Numpy (Numerical Python)

Numpy Bilimsel hesaplama işlemleri kolaylaştırmak için yazılmış olan bir python kütüphanesidir. Python'un yetersiz kaldığı bazı sayısal işlemlerde ihtiyaçları gidermek için yazılmış bir kütüphanedir. Veri madenciliği, makine öğrenmesi ve veri analizi konusunda çok faydalı bir modüldür. Hesaplama işlemleri ve dönüşüm işlemlerinde sıkça kullanılan kod yapısı numpy kütüphanesi ile basit bir seviyede ve az kod yazacak şekilde tasarlanmıştır. Matematiksel işlemleri çok hızlı yapmasından dolayı sıkça kullanılan bir kütüphanedir.

Özellikleri:
Bilimsel hesaplamalar için kullanılır.
Numpy array yapısı, listelere göre oldukça az yer kaplar.
Array'lerin elemanlarına erişmek listelere erişmekten daha hızlıdır.
Arraylar/ Çok boyutlu arraylar ve matrisler üzerinde yüksek performanslı çalışma imkanı sağlar.
Temelleri 1995 yılında(matrix-sig, Guido van rossum) atılmış geliştirmelerle beraber 2005(Travis Oliphant) yılında tamamlanmıştır.
Listelere benzemektedir ancak verimli veri saklama ve vektörel operasyon gibi özellikleri vardır.
Listelerin depolama biçimi çok yer kapladığı ve yavaş çalıştığı için numpy kullanmak daha avantajlıdır.
Tek boyutlu diziler vektör, iki boyutlu diziler matris olarak ifade edilir.
Numpy arraylerinin kendine özgü metodları olduğu için python listelerinde kendimizin yazması gereken metotlar numpy ile hazır halde gelmektedir.
Numpy dizilerinin kendi veri tipleri bulunmaktadır. Bunlara ndarray olarak geçer. Numpy kütüphanesi kullanmak için jupyter notebook ya da spyder editörünü kullanabiliriz.
Eğer farklı bir platform kullanacaksanız numpy kütüphanesinin yüklenmesi gerekmektedir. Bunun için komut isteminde **pip install numpy** komutu yazılmalıdır.
Kütüphanenin yüklendikten sonra python'a dahil edilmesi için
import numpy as np

komutu kullanılmalıdır. Şayet kütüphane oluşturulmadan numpy komutları kullanılmaya çalışılırsa python hata verir. Jupyter üzerinde çalışırken sayfanın en üstünde kütüphaneyi dahil etme işlemi yaptığımızda sonraki satırlarda tekrar **import numpy as np** komutunu çalıştırmamıza gerek yoktur. Ancak jupyter'i kapatıp tekrar açtığımızda bu kütüphanenin tekrar yüklenmesi gerekmektedir. Aşağıdaki örneklerde tekrar tekrar kütüphane dahil edilmeyecektir.

Şimdi normal bir liste ve bir numpy dizisi oluşturup aradaki farkı inceleyelim.

```
a=[1,2,3,4]
b=[5,6,7,8]
liste=list()
for i in range(len(a)):
    liste.append(a[i]*b[i])
print(liste)
```

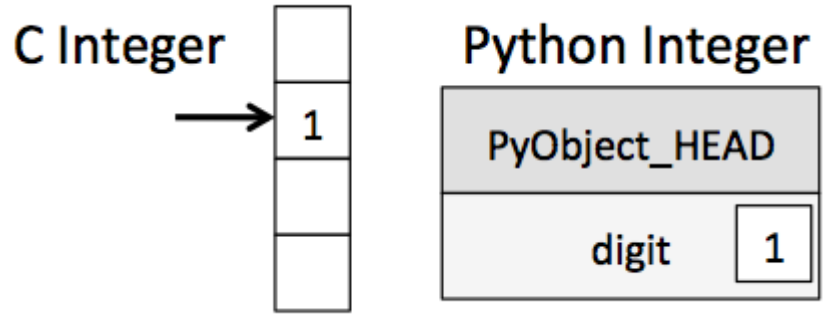
[5, 12, 21, 32]

Aynı işlemi numpy dizileri ile yapmak istersek:

