeri, serideki son elemanı 7'yi.  
**veriler[-2]** değeri sondan bir önceki değeri 6'yı ifade eder.

```

>>> veriler = ["Tuna", "Can", "Emel", "Oya", "Ahmet", "Mehmet"]
# değişken ile indeks numaralı elemanı çağırmak
>>> indeks = 0          >>> indeks = -1          >>> indeks = 0
>>> veriler[indeks]     >>> veriler[indeks]     >>> veriler[indeks+1]
'Tuna'                  'Mehmet'                  'Can'
>>> indeks = 3          >>> indeks = -4          indeks = -1
>>> veriler[indeks]     >>> veriler[indeks]     >>> veriler[indeks-1]
'Oya'                   'Emel'                   'Ahmet'

```

Liste elemanlarına erişmek için tam sayı değişkenleri kullanılabilir. Değişkenin değeri o anda ne ise o indeks numarasındaki eleman çağrılır.

#### 1.4.3. Liste Elemanlarının Değerlerini Değiştirme

```

veriler = [1,2,3,4,5]
# indeks numarasıyla elemanı değiştirmek
>>> veriler[0]= 10      >>> veriler[2]= 30      >>> veriler[-1]= 50
>>> veriler             >>> veriler             >>> veriler
[10, 2, 3, 4, 5]       [10, 2, 30, 4, 5]       [10, 2, 30, 4, 50]

```

Listeler değiştirilebilen (**mutable**) bir veri tipidir. Dolayısıyla listeler üzerinde doğrudan değişiklik yapabiliriz. Listenin içerisindeki hangi elemanın değerini değiştirmek istiyorsanız indeks kullanarak istediğiniz elemanın değerini değiştirebilirsiniz. Yukarıdaki örnekte ilk elemanın, orta elemanın ve son elemanın değerleri on katlarıyla değiştirilmiş.

```

>>> veriler = ['Tuna', 'Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet']
>>> veriler[0] = 'Fırat'
>>> veriler
['Fırat', 'Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet']
>>> veriler[3] = 'Mert'
>>> veriler
['Fırat', 'Can', 'Emel', 'Mert', 'Ahmet', 'Mehmet']
>>> indeks = 4
>>> veriler[indeks] = 'Tolga'
>>> veriler
['Fırat', 'Can', 'Emel', 'Mert', 'Tolga', 'Mehmet']
>>> veriler[indeks-1] = 'Emir'
>>> veriler
['Fırat', 'Can', 'Emel', 'Emir', 'Tolga', 'Mehmet']

```

İndeks kullanarak içinde karakter dizileri barındıran listenin elemanları da değiştirilebilir.

#### 1.4.4. İndeks Sınırlarına Çıkmak

```

>>> veriler = [3,5,7,9,11]
>>> veriler[17]
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in
<module>
    veriler[17]
IndexError: list index out of range

>>> veriler[-7] = 27
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in
<module>
    veriler[-7] = 27
IndexError: list assignment index out of range

```

Eğer listeden bir eleman çağırmak için olmayan bir indeksi verirsek veya olmayan bir indeksteki elemana değer atanırsa hata çıkar.

#### 1.4.5. len() Fonksiyonu

```

>>> liste_say = [1,2,3,4,5]
>>> len(liste_say)
5

>>> liste_karisik = [7,1.22,"Can",[1,2,3]]
>>> len(liste_karisik)
4

```

Listenin eleman sayısına listenin uzunluğu denir. **len(liste)** fonksiyonu değer olarak aldığı listenin eleman sayısını verir. **len** İngilizcedeki uzunluk **length** kelimesinin kısaltmasıdır. **liste\_karisik** listesinin içerisinde **"Can"** karakter dizisi ve **[1,2,3]** listesi birer eleman olarak sayıldığı için **liste\_karisik** listesindeki eleman sayısı **4** olarak hesaplar.

```

>>> veriler = [0,1,2,3,4,5,6,7]
# eleman sayısı
>>> len(veriler)
8
# son elemanın indeks numarası
>>> veriler[len(veriler)-1]
7
# son eleman
>>> veriler[-1]
7
>>> veriler[7]
7

# Listenin Son Elemanı
>>> elemanSayisi = len(veriler)
>>> elemanSayisi
8
# eleman sayısından 1 çıkarsa
# son elemanın indeksi bulunur
>>> sonEleman=len(veriler)-1
>>> veriler[sonEleman]
7

```

**len()** fonksiyonu listelerin sayısını bulmak için kullanılıyordu. **len(veriler)-1** işlemi ile listenin eleman sayısının bir eksiği listedeki son elemanın indeks numarasını verir. **liste[len(veriler)-1]** işlemi listesindeki son elemanı getirir. **liste[-1]** işlemiyle de listenin son elemanını bulabilirsiniz.

#### 1.4.6. in Operatörü



```
>>> veriler = ["Tuna","Can","Emel","Oya","Ahmet","Mehmet"]
# Eleman verilerde varsa True          # Eleman verilerde yoksa False
>>> "Tuna" in veriler                  >>> "Tolga" in veriler
True                                   False
```

`in` operatörü bir elemanın listede olup olmadığını kontrol eder. Eğer eleman listede varsa **True** yoksa **False** döner. `in` operatörü koşullu durumlar ve döngüler konusunda yine karşımıza çıkacak.

#### 1.4.7. Dilimleme

**liste[ Başlangıç : Bitiş : Basamak ]**

```
>>> veriler = [0,1,2,3,4,5,6,7,8,9,10,11,12]
# Veriler İçerisinden Belirli Sayıda ve Koşulda Elemanları Seçme
# 3 - 9 arasında 2 şer arayla          # En baştan son elemana 3 er
>>> veriler[3:10:2]                    >>> veriler[:3]
[3, 5, 7, 9]                          [0, 3, 6, 9, 12]
# en baştan 4 Eleman                  # 2.'den son elemana 3 er
>>> veriler[:4]                        >>> veriler[2::3]
[0, 1, 2, 3]                          [2, 5, 8, 11]
# en baştan 4 Eleman                  # baştan 5'e 2 şer
>>> veriler[0:4]                       >>> veriler[:5:2]
[0, 1, 2, 3]                          [0, 2, 4]
# 3 - 10 arası elemanlar              # verilerin tersi
>>> veriler[3:10]                      >>> veriler[::-1]
[3, 4, 5, 6, 7, 8, 9]                 [12,11,10,9,8,7,6,5,4,3,2,1,0]
# 8. Elemandan son elemana           # 2'şer arayla verilerin tersi
>>> veriler[8:]                        >>> veriler[::-2]
[8, 9, 10, 11, 12]                    [12, 10, 8, 6, 4, 2, 0]
# Verilerdeki elemanların tümünü farklı bir yolla listelenmiş
>>> veriler[:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Liste içerisinde farklı aralıklarda istenen sayıda elemana ulaşılabilir. **liste[Başlangıç : Bitiş : Basamak]** işlemiyle liste içerisinde hangi elemanların seçileceği belirtilir. Üsteki örnekte liste içerisinde belirli sayıda ve koşulda elemanların nasıl seçileceği örneklerle gösterilmiştir. Listelerde yapılan bu işleme **dilimleme** denir.