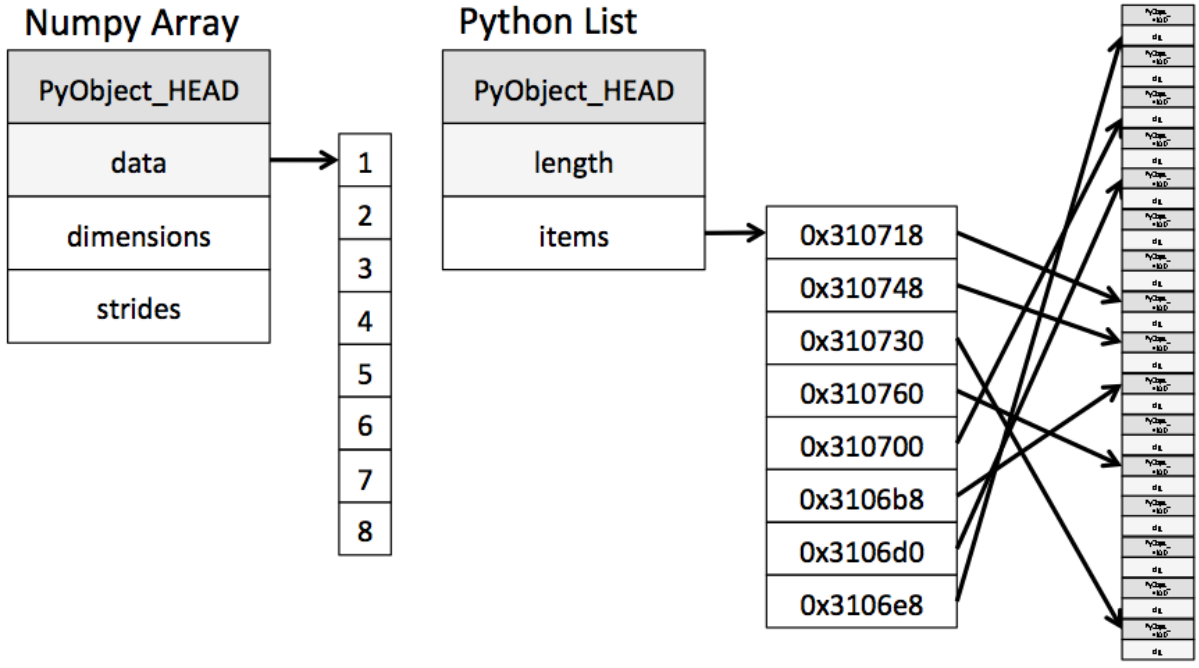
```
import numpy as np
a=np.array([1,2,3,4])
b=([5,6,7,8])
print(a*b)
```

[5 12 21 32]

Numpy kütüphanesi bize matematiksel ve istatistiksel işlemlerde döngülerden arınmış işlem yapmamızı sağlar.



Resim 1
C liste elemanının bellekte kapladığı alan ve python listesinin kapladığı alan
Bir python listesi içerisinde farklı veri tiplerinden değerler bulunabilir
liste=[5,9,"asd",7.8]
yukarıdaki gibi bir tanımlamada liste içerisinde iki adet integer değer, bir adet karakter dizisi ve bir adet float ifade bulunmaktadır. Burada her bir eleman kendi tip bilgisini, referans bilgisini barındırmak durumundadır. Bütün değişkenler aynı olsa bile her bir eleman tip ve referans bilgisini tutmak durumundadır.
Bir numpy array ile python list karşılaştırırsak.



Resim 2

Numpy Array Oluşturma İşlemleri

Numpy dizilerine Array denilmektedir. Bir python listesi oluşturmak için
liste=[1,2,3,4]
liste
[1, 2, 3, 4]
Şeklinde bir yapı kullanılmaktadır.
Şimdi bir tane array oluşturalım
import numpy as np

a=np.array([1,2,3,4])
a
array([1, 2, 3, 4])
buradaki yapı incelenirse önce numpy kütüphanesi programa dahil edilmiş **as np** komutu ile numpy kütüphanesinin kullanımı kolaylaşmıştır. Yani bu sayede numpy listesi oluştururken
a=numpy.array([1,2,3,4])
şeklinde bir kullanım yerine basit ve kısaca **a=np.array([1,2,3,4])** şeklinde bir kullanım yapabileceğiz.
array() fonksiyonu bir numpy array oluşturmamızı sağlar.
Numpy array'leri oluştururken çift parantez ya da parantez ve köşeli parantez beraber kullanılmalıdır
a=np.array([1,2,3,4])
a=np.array((1,2,3,4))
her iki kullanım da doğrudur. Ancak tek parantez kullanılırsa python hata verir.
a=np.array(1,2,3,4)

```
a=np.array(1,2,3,4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-26-46868a44e74f> in <module>
----> 1 a=np.array(1,2,3,4)

ValueError: only 2 non-keyword arguments accepted
```

Resim 6
Şimdi oluşturduğumuz array'in veri tipine bakalım.
type(a)
numpy.ndarray
boş bir numpy array oluşturmak istersek
b=np.array([])
b
array([], dtype=float64)
şeklinde bir kullanım yapabiliriz.

Liste veya Tuple'lardan Array Oluşturmak

Elimizde bulunan liste ve tuple'lardan da kolaylıkla array oluşturabiliriz. Listelerden array oluşturmak için,
a=[1,2,3,4,5,6]
np.array(a)
array([1, 2, 3, 4, 5, 6])
tuple'lardan array oluşturmak için
b=[10,11,12,13,14]
np.array(b)
array([10, 11, 12, 13, 14])

hatırlanırsa önceki bölümde array oluşturmak için iç içe parantez ve köşeli parantez kullanıyorduk. Oy saki liste ve tuple'lardan array oluştururken tek parantez işimizi görmektedir. Bunun nedeni liste ve t uple'lerin yapısı itibariyle parantezler içermesidir. İncelemek istersek
a=[1,2,3,4,5,6]
a
[1, 2, 3, 4, 5, 6]

Görüldüğü üzere program çıktısında köşeli parantez bulunmaktadır.

Numpy Array'lerinin Veri Tipleri

Hatırlanacağı üzere bölümün başında numpy array'lerinin tüm elemanlarının aynı veri tipinde olacağından bahsetmiştik. Örnek olarak

```
a=np.array([1,2,3,4])
a
array([1, 2, 3, 4])
```

şeklinde bir kullanımda dizinin tüm elemanlarının int veri tipinden olduğu görülmektedir. Bu söylediğimiz işlemi doğrulamak için

```
for i in a:
    print(type(i))
<class 'numpy.int32'>
<class 'numpy.int32'>
<class 'numpy.int32'>
<class 'numpy.int32'>
```

Şeklinde bir kullanım ile her bir elemanın veri tipinin int veri tipinden olduğunu görmüş olduk.
Yine içerisinde float değerlerden olan bir liste oluşturursak

```
a=np.array([3.14,5.6,7.8,8.2])
for i in a:
    print(type(i))
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

Tüm değerlerin float veri tipinden olduğunu göreceğiz.
Bu durumda numpy arraylarinin tüm verilerinin aynı veri tipinde olmadığı durumlarda ise ne gibi bir çıktı alabiliriz? Eğer array içerisinde bir adet float veri tipinden veri olsa dahi tüm elemanlar float veri tipinden olacaktır.

```
a=np.array([3,4,5.2,7.2,9])
a
array([3. , 4. , 5.2, 7.2, 9. ])
for i in a:
    print(type(i))
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

Görüldüğü üzere tüm değişkenler aynı veri tipine dönüştürülmüştür.
Dilersek farklı veri tipinden değerlere sahip bir array'i istediğimiz veri tipinden oluşturabiliriz.

```
a=np.array([3,4,5.2,7.2,9],dtype="int32")
a
array([3, 4, 5, 7, 9])
```

görüldüğü üzere dizinin tüm elemanları integer veri tipine dönüştürüldü. Burada int32, int64 gibi farklı kullanımlar görülmektedir. Bunlar değişkenlerin bellekte kapladıkları alanı ifade etmektedir.