```
>>> veriler = [0,1,2,3,4,5,6,7,8,9,10,11,12]
# Dilimlemede bitiş değerindeki eleman listeye alınmaz.
# 7. Eleman listelenmiyor              # 12. Eleman listelenmiyor
>>> veriler[:7]                        >>> veriler[2:12:2]
[0, 1, 2, 3, 4, 5, 6]                 [2, 4, 6, 8, 10]
>>> veriler[7]                         >>> veriler[12]
7                                      12
# 11. Eleman listelenmiyor            # 11. Eleman listelenmiyor
>>> veriler[3:11]                      >>> veriler[1:11:2]
[3, 4, 5, 6, 7, 8, 9, 10]              [1, 3, 5, 7, 9]
>>> veriler[11]                        >>> veriler[11]
11                                      11
```

Listelenen elemanlar **Bitiş** değerinden önceki elemanlardır. **Bitiş** değerindeki eleman listeye alınmaz.

```
>>> veriler = [1,2,3]
>>> veriler
[1,2,3]
# listenin tüm elemanlarını dilimlenerek değiştirmek
>>> veriler[:] = 'Oya', 'Ahmet', 'Mehmet'
>>> veriler
['Oya', 'Ahmet', 'Mehmet']
```

Bir listeye listenin eleman sayısında değer atanarak listenin tüm değerleri değiştirilebilir. `veriler[:]='Oya','Ahmet','Mehmet'` işlemleriyle liste içindeki tüm değerler değişecektir.

```
>>> veriler = ['a', 'b', 'c', 'd', 1, 2, 3, 4]
# listede belirtilen elemanlar dilimlenerek değiştirilebilir
# baştan iki eleman değiştiriliyor
>>> veriler[:2] = [50,60]
>>> veriler
[50, 60, 'c', 'd', 1, 2, 3, 4]
>>> veriler[2:4]
['c', 'd']
# 2. 3. ve 4. Eleman değiştiriliyor
>>> veriler[2:5] = [70,80,90]
>>> veriler
[50, 60, 70, 80, 90, 2, 3, 4]
```

Liste içerisinde belirtilen elemanlar değiştirilebilir. Üsteki listede baştan iki eleman **50** ve **60** değeriyle, sonra 2. eleman dahil 5. elemana kadar olan elemanlar **70**, **80** ve **90** değeriyle değiştirilmiş.

```
>>> isimler = ['Tuna', 'Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet', 'Tolga']
>>> isimler[1:5]
['Can', 'Emel', 'Oya', 'Ahmet']
>>> isimler[1:5] = ['Kerem', 'Alp', 'Ada', 'Mert']
>>> isimler
['Tuna', 'Kerem', 'Alp', 'Ada', 'Mert', 'Mehmet', 'Tolga']
```

`isimler` listesindeki 1., 2., 3. ve 4. elemanlar farklı elemanlarla değiştirildi.

```
>>> isimler = ['Tuna', 'Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet', 'Tolga']
# 2 ve 5 arasındaki elemanlar
>>> isimler[2:5]
['Emel', 'Oya', 'Ahmet']
# 2 ve 5 arasındaki elemanlar listeden siliniyor
>>> isimler[2:5] = []
>>> isimler
['Tuna', 'Can', 'Mehmet', 'Tolga']
```

`isimler[2:5] = []` işlemleriyle dilimle yöntemiyle liste içinde belirtilen elemanlar listeden silinebilir.

```
>>> veriler = [0,1,2,3,4,5,6,7,8,9,10,11,12]
# Değişkenler kullanılarak liste elemanları dilimlenebilir
# 9. Elemandan sonrası # 3. Eleman ile 11. Eleman arası 2 şer
>>> Başlangıç = 9 >>> Başlangıç, Bitiş, Basamak = 3, 11, 2
>>> veriler[Başlangıç:] >>> veriler[Başlangıç : Bitiş : Basamak]
[9, 10, 11, 12] [3, 5, 7, 9]
# 5 ve 11. Elemana kadar 11 yok # Baştan 11. Elemana 3 er
>>> Başlangıç, Bitiş = 5, 11 >>> Başlangıç, Bitiş, Basamak = 0, 11, 3
>>> veriler[Başlangıç:Bitiş] >>> veriler[Başlangıç : Bitiş : Basamak]
[5, 6, 7, 8, 9, 10] [0, 3, 6, 9]
```

Değişkenler kullanılarak dilimleme yapılabilir.

```
# Değişkenler kullanılarak liste elemanları dilimlenebilir
>>> Başlangıç,Bitiş,Basamak = None,None,None
>>> veriler = list(range(10))
>>> veriler[Başlangıç:Bitiş:Basamak]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> Başlangıç,Bitiş,Basamak = 3,None,None
>>> veriler[Başlangıç:Bitiş:Basamak]
[3, 4, 5, 6, 7, 8, 9]
>>> Başlangıç,Bitiş,Basamak = 3,8,None
>>> veriler[Başlangıç:Bitiş:Basamak]
[3, 4, 5, 6, 7]
>>> Başlangıç,Bitiş,Basamak = 3,8,2
>>> veriler[Başlangıç:Bitiş:Basamak]
[3, 5, 7]
>>> Başlangıç,Bitiş,Basamak = None,None,-1
>>> veriler[Başlangıç:Bitiş:Basamak]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Geniş kapsamlı bir program yaptığınızda listelerin elemanlarına değişkenler kullanarak erişirsiniz. Üstteki örnekte listenin **Başlangıç**, **Bitiş** ve **Basamak** değişkenlerine değer verilerek dilimlenmesi gösterilmiştir.

#### 1.4.8. range() Fonksiyonu

```
>>> veriler = list(range(10))
>>> veriler
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

**range()** fonksiyonu listeler için otomatik tam sayı değeri üretmekte kullanılır. **liste = list(range(10))** şeklinde kullanılan **range()** fonksiyonu **0'dan 10'a** (10 hariç) kadar sayılardan oluşan liste oluşturur.

```
>>> veriler = list(range(1,20,2))
>>> veriler
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> type(range(1,20,2))
<class 'range'>
>>> type(veriler)
<class 'list'>
```

**range( Başlangıç , Bitiş , Basamak )** **range()** fonksiyonun içine girilen

**Başlangıç** üretilecek ilk sayıdır

**Bitiş** kaçınıcı sayıya kadar sayı üretileceğini belirtir. Üretilen son sayı **Bitiş** değerinden bir küçüktür

**Basamak** üretilen sayıların kaçar kaçar üretileceğini belirtir.

**veriler = list(range(1,20,2))** işleminde **range()** fonksiyonu **1** den **20'ye** (**20 hariç**) kadar ikişer ikişer artan sayılardan bir dizi oluşturur ve **veriler** isimli **list** tipindeki değişkene atar. **range()** fonksiyonunu en çok döngüler konusunda kullanacağız.

```
>>> veriler = list(range(11))
>>> veriler
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> veriler = list(range(3,10))
>>> veriler
[3, 4, 5, 6, 7, 8, 9]
>>> veriler = list(range(-9,10))
>>> veriler
[-9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> veriler = list(range(0,10,3))
>>> veriler
[0, 3, 6, 9]
```

`range( Başlangıç )` şeklinde kullanılan `range()` fonksiyonu 0 dan `Başlangıç` değerine kadar sayı üretir.

`veriler = list(range(10))` 0'dan 10'a (10 hariç) kadar sayıları üretir. [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

`range( Başlangıç , Bitiş )` şeklinde kullanılan `range()` fonksiyonu `Başlangıç` - `Bitiş` arasında sayı üretir.

`veriler = list(range(3,10))` 3'ten 10'a (10 hariç) kadar sayıları üretir. [3, 4, 5, 6, 7, 8, 9]

`range( Başlangıç , Bitiş , Basamak )` şeklinde kullanılan `range()` fonksiyonu `Başlangıç` - `Bitiş` arasında sayıları `Basamak` değerine göre üretir.

`veriler = list(range(0,10,3))` 0'dan 10'a (10 hariç) kadar sayıları 3'er 3'er üretir. [0, 3, 6, 9]

#### 1.4.9. İç İçe Listeler

```
>>> veriler=[[0,1,2],[3,4,5],[6,7,8]]
>>> veriler
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
# 1. Veriler 0. eleman
>>> veriler[1][0]
3
# 0. Veriler 2. eleman
>>> veriler[0][2]
2
# 2. Veriler 2. eleman
>>> veriler[2][2]
8
# 1. Veriler 1. eleman
>>> veriler[1][1]
4
# 2. Veriler 2. eleman
>>> veriler[-1][-1]
8

>>> list_1 = [9,8,7]
>>> list_2 = [6,5,4]
>>> list_3 = [3,2,1]
>>> veriler=[list_1,list_2,list_3]
>>> veriler
[[9, 8, 7], [6, 5, 4], [3, 2, 1]]
>>> a,b = 1,2
>>> veriler[a][b]
4
>>> a,b = 0,0
>>> veriler[a][b]
9
>>> a,b = 2,2
>>> veriler[a][b]
1
>>> a,b = -1,-1
>>> veriler[a][b]
1
```

İç içe listeler bir listenin başka bir liste içinde bulunduğu listelerdir. Bu tip listeler matris veya ağaç yapılarında kullanılmaktadır. Listenin elemanlarına çağırarak için iki tane `[]` operatörü kullanılır. `veriler[2][2]` 3. listenin 3. elemanı çağırarak için kullanılır.

```
>>> veriler = ['Tuna', 'Can', ['Emel', 'Oya', 'Ahmet'], 39, 41, 3.14, 5.67]
# ['Emel', 'Oya', 'Ahmet'] liste içinde 1 eleman olarak sayılır
>>> len(veriler)
7
# ['Emel', 'Oya', 'Ahmet'] elemanına veriler[2] ile erişilir
>>> veriler[2]
['Emel', 'Oya', 'Ahmet']
# liste[2] elemanlarına erişim için veriler[2][indeks] işlemi yapılır
>>> veriler[2][0]
'Emel'
>>> veriler[2][1]
'Oya'
>>> veriler[2][1:]
['Oya', 'Ahmet']
# liste[2] elemanlarıyla liste_yeni oluşturuluyor
>>> veriler_yeni = veriler[2]
# liste_yeni içinde iç liste yok
>>> veriler_yeni
['Emel', 'Oya', 'Ahmet']
# liste_yeni elemanları liste_yeni[indeks] ile çağrılır
>>> veriler_yeni[0]
'Emel'
>>> veriler_yeni[1]
'Oya'
```

`veriler` içinde ['Emel', 'Oya', 'Ahmet'] gibi bir liste daha var. Bu liste ana listenin elemanlarından biridir ve bu da öteki elemanlar gibi tek elemanlık bir yer kaplar. liste içinde bulunan ['Emel', 'Oya', 'Ahmet'] listesi 2. indekste olduğu için bu iç listeye erişim `veriler[2]` işlemiyle yapılır. `veriler[2]` listesinin elemanlarına erişim için `veriler[2][indeks]` işlemiyle yapılır. `veriler[2]` listesi daha rahat kullanım için `veriler_yeni = veriler[2]` işlemiyle yeni bir listeye kopyalanırsa `veriler_yeni` tekli listeler gibi kullanılabilir.

#### 1.4.10. Liste Metotları

Sınıflar için özelleşmiş fonksiyonlara metod denir. Nesne tabanlı programlama konusunda metod kavramı detaylı bir şekilde anlatılacak.

##### 1.4.10.1. `append()` Metodu

```
# append() ile listelere değer ekleme
>>> veriler = [3,5,7]
>>> veriler.append(9)
>>> veriler.append(3.14)
>>> veriler.append("Can")
>>> veriler
[3, 5, 7, 9, 3.14, 'Can']
```

`append(eleman)` metodu listeye eleman eklemek için kullanılır. `append()` ile eklenen veri listenin sonuna eklenir.

```
# append() fonksiyonu ile aynı işi yapan listeye eleman ekleme
>>> veriler = [3,5,7]
>>> veriler[len(veriler):] = [9]
>>> veriler[len(veriler):] = [3.14]
>>> veriler[len(veriler):] = ["Can"]
>>> veriler
[3, 5, 7, 9, 3.14, 'Can']
```

`veriler[len(veriler):] = [9]` işlemi `veriler.append(9)` işlemiyle aynı işi yaparak listenin sonuna eleman ekler.

```
# += ile listeye değer ekleme
>>> veriler = [3,5,7]
>>> veriler += [9]
>>> veriler += [3.14]
>>> veriler += ["Can"]
>>> veriler
[3, 5, 7, 9, 3.14, 'Can']
>>> veriler += [0,1,2]
>>> veriler
[3, 5, 7, 9, 3.14, 'Can', 0, 1, 2]
>>> a = 4
>>> veriler += [a]
>>> veriler += [a+4]
>>> veriler
[3, 5, 7, 9, 3.14, 'Can', 0, 1, 2, 4, 8]
```

`append()` metodunun dışında `+="` operatörüyle diziye eleman eklenebilir. Fakat `+="` operatörü diziye kendisiyle toplayarak yeniden diziye atama yaptığı için diziye eleman eklemek `append()` metoduna göre daha yavaş olacaktır. Bu yüzden bir diziye eleman eklemenin en iyi yolu `append()` metodunu kullanmaktır.

##### 1.4.10.2. `pop()` Metodu

<pre># son elemanı listeden çıkarmak &gt;&gt;&gt; veriler [3, 5, 7, 9, 3.14, 'Can'] &gt;&gt;&gt; veriler.pop() 'Can' &gt;&gt;&gt; veriler.pop() 3.14 &gt;&gt;&gt; veriler.pop() 9 &gt;&gt;&gt; veriler [3, 5, 7]</pre>	<pre># belirli elemanı listeden çıkarmak &gt;&gt;&gt; veriler [3, 5, 7, 9, 3.14] &gt;&gt;&gt; veriler.pop(2) 7 &gt;&gt;&gt; veriler [3, 5, 9, 3.14] &gt;&gt;&gt; veriler.pop(-2) 9 &gt;&gt;&gt; veriler [3, 5, 3.14]</pre>
--	--

**pop()** metodu değer vermeden kullanılırsa listenin son indeksindeki eleman listeden atılır ve atılan eleman ekrana basılır. **pop(indeks)** metoduna indeks değeri verirse **pop()** metodu verdiğimiz değere karşılık gelen indeksteki elemanı listeden atar ve attığı elemanı ekrana basar.