Eğer varolan bir array'in veri tipini dönüştürmek istersek

a.astype(np.float32)

```
array([3., 4., 5., 7., 9.], dtype=float32)
```

Sıfırlardan(0) Oluşan Array Oluşturma İşlemi:

Numpy kütüphanesinde zeros() fonksiyonu kullanılarak sıfırlardan oluşan array'lar oluşturulabilir.

np.zeros(10)

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

zeros() fonksiyonuna girdiğimiz parametre ile array'in kaç elemandan oluşacağını belirleriz. Burada dizi elemanlarımız varsayılan olarak float veri tipinde oluşturulmuştur. Dilersek zeros() fonksiyonuna vereceğimiz **dtype="int32"** parametresi ile listenin elemanlarının veri tipini değiştirebiliriz.

np.zeros(10,dtype="int32")

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

Birlerden Oluşan Array Oluşturma İşlemi:

Numpy kütüphanesinde ones() fonksiyonu kullanılarak birlerden oluşan array'lar oluşturulabilir.

np.ones(10)

array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])

aynı şekilde zeros() fonksiyonunda olduğu gibi verdiğimiz parametre ile array'in kaç elemanlı olacağını belirleyebilir, dtype() fonksiyonu ile array'in elemanlarının veri tipinin ne olacağını belirleyebiliriz.

Sabit Bir Sayıdan Array Oluşturma İşlemi:

zeros() ve ones() fonksiyonları ile sadece sıfır ve birlerden oluşan değerler üretebiliyorduk. Farklı sayılardan oluşan arraylar için full() fonksiyonu kullanılabilir. Ancak full() fonksiyonu içerisine iki adet parametre eklemek gerekmektedir. Birinci parametre değerin kaç defa tekrarlanacağını ikinci parametre ise hangi değerin tekrarlanacağını belirtmektedir.

np.full(8,5)

array([5, 5, 5, 5, 5, 5, 5, 5])

full() fonksiyonundan ayrı olarak istediğimiz sayılardan oluşan bir array oluştururken bir yöntem daha kullanabiliriz. Hatırlanacağı üzere ones() fonksiyonu ile istediğimiz uzunlukta değerleri bir olan array'lar oluşturabiliyorduk. Bu şekilde bir array oluşturarak oluşturmak istediğimiz değerle array'e çarpma işlemi uygulayabiliriz.

np.ones(7)*4

array([4., 4., 4., 4., 4., 4., 4.])

görüldüğü üzere full fonksiyonu ile aynı işlemi yapmış olduk.

Matris(iki boyutlu array) Oluşturma İşlemi

Buraya kadarki bölümde örneklerimizi hep tek boyutlu diziler üzerinde yaptık. Tek boyutlu diziler vektör olarak ifade edilmektedir. Matrisler ise iki boyutlu dizilere denilmektedir.

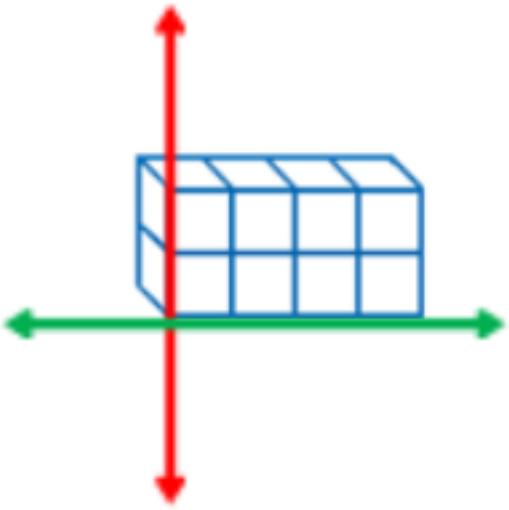
np.array([[1,2,3],[4,5,6]])

```
array([[1, 2, 3],
       [4, 5, 6]])
```

Diziler verileri kümeler halinde tutabilmemizi sağlar. Tek boyutlu, iki boyutlu, üç boyutlu hatta çok daha fazla boyuta sahip diziler olabilir.