```
# del ile elemanı listeden çıkarmak
>>> veriler = [3, 5, 7, 9, 3.14, 'Can']
>>> del veriler[2] # 7 siliniyor
>>> veriler
[3, 5, 9, 3.14, 'Can']
>>> veriler[1:4]
[5, 9, 3.14]
>>> del veriler[1:4] # 5, 9, 3.14 siliniyor
>>> veriler
[3, 'Can']
```

**del** fonksiyonuyla **del veriler[indeks]** şeklinde belirtilen liste elemanı listeden silinebilir. **del veriler[başlangıç:bitiş:basamak]** ifadesiyle dilimlenerek listeden eleman silmek **del** fonksiyonun **pop()** metoduna göre farklıdır. **del veriler[1:4]** ile belirtilen elemanlar dilimlenerek listeden silinir.

### 1.4.10.3. sort() Metodu ve sorted() Fonksiyonu

#### 1.4.10.3.1. sort() Metodu

```
>>> veriler = [5,7,0,1,9,2,6,4,3,8]
# kalıcı şekilde küçükten büyüğe sıralama
>>> veriler.sort()
>>> veriler
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# kalıcı şekilde büyükten küçüğe sıralama
>>> veriler.sort(reverse = True)
>>> veriler
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

**sort()** metoduyla liste içerisindeki elemanlar küçükten büyüğe doğru sıralanır. Eğer liste elemanları büyükten küçüğe doğru sıralanmak isteniyorsa **sort()** metoduna **"reverse = True"** değeri girilir. **sort(reverse = True)** ifadesi listeyi büyükten küçüğe doğru sıralar. **sort()** metoduyla liste içindeki verilerin sırası kalıcı olarak değişir.

```
>>> veriler = ["Can","Tuna","Ahmet","Mehmet"]
# kalıcı şekilde alfabetik olarak küçükten büyüğe sıralama
>>> veriler.sort()
>>> veriler
['Ahmet', 'Can', 'Mehmet', 'Tuna']
# kalıcı şekilde alfabetik olarak büyükten küçüğe sıralama
>>> veriler.sort(reverse = True)
>>> veriler
['Tuna', 'Mehmet', 'Can', 'Ahmet']
```

`sort()` metodu liste içindeki karakter değerlerini kalıcı bir şekilde alfabetik olarak sıralar.

#### 1.4.10.3.2. `sorted()` Fonksiyonu

```
>>> veriler = [5,7,0,1,9,2,6,4,3,8]
# geçici şekilde küçükten büyüğe sıralama
>>> sorted(veriler)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# geçici şekilde büyükten küçüğe sıralama
>>> sorted(veriler,reverse=True)
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
# liste verilerinin yeri değişmiyor
>>> veriler
[5, 7, 0, 1, 9, 2, 6, 4, 3, 8]
```

Liste elemanlarını kalıcı olarak sıralamak istemiyorsanız `sorted()` fonksiyonunu kullanabilirsiniz. `sorted()` fonksiyonu aldığı liste tipli veriyi sıralı bir şekilde döner.

```
# geçici şekilde alfabetik olarak küçükten büyüğe sıralama
>>> veriler = ["Can","Tuna","Ahmet","Mehmet"]
>>> sorted(veriler)
['Ahmet', 'Can', 'Mehmet', 'Tuna']
# geçici şekilde alfabetik olarak büyükten küçüğe sıralama
>>> sorted(veriler,reverse=True)
['Tuna', 'Mehmet', 'Can', 'Ahmet']
# liste verilerinin yeri değişmiyor
>>> veriler
['Can', 'Tuna', 'Ahmet', 'Mehmet']
```

`sorted()` metodu liste içindeki karakter değerlerini geçici bir şekilde alfabetik olarak sıralar.

#### 1.4.10.3.3. Karakter Dizisi Elemanlarının Alfabetik Sıralanması

Python harfe duyarlı (**case-sensitive**) bir dil olduğu için büyük ve küçük harflerin sıralanması farklı olacaktır. Büyük harflerin **ASCII** tablosundaki değerleri daha küçük olduğu için büyük harfli karakter elemanları sıralamada önde olur. Bu yüzden bir listenin karakter elemanlarını alfabetik olarak sıralamak, tüm karakterler küçük harf değilse karmaşık bir işlemdir.

```
# Liste içindeki karakter elemanları
>>> veriler = ["Abd","ABE","abF","abc"]
# geçici şekilde alfabetik olarak sıralanmıyor
>>> sorted(veriler)
['ABE', 'Abd', 'abF', 'abc']
# kalıcı şekilde alfabetik olarak sıralanmıyor
>>> veriler.sort()
>>> veriler
['ABE', 'Abd', 'abF', 'abc']
```

Örnekte görüldüğü gibi liste içerisinde ilk iki harfi büyük veya küçük **'ab'** ile başlayan karakter dizisi elemanları bulunmakta. `veriler` listesi sıralandığında karakter elemanlarının büyük harfle başlayanlarının önce, küçük harfle başlayanların sonra sıralandığı görülmekte. Bu tip bir sıralama bizim istediğimiz **'abc', 'Abd', 'ABE', 'abF'** alfabetik sıralaması değil.

---

```
# Liste içindeki karakter elemanları
>>> veriler = ["Abd","ABE","abF","abc"]
# key ile alfabetik olarak geçici sıralanıyor
>>> sorted(veriler,key=str.lower)
['abc', 'Abd', 'ABE', 'abF']
# key ile alfabetik olarak kalıcı sıralanıyor
>>> veriler.sort(key=str.lower)
>>> veriler
['abc', 'Abd', 'ABE', 'abF']
```

---

**sorted()** fonksiyonunda ve **sort()** metodunda kullanılan **key=str.lower** argümanı liste içinde sıralanacak karakter elemanlarının harflerini küçük harfe çevirir. Tüm karakter elemanları küçük harfli olduğunda alfabetik sıralama düzgün yapılır ve elemanlar orijinal halleriyle sıralı bir şekilde listelenirler.

---

```
# Liste içindeki karakter elemanları
>>> veriler = ["Abd","ABE","abF","abc"]
# key ile tersten alfabetik olarak geçici sıralanıyor
>>> sorted(veriler, key=str.lower, reverse=True)
['abF', 'ABE', 'Abd', 'abc']
# key ile tersten alfabetik olarak kalıcı sıralanıyor
>>> veriler.sort(key=str.lower, reverse=True)
>>> veriler
['abF', 'ABE', 'Abd', 'abc']
```

---

Liste içinde karakter elemanlarını tersten sıralamak istiyorsanız **sort()** metodunda ve **sorted()** fonksiyonunda **reverse=True** argümanını girmeniz gerekir.

---

```
# Liste içindeki karakter elemanları
>>> veriler = ["abc","Abd","ABE","abF"]
# karakter elemanlarının tümü küçük harfli oluşturuldu
>>> list(map(str.lower,veriler))
['abd', 'abe', 'abf', 'abc']
# geçici şekilde alfabetik olarak sıralanmıyor
>>> sorted(list(map(str.lower,veriler)))
['abc', 'abd', 'abe', 'abf']
```

---

İleriki konularda göreceğimiz liste içindeki karakter elemanlarını alfabetik sıralama kod örneği. Bu kodun diğer kodlardan farkı listedeki elemanların tümünü hem küçük harfli yapar hem de alfabetik olarak sıralar.

#### 1.4.10.4. extend() Metodu

---

```
>>> veriler = [1,3,5,7,9]
>>> liste_ek = [0,2,4,6,8]
# liste'ye liste_ek ekleniyor.
>>> veriler.extend(liste_ek)
>>> veriler
[1, 3, 5, 7, 9, 0, 2, 4, 6, 8]
# bir liste eklendiğinde liste yeniden sıralanarak düzenlenebilir.
>>> veriler.sort()
>>> veriler
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# liste'ye liste verisi ekleniyor.
>>> veriler.extend([10,11,12])
>>> veriler
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

---

**extend(liste)** metodu listeye başka bir liste eklemek için kullanılır.



```

>>> veriler = [1,3,5,7,9]
>>> liste_ek = [0,2,4,6,8]
# liste'ye liste_ek ekleniyor.
>>> veriler[len(veriler):] = liste_ek
>>> veriler
[1, 3, 5, 7, 9, 0, 2, 4, 6, 8]
# liste yeniden sıralanıyor
>>> veriler.sort()
>>> veriler
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# liste'ye liste verisi ekleniyor.
>>> veriler[len(veriler):] = [10,11,12]
>>> veriler
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

```

`veriler[len(veriler):] = [10,11,12]` işlemi `veriler.extend([10,11,12])` işlemiyle aynı işi yapar.

```

>>> veriler = ["Tuna","Can","Emel"]
>>> liste_ek = ["Oya","Ahmet"]
# liste + operatörü ile ek listeye birleştirildi.
>>> veriler = veriler + liste_ek
>>> veriler
['Tuna', 'Can', 'Emel', 'Oya', 'Ahmet']
# liste + operatörü ile ek liste verisiyle birleştirildi.
>>> veriler = veriler + ["Mehmet","Tolga"]
>>> veriler
['Tuna', 'Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet', 'Tolga']

```

Bir listeye `+` operatörüyle kendi ile toplanarak başka bir liste eklenebilir. Fakat `+` operatörüyle bir listeye liste eklemek `extend()` metoduna göre daha fazla kaynak gerektirir ve yavaştır. Bunun sebebi, `extend()` metodu listeye sadece diğer listeyi ekler ama `+` operatörü listenin kendisini ve ek listeyi birleştirerek kendisine atar.

```

>>> veriler = [1,3,5]
# + ile listeye liste dışı veri eklenemez
>>> veriler = veriler + 7
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    veriler = veriler + 7
TypeError: can only concatenate list (not "int") to list
# extend() ile listeye liste dışı veri eklenemez
>>> veriler.extend(7)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    veriler.extend(7)
TypeError: 'int' object is not iterable

```

`extend()` ve `+` operatörüyle listeye ekleme yapmak için eklenecek verinin de liste türünde olması gerekir.

#### 1.4.10.5. insert() Metodu

```

>>> veriler = [1,3,5]
# 1. indekse 14 girildi
>>> veriler.insert(1,14)
>>> veriler
[1, 14, 3, 5]
# 3. indekse 21 girildi
>>> veriler.insert(3,21)
>>> veriler
[1, 14, 3, 21, 5]

# append() ile aynı işi yapıyor
>>> veriler.insert(len(veriler),47)
>>> veriler
[1, 14, 3, 21, 5, 47]
# ilk eleman 63 girildi
>>> veriler.insert(0,63)
>>> veriler
[63, 1, 14, 3, 21, 5, 47]

```

```
>>> veriler = ["Ahmet","Tuna"]
# 0. indekse eleman ekleniyor
>>> veriler.insert(0,"Can")
>>> veriler
['Can', 'Ahmet', 'Tuna']
# append metodu gibi liste sonuna eleman ekleniyor
>>> veriler.insert(len(veriler),"Potur")
>>> veriler
['Can', 'Ahmet', 'Tuna', 'Potur']
# 2. indekse eleman ekleniyor
>>> veriler.insert(2,"Mehmet")
>>> veriler
['Can', 'Ahmet', 'Mehmet', 'Potur', 'Tuna']
```

`insert()` metodu bir elemanı listenin belli bir indeksine eklememizi sağlar. `veriler.insert(len(veriler),"Potur")` ve `veriler.append("Potur")` aynı işi yapar.

#### 1.4.10.6. remove() Metodu

```
>>> veriler = ["Can","Ahmet","Tuna","Potur","Mehmet"]
# liste elemanı remove ile listeden çıkarılıyor
>>> veriler.remove("Ahmet")
>>> veriler
['Can', 'Tuna', 'Potur', 'Mehmet']
# remove fonksiyonu değişken ile kullanılarak listeden eleman çıkartılıyor
>>> isim = "Can"
>>> veriler.remove(isim)
>>> veriler
['Tuna', 'Potur', 'Mehmet']
```

`remove(eleman)` metodu listeden belirtilen elemanın atılması için kullanılır.

```
# listede 2 üç defa kullanılmış
>>> veriler = [1,2,3,4,2,5,2]
# listede bulunan ilk '2' değeri kaldırılır
>>> veriler.remove(2)
>>> veriler
[1, 3, 4, 2, 5, 2]
# listede bulunan ilk '2' değeri kaldırılır
>>> veriler.remove(2)
>>> veriler
[1, 3, 4, 5, 2]
>>> veriler.sort()
>>> veriler
[1, 2, 3, 4, 5]
```

Eğer listede aynı elemandan birden fazla varsa ilk eleman listeden çıkarılır. Üstteki örnekte 2 sayısı listenin 1., 4. ve 6. indeks numaralarında bulunmakta. `veriler.remove(2)` metodu çalıştırılınca önce 1. indekste bulunan 2 değeri kaldırılır. Eğer `veriler.remove(2)` bir daha çalıştırılırsa 3. indekste bulunan 2 değeri kaldırılır.

```
>>> veriler = ['Tuna', 'Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet', 'Tolga']
# liste bulunmayan eleman remove ile çıkartılmaya çalışınca hata çıkar
>>> veriler.remove("Hasan")
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    veriler.remove("Eddard")
ValueError: list.remove(x): x not in list
```

Listede bulunmayan elemanı `remove()` metodu ile listeden çıkartılmaya çalışırsanız hata oluşacaktır.

---

```
>>> veriler = ['Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet']
# eleman listede var mı kontrol ediliyor
>>> if 'Tuna' in veriler :
    veriler.remove('Tuna') # eleman varsa listeden çıkar

>>> veriler
['Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet']
```

---

Koşullu durumlarda göreceğimiz **if** komutuyla ve **in** operatörüyle çıkartmak istediğiniz elemanın listede olup olmadığı kontrol edilir. Eğer eleman listede varsa listeden çıkartılır yoksa listeden çıkartma işlemi yapılmaz. Çıkartılacak elemanın listede varlığını kontrol etmek hata oluşmasını engelleyecektir. Bu şekilde işlem yapmak kodunuzun daha güvenli çalışmasını sağlayacaktır.

---

```
>>> veriler = ['Tuna', 'Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet']
# eleman listede var mı kontrol ediliyor
>>> if 'Tuna' in veriler :
    veriler.remove('Tuna') # eleman varsa listeden çıkar

>>> veriler
['Can', 'Emel', 'Oya', 'Ahmet', 'Mehmet']
```

---

**if** komutu ve **in** operatörüyle elemanın listede olduğu kontrol edilerek eleman listeden çıkartılabilir. Üstteki örnekte eleman güvenli bir şekilde listeden çıkartıldı.

#### 1.4.10.7. index() Metodu

---

```
>>> veriler = [0,1,2,3,4,5,6,7,8,9]
# 6. eleman baştan başlayarak 6.indekste
>>> veriler.index(6)
6
# 7. eleman 3. indeksten itibaren arandığından 7. indekste
>>> veriler.index(7,3)
7
```

---

**index(eleman)** metodu verilen bir değerın baştan başlayarak hangi indekste olduğunu döner. Elemanın hangi indeksten başlanarak aranacağı **index(eleman , arama başlama indeksi)** ifadesiyle yapılır. **index(7,3)** ifadesiyle arama 3. indeksten başlar ve aranan eleman bulunduğu anda indeks değeri döner.

---