Tek boyutlu dizi

İki boyutlu dizi



Biz veri analizinde sıklıkla tek ve iki boyutlu dizilerden faydalanacağız. Şimdi ones() fonksiyonu ile 3x3 lük bir matris oluşturalım.

np.ones((3,3))

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

Matris oluştururken kullandığımız fonksiyonlar iki adet parametre almaktadır. Yukarıdaki örnekte 3x3 lük matris oluşturduk. Burada parantez içinde kullandığımız parametrelerden birincisi satır ikincisi ise sütunu ifade etmektedir. Başka bir örnekle açıklamak istersek:

np.ones((2,4))

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

Görüldüğü üzere 2 satır 4 sütunluk bir matris oluşturulmuştur.

arange() Fonksiyonu

array'imizin elemanlarını birbirinden farklı değerlerden oluşturmak, yani doğrusal bir dizi oluşturmak isteyebiliriz. Ya da değerler 2'şerli ya da 3'erli olarak artsın istersek arange() fonksiyonunu kullanabiliriz.

np.arange(0,10)

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

burada arange() fonksiyonu sayesinde belirlediğimiz başlangıç ve bitiş değerlerine göre biz array oluşturduk. Aynı python'daki range() fonksiyonu gibi başlangıç değerini(0) aldı ancak bitiş değerini(10) almadı.

Aynı şekilde oluşturacağımız array'in değerlerini atlamalı olarak da oluşturabiliriz.

np.arange(20,30,2)

array([20, 22, 24, 26, 28])

değerlerimizi 20 den 30 a kadar 2'şer atlamaları olarak oluşturduk.

Arange fonksiyonu ile float ifadelerden oluşan bir array'de oluşturulabilir.

np.arange(0,3,0.5)

```
array([0. , 0.5, 1. , 1.5, 2. , 2.5])
```

burada da 0 ile 3 arası 0.5 artan değerlerden oluşan bir array oluşturduk.

linspace() Fonksiyonu (lineer / Eşit aralıklarla ilerleyen)

iki değer arasında belirli sayıda değer oluşturabiliriz. Bunun için linspace() fonksiyonunu kullanmalıyız.

np.linspace(0,15,10)

```
array([ 0.        ,  1.66666667,  3.33333333,  5.        ,  6.66666667,
        8.33333333, 10.        , 11.66666667, 13.33333333, 15.        ])
```

Burada ise başlangıç(0) ve bitiş(15) değerleri arasında 10 adet sayı oluşturuldu. Ancak burada oluşturulan değerler dağılımlara göre oluşturulmuştur.

np.linspace(0,100,5)

```
array([  0.,  25.,  50.,  75., 100.])
```

0 ile 100 arasındaki değerler 5 eşit parçaya bölünmüştür.

random() Fonksiyonu

random fonksiyonunun çok sayıda metodu bulunmaktadır.

random.rand() metodu:

0-1 arasında verdiğimiz parametre kadar değer üretir.

np.random.rand(5)

```
array([0.65512923, 0.83644559, 0.86527025, 0.17550102, 0.28205988])
```

5 adet değer üretildi

np.random.rand(2,3)

```
array([[0.51889336, 0.81748053, 0.48182594],
       [0.60410118, 0.15396498, 0.95289424]])
```

2X3 lük bir matris oluşturuldu.

random.normal() metodu:

Ortalaması 0 varyansı 1 olan 3x3 lük matris oluşturmak için,

np.random.normal(0,10,(3,3))

```
array([[-0.37367756, -0.21996554,  0.69922528],
       [-0.17753236,  0.98836675, -0.20855285],
       [ 0.95728725,  0.3078482 ,  0.57601348]])
```

Ya da

np.random.normal(0,10,size=(3,3))

```
array([[-18.31512911,   0.29995969,  10.23188918],
       [ -3.18959774,  12.70365179,  -3.04027827],
       [  5.04477049,   1.70456876,  -7.90966551]])
```

Şimdi de ortalaması 10 standart sapması 3 olan bir matris oluşturalım.

```
np.random.normal(10,3,(3,3))
array([[14.42962125,  9.81662023, 11.70311033],
       [10.63579681,  7.32817475,  8.2686081 ],
       [ 9.02782419, 14.17611246,  9.23203997]])
```

random.randint() metodu:

Bu metod ile girilen değerler arasında istenilen sayı kadar rasgele integer sayı üretmemizi sağlar. 0-10 arasında rasgele bir sayı oluşturmak istersek

np.random.randint(0,10)

```
8
```

Şeklinde bir kullanım yapabiliriz.