```
# indeksi bulunacak eleman listenin başından başlayarak aranır
# Listede aynı elemandan birden çok varsa
# ilk bulunan elemanın indeksi döner
>>> veriler = ['Tuna', 'Can', 'Emel', 'Tuna', 'Mehmet', 'Tuna']
# arama listenin başından başladığı için ilk bulunan elemanın indeksi döner
>>> veriler.index('Tuna')
0
# arama 2. elemandan itibaren başladığı için 2. elemanın indeksi döner
>>> veriler.index('Tuna',2)
3
# arama 4. elemandan itibaren başladığı için 4. elemanın indeksi döner
>>> veriler.index('Tuna',4)
5
```

---

**index()** fonksiyonuyla liste içerisinde indeksi bulunmak istenen elemanın aranması listenin başından başlanarak yapılır. Eğer listede aynı elemandan birden çok varsa ilk bulunan listenin indeks numarası sonuç olarak döner. Üstteki örnekte yapıldığı gibi elemanın nerede aranacağı belirtilirse indeks değeri değişecektir. **'Tuna' 0., 3. ve 5.** indekslerde bulunmakta **liste.index('Tuna',2)** şeklinde yapılan aramada ilk **'Tuna'** elemanı atlanacak arama 2. indeksle başlayacağı için bulunan indeks değeri 3 çıkar. Liste içinde aramanın nereden başlanarak yapılacağını belirtmek büyük listelerde kodunuzun daha hızlı çalışmasını sağlar.

#### 1.4.10.8. count() Metodu

```
>>> veriler = ["Emel", "Emel", "Oya", "Emel", "Ahmet", "Mehmet", "Ahmet"]
# count() metodu listede elemanın kaç tane olduğunu döner
>>> veriler.count("Emel")          >>> veriler.count("Oya")
3                                  1
>>> veriler.count("Ahmet")        >>> veriler.count("Mehmet")
2                                  1
```

**count()** metodu listede aranan elemanın kaç kere kullanıldığını döner.

#### 1.4.10.9. clear() Metodu

```
>>> veriler = [0,1,2,3,4,5,6,7,8,9]
# clear metodu liste içindeki elemanları silerek listeyi boşaltır
>>> veriler.clear()
>>> veriler
[]
# clear metodu ile aynı işi yapan diğer işlemler
>>> veriler = list()          >>> veriler[:]=[]
>>> veriler = []             >>> del veriler[:]
```

**clear()** metodu listenin içindeki tüm elemanları silerek listeyi boş liste haline çevirir. **clear()** metoduyla liste boş olarak kod içerisinde varlığını sürdürür. Listelerin içleri boş liste atamasıyla da silinebilir. **liste = list()**, **liste= []**, **veriler[:]=[]** veya **del veriler[:]** işlemleriyle de listelerin içleri silinebilir.

```
# liste değişkeni tamamen siliniyor
>>> del veriler
>>> veriler
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    veriler
NameError: name 'veriler' is not defined
```

**del** fonksiyonu **del liste** şekilde kullanıldığında zaman liste değişkenini tamamen siler.

#### 1.4.10.10. copy() Metodu

```
>>> veriler = ["Ahmet","Tuna","Potur"]
# liste veriler_yeni' ye kopyalanarak iki liste oluşturuluyor
>>> veriler_yeni = veriler.copy()
>>> veriler_yeni
['Ahmet', 'Tuna', 'Potur']
>>> veriler_yeni.append("Can")
>>> veriler_yeni.append("Mehmet")
>>> veriler
['Ahmet', 'Tuna', 'Potur']
# liste_yeni listesine eklemeler yapılmış
>>> veriler_yeni
['Ahmet', 'Tuna', 'Potur', 'Can', 'Mehmet']
```

**copy()** metodu listeyi başka bir listeye kopyalar. Bilgilerin kopyalandığı listede herhangi bir değişiklik yapsanız dahi kopyalanan liste bu değişikliklerden etkilenmez. Listeleri kopyalamanın neden **copy()** metoduyla yapıldığı Nesne tabanlı programlama konusunda anlatılacak.

---

```
>>> veriler = [0,2,4,6,8]
# liste içindeki tüm bilgiler [:] ile liste_yeni'ye kopyalanıyor
>>> veriler_yeni = veriler[:]
>>> veriler_yeni
[0, 2, 4, 6, 8]
>>> veriler_yeni.extend([1,3,5,7,9])
>>> veriler
[0, 2, 4, 6, 8]
>>> veriler_yeni
[0, 2, 4, 6, 8, 1, 3, 5, 7, 9]
```

---

`[:]` operatörü kullanarak da listede bulunan elamanlar başka bir listeye kopyalanabilir.

---

```
>>> veriler = [0,2,4,6,8]
# = ile listeler bir birlerine kopyalanamaz
>>> veriler_yeni = veriler
>>> veriler_yeni.extend([1,3,5,7,9])
>>> veriler
[0, 2, 4, 6, 8, 1, 3, 5, 7, 9]
>>> veriler_yeni
[0, 2, 4, 6, 8, 1, 3, 5, 7, 9]
```

---

Eğer '=' operatörü ile değişkenlerde yapıldığı gibi bir liste başka bir değişkene atanırsa iki farklı liste oluşmaz aynı listeyi gösteren farklı isimde liste adı oluşur. Örnekte `veriler_yeni = veriler` işlemiyle `veriler_yeni` listesi `veriler` listesinin elemanlarını kullanır hale geldi. Yani `veriler` ve `veriler_yeni` aynı değerlere sahip oldu. `veriler_yeni.extend([1,3,5,7,9])` işlemiyle hem `veriler` listesine hem `veriler_yeni` listesine elemanlar eklendi. Listelerde '=' operatörüyle yapılan atamanın neden bu şekilde çalıştığı Nesne tabanlı programlama konusunda anlatılacak.

#### 1.4.10.11. reverse() Metodu

---

##### 1.4.10.11.1. reverse() Metodu

---

```
# karakter verileri
>>> veriler = ["abc", "Abd", "ABE", "abF"]
# liste tersten yerleştirildi
>>> veriler.reverse()
>>> veriler
['abF', 'ABE', 'Abd', 'abc']
# sayısal veriler
>>> veriler = [10,5,9,3,7]
# liste tersten yerleştirildi
>>> veriler.reverse()
>>> veriler
[7, 3, 9, 5, 10]
# sayısal olarak büyükten küçüğe sıralama
>>> sorted(veriler)
[3, 5, 7, 9, 10]
# sayısal olarak küçükten büyüğe sıralama
>>> sorted(veriler,reverse=True)
[10, 9, 7, 5, 3]
```

---

`reverse()` metodu liste içinde elamanları tersi şekilde sıralayarak listeye yeniden yerleştirir. Örnekte görüldüğü gibi `reverse()` metodu liste elemanlarını büyükten küçüğe doğru sıralamadı listenin içindeki yerleşim şekline göre tersten sıraladı.

##### 1.4.10.11.2. reversed() Fonksiyonu

```

>>> # karakter verileri
>>> veriler = ["abc", "Abd", "ABE", "abF"]
>>> # liste geçici olarak tersten yerleştirildi
>>> list(reversed(veriler))
['abF', 'ABE', 'Abd', 'abc']
>>> # sayısal veriler
>>> veriler = [10, 5, 9, 3, 7]
>>> # liste geçici olarak tersten yerleştirildi
>>> list(reversed(veriler))
[7, 3, 9, 5, 10]

```

Liste elemanlarını geçici olarak tersten yerleştirmek istemiyorsanız **reversed()** fonksiyonunu kullanabilirsiniz. **reversed()** fonksiyonu aldığı liste tipli veriyi elemanların yerleşimi tersten olacak şekilde döner. **reversed()** fonksiyonun kullanılabilmesi için **list()** fonksiyonun içinde **list(reversed(liste))** şeklinde kullanılması gerekir.

#### 1.4.11. Liste Verileri Üzerinde Doğrudan Çalışmak

Python nesne tabanlı bir programlama dili olduğu için Python'da her şey bir nesnedir. Nesne tabanlı programlama konusunda Python'un bu özelliğinden detaylı bir şekilde bahsedilecek. Listelerde diğer tüm veri tipleri gibi nesnel bir veri tipi olduğundan liste verilerini aynı değişkenleri kullandığımız gibi kullanabiliriz. Özellikle bu tarz kullanım şekli Karakter dizilerinde formatlama metodunda karşımıza çıkacak.

```

# liste verileri üzerinde doğrudan çalışmak
>>> [0,1,2,3,4,5,6,7,8,9][3]
3
>>> [0,1,2,3,4,5,6,7,8,9][3:8]
[3, 4, 5, 6, 7]
>>> [0,1,2,3,4,5,6,7,8,9][3:8:2]
[3, 5, 7]
>>> [1,2,2,3,3,3,4,4,4,4].count(3)
3
>>> [0,1,2,3,4,5,6,7,8,9].pop()
9
>>> [0,1,2,3,4,5,6,7,8,9].index(5)
5
>>> ['Tuna', 'Can', ['Emel', 'Oya', 'Ahmet'], 39, 3.14][3:]
[39, 3.14]
>>> ['Tuna', 'Can', ['Emel', 'Oya', 'Ahmet'], 39, 3.14][2][1]
'Oya'
>>> ['Tuna', 'Can', ['Emel', 'Oya', 'Ahmet'], 39, 3.14].index('Can')
1
# liste verisi içinde bulunan veri indeksle bir değişkene atanıyor
>>> veriler = ['Tuna', 'Can', ['Emel', 'Oya', 'Ahmet'], 39, 3.14][2]
>>> veriler
['Emel', 'Oya', 'Ahmet']

```

Örnekte görüldüğü gibi liste verisi aynı liste tipi değişkenler gibi kullanılabilir. Fakat bu tarz kullanım çok dezavantajlı olduğu için tercih edilmemelidir.

#### 1.4.12. Liste İşlemleri

```

>>> liste_1 = ['a','b','c','d']
>>> liste_2 = ['e','f','g','h']
# listeler toplanabilir
>>> liste_1 + liste_2
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

```

Listeler birbiriyle toplanabilir. İki liste toplandığı zaman ikinci liste birinci listeye eklenir ve sonuç liste olarak çıkar.

```
# listeler sadece listelerle toplanabilir
>>> ['a','b','c','d'] + 'e'
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    ['a','b','c','d'] + 'e'
TypeError: can only concatenate list (not "str") to list
```

Toplama işlemini listelerde kullanabilmek için her iki değerinde liste olması gerekir.

```
>>> veriler = ['a','b','c','d']
# iki liste toplanarak bir listeye atanabilir
>>> liste_t = veriler + [1,2,3,4]
>>> liste_t
['a', 'b', 'c', 'd', 1, 2, 3, 4]
```

İki liste toplandığında listeler birbirine eklenir. Oluşan liste başka bir listeye eklenerek yeni bir liste oluşturur.

```
>>> liste = ['a','b','c','d']
# liste elemanları çarpılarak tekrar ettirilebilir
>>> liste * 3
['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']
```

Liste çarpılarak tekrar ettirilebilir. Sonuç tekrar ettirilerek oluşan yeni bir listedir.

```
>>> veriler = ['a','b','c','d']
# listenin içeriğini değiştirmek istiyorsanız işlemi listeye atamalısınız.
>>> veriler
['a', 'b', 'c', 'd']
>>> veriler = veriler * 3
>>> veriler
['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']
```

Listeler ile yaptığınız işlemlerin sonuçlarını bir listeye atamazsanız çalıştığınız listenin değeri değişmeyecektir. Eğer çarpma işlemi gibi bir işlem yapıp listenin değerini çıkan sonuca göre değiştirmek istiyorsanız, yaptığınız işlemi kullandığınız listeye atamalısınız. Önce işlem sonra atama yapılarak listenin değeri işlemin sonucuyla değişecektir.

```
>>> veriler = list(range(6))
>>> veriler
[0, 1, 2, 3, 4, 5]
# veriler listesinin ilk elemanı diğer elemanlar ile hesaplanarak değişiyor
>>> veriler[0] = (veriler[2] + veriler[4])*3
>>> veriler
[18, 1, 2, 3, 4, 5]
# veriler listesinin 4. Elemanı değişken ile yapılan hesaplama ile değişiyor
>>> x = 3
>>> veriler[4] = (x + 7)*5
>>> veriler
[18, 1, 2, 3, 50, 5]
```

Listelerin elemanları hesaplama yapılarak değiştirilebilir. Hatta listenin bir elemanı listenin diğer elemanları kullanılarak yapılan bir işlemin sonucuyla da değiştirilebilir.

```
# değişkenlere sırayla değerleri atanıyor
>>> a,b,c,d,e = 5,3.14,"Can","39",1981
# değişkenler kullanılarak veriler oluşturuluyor
>>> veriler = [a,b,c,d,e]
# veriler değişkenlerin değeriyle oluşturuluyor
>>> veriler
[5, 3.14, 'Can', '39', 1981]
>>> a
5
# değişkenin değerini değiştirmek listenin elemanlarını değiştirmez
>>> a = 19
>>> veriler
[5, 3.14, 'Can', '39', 1981]
```

Değişkenlerin taşıdıkları değerlerden bir liste oluşturmak mümkündür. Değişkenler ile liste oluşturulduğunda değişkenler sadece listeyi oluşturmak için kullanılır. Sonrasında değişkenlerin değerleri değişse bile listenin verileri değişmeyecektir. Yani değişken değerleriyle oluşan liste ve değişkenler ayrı verilerdir.

```
# karakter dizisi elemanları metotları kullanılabilir
>>> veriler = ['tuna', 'can', 'EMEL']
>>> veriler[0].title()
'Tuna'
>>> veriler[1].upper()
'CAN'
>>> veriler[2].lower()
'emel'
>>> veriler[2].title()
'Emel'
```

Liste içerisinde karakter dizisi olan elemanlar varsa karakter dizisi elemanları üzerinde karakter dizisi elemanlarına ait metotlar kullanılabilir. Liste içerisindeki karakter dizilerinin metotlarının kullanılabilmesi için karakter dizisi elemanlarının birer birer çağrılması gerekir. **title()** metodu karakterlerin ilk harfini büyük yapar, **lower()** karakterlerin tümünü küçük harf yapar ve **upper()** karakterlerin tüm harflerini büyük harf yapar. Karakter dizilerinin metotları ileride daha detaylı incelenecek.

```
# dizi elemanlarını karakterler ile birleştirmek
>>> veriler
['tuna', 'can', 'EMEL']
>>> mesaj = "Ailenin en küçük çocuğu " + veriler[0].title() + "."
>>> mesaj
'Ailenin en küçük çocuğu Tuna.'
```

Liste içerisinde bulunan karakter elemanı listeden metot kullanarak çekilerek başka karakter dizileriyle birleştirilebilir. Örnekte **veriler[0].title()** ifadesiyle **'tuna'** elemanı baş harfi büyük harf yapılarak (**'Tuna'**) öndeki cümleyle birleştirilmiş.

## 1.5. Demetler (tuple) Veri Tipi

## 1.6. Karakter Dizileri (string) Veri Tipi

Karakter verisi ile karakter dizisi aynı anlamı taşır bu bölümden sonra karakter verileri için **Karakter Dizisi** tanımını kullanacağız. **Karakter verilerini(string)** daha önce basit bir şekilde görmüştük bu bölümde karakter dizilerini detaylı bir şekilde işleyeceğiz.



Karakter dizileri yapıları gereği listelere oldukça benzerler. Karakter dizileri tıpkı listeler gibi, indekslenirler, parçalanırlar ve üzerinde değişik işlemler yapabildiğimiz fonksiyonları barındırırlar. Ancak karakter dizilerinin listelerden önemli farkları bulunmaktadır. Karakter dizileri değiştirilemez bir veri tipidir.

### 1.6.1. Karakter Dizilerini Oluşturma

```
>>> adı = 'Ahmet Tuna POTUR'
>>> print(adı)
Ahmet Tuna POTUR
```

Bir değişkene eşittir(=) operatörü ile değer atadığımızda değişken atadığımız değerın tipinde oluşur. **adı** değişkenine tırnak(' ') içinde 'Ahmet Tuna POTUR' metni yazarak **string** değeri atadığımızda karakter dizisi oluşturduk. **adı** değişkeni içinde bulunan karakter dizisi **print()** fonksiyonuna girildiği için **adı** değişkeni içinde bulunan karakter verisi ekrana yazdırıldı.

```
>>> #Tek tırnak ile      >>> #Çift Tırnak ile      >>> #Üç Tırnak ile
>>> print('Tuna POTUR') >>> print("Tuna POTUR") >>> print("""Tuna POTUR""")
Tuna POTUR             Tuna POTUR             Tuna POTUR
```

Tırnak içerisinde yazılan her ifade karakter verisi olarak algılanır. Python’da **string** oluşturmak için **çift tırnak("")**, **tek tırnak(' ')** veya **üç tırnak (""" """)** sembolleri kullanılır. Üstteki kodlarda gördüğünüz gibi ekrana yazdırılan metinlerin hiçbir farkı yok.

```
>>> print("Ahmet Tuna POTUR")
Ahmet Tuna POTUR
```

**print()** fonksiyonun içine değişken kullanılmadan doğrudan değer girişi yapılabilir. Burada “Ahmet Tuna POTUR” karakter verisi değişken kullanılmadan doğrudan ekrana yazdırıldı.

```
>>> print('Lüleburgaz')
```

**SyntaxError: EOL while scanning string literal**

Karakter dizini oluştururken hangi tırnak işaretini kullandıysanız karakter dizisini o tırnakla bitirmelisiniz. Eğer karakter dizinin başındaki ve sonundaki tırnak farklıysa hata oluşacaktır.

```
>>> print('Can'ın 15. yaş günü')
```

**SyntaxError: invalid syntax**

**tek tırnak(' ')** ile oluşturulan bir karakter verisi kullanıyorsanız içinde **Can'ın** metnindeki gibi bir tırnak kullanamazsınız. Python karakter dizisini oluşturmaya başladığında önce **tek tırnak(' ')** ile başlanan karakteri okur ve karakter dizisi oluşturulacağını anlar. **İkinci tek tırnak** geldiğinde karakter dizisinin bittiğini düşünür. Eğer **İkinci tek tırnaktan** sonra yine bir tırnak daha okunursa Python burada ne yapılmak istendiğini anlamayacak ve hata verecektir.

```
>>> print("Can'ın 15. yaş günü")
Can'ın 15. yaş günü
```

Eğer karakter dizisi içerisinde **Can'ın** metnindeki gibi **tek tırnak(' ')** kullanmak istiyorsanız oluşturmak istediğiniz karakter dizisini **çift tırnak("")** içerisinde oluşturmalsınız.

```
>>> print('Bugün öğrenciler "Dostluk" adlı şiiri incelediler.')
>>> print("Bugün öğrenciler "Dostluk" adlı şiiri incelediler.")
Bugün öğrenciler "Dostluk" adlı şiiri incelediler.
```

İçinde **çift tırnak**("") içeren bir karakter dizisi oluşturmak için üstte gösterilen iki tanımlamada düzgün çalışacaktır ve aynı çıktıyı verecektir. **tek tırnak**(' ') içinde bulunan **çift tırnak**("") karakterleri tırnak tipleri farklı olduğu için kullanılabilir.

```
>>> print("""üç tırnak
birden çok satırlı
metinler için kullanılır""")
üç tırnak
birden çok satırlı
metinler için kullanılır
```

**üç tırnak** (""" """) birden fazla satıra yayılmış karakter dizinlerini tanımlamak için kullanılır.

```
>>> print("""
[A]=====PYTHON=====[-] [o] [x]
|
|           Merhaba Dünya
|           Penceresine
|           Hoşgeldiniz
|
|
|=====
""")

[A]=====PYTHON=====[-] [o] [x]
|
|           Merhaba Dünya
|           Penceresine
|           Hoşgeldiniz
|
|
|=====
```

Üsteki gibi bir çıktı vermek istiyorsanız **üç tırnak** (""" """) kullanmanız gerekir.

```
>>> ada_lovelace="""Ada Lovelace sahip olduğu maddi kaynak sayesinde
Babbage'in en büyük destekçilerinden biriydi.
Kendisi de matematikçi olan Ada Analitik Makine üzerine yazdığı notlarda
Bernouli sayılarını hesaplamak için Analitik Motorun Algoritmasını
tanımlıyordu. İlk algoritmayı tasarladığı için
Ada ilk bilgisayar programcısı olarak kabul edilir.
"""
>>> print(ada_lovelace)
Ada Lovelace sahip olduğu maddi kaynak sayesinde
Babbage'in en büyük destekçilerinden biriydi.
Kendisi de matematikçi olan Ada Analitik Makine üzerine yazdığı notlarda
Bernouli sayılarını hesaplamak için Analitik Motorun Algoritmasını
tanımlıyordu. İlk algoritmayı tasarladığı için
Ada ilk bilgisayar programcısı olarak kabul edilir.
```

İsterseniz **üç tırnak** (""" """) ile yazdığınız karakter dizisini bir değişkene atayabilirsiniz.

### 1.6.2. Özel Karakterler

### 1.6.3. Formatlama

## 1.7. print() Fonksiyonu Hakkında Detaylı Bilgi

`print()` fonksiyonunu daha önce basit bir şekilde görmüştük. Bu bölümde `print()` fonksiyonunun görmediğimiz özelliklerini görerek eksik özelliklerini öğrenelim. `print()` fonksiyonunun tanımını yeniden yapmakta yarar var. `print()` fonksiyonu parantez içine girilen değeri veya değerleri ekrana yazdırılır.

## 1.8. Sözlük Veri Tipi

## 1.9. Kümeler (Sets) Veri Tipi

## 2. Kullanışlı Python Fonksiyonları

2.1. `dir()` Fonksiyonu

2.2. `help()` Fonksiyonu

2.3. `del()` Fonksiyonu

2.4. `abs()` Fonksiyonu

2.5. `round()` Fonksiyonu

2.6. `max()` ve `min()` Fonksiyonu

2.7. `sum()` Fonksiyonu

`sum()` fonksiyonu verilen değerleri toplayarak döndürür. Değerlerin liste, demet vb. şeklinde verilmesi gerekir.

## 3. Koşul Durumları