Eğer random fonksiyonunu kullanırken tek parametre verirsek

np.random.randint(10)

```
2
```

Başlangıç değeri 0 verilmiş gibi değer üretir.

Ya da 3. Bir parametre girilerek kaç tane sayı oluşturabileceğimizi belirleyebiliriz.

np.random.randint(0,10,8)

```
array([6, 6, 8, 6, 4, 6, 6, 3])
```

8 elemanlı bir vektör(tek boyutlu dizi) oluşturduk.

3X3 lük bir matris oluşturmak istersek

np.random.randint(0,10,(3,3))

```
array([[4, 4, 4],
       [9, 3, 2],
       [0, 0, 5]])
```

Ya da

np.random.randint(0,10,size=(3,3))

```
array([[6, 9, 4],
       [5, 3, 7],
       [5, 4, 3]])
```

random.randn() metodu:

normal dağılımlı rasgele değerlerden oluşan bir array oluşturmak için kullanılır.

np.random.randn(3,4)

```
array([[ 0.53203054,  0.84466037,  1.65647989,  0.32960669],
       [ 2.90905041, -0.94299365, -0.50252075,  1.07493087],
       [ 0.18271029, -1.67103378,  0.86828259, -0.53017293]])
```

Eye() Fonksiyonu:

Birim matris oluşturmak için kullanılır. Satır ve sütunu aynı olan matrislerde köşegenlerin 1 olmasını sağlar.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Resim 7

Resimdeki gibi bir matris oluşturmak için kullanılır.

np.eye(4)

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Biçimlendirme İşlemleri

Numpy array'leri üzerinde biçimlendirme işlemleri yapılabilir.

ndim: boyut sayısı

shape: boyut bilgisi

size: eleman sayısı

dtype: veri tipi

a=np.random.randint(1,10,size=10)

a

```
array([3, 9, 2, 3, 8, 5, 4, 1, 8, 5])
```

şeklinde bir dizi oluşturarak üzerinde biçimlendirme özelliklerine bakabiliriz.

a.ndim

```
1
```

Tek boyutlu bir dizi olduğunu belirtti

a.shape

```
(10,)
```

Burada boyutlarını gösterme işlemi yaptı, dizimizin tek boyutu ve bu tek boyutta 10 elemanı bulunmaktadır.

a.size

```
10
```

Array üzerinde toplam 10 adet eleman bulunmaktadır.

a.dtype

```
dtype('int32')
```

dizinin içindeki elemanların veri tipini verir.

Aynı işlemleri matrisler için yapmak istersek

a=np.random.randint(1,10,size=(3,3))

a

```
array([[7, 3, 3],
       [2, 8, 7],
       [6, 6, 3]])
```

a.ndim

```
2
```

2 boyutlu dizi olduğunu belirtir.

a.shape

```
(3, 3)
```

Sahip olduğu boyutlar ve her bir boyutun ayrı ayrı eleman sayısını gösterdi

a.size

```
9
```

Toplam eleman sayısını gösterdi

a.dtype

```
dtype('int32')
```

dizinin içindeki elemanların veri tipini verir.

3 Boyutlu Array'ler İle Çalışmak:

3 boyutlu array oluşturmak için size fonksiyonuna 3. Bir parametre daha girmek gerekmektedir.

a=np.random.randint(0,10,size=(2,4,3))

a

```
array([[[2, 4, 8],
        [6, 1, 7],
        [1, 6, 4],
        [2, 4, 5]],

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

Burada size fonksiyonuna girdiğimiz 1. Parametre array'in kaç boyutlu olduğunu, ikinci parametre satır sayısını ve 3. Parametre ise sütun sayısını belirtmektedir.

```
a.ndim
```

```
3
```

3 boyutlu dizi olduğunu belirtir.

```
a.shape
```

```
(2, 4, 3)
```

Sahip olduğu boyutları ve her boyutun eleman eleman sayısını verir.

a.size

```
24
```

Eleman sayısına ulaştık

a.dtype

```
dtype('int32')
```

Array'lerde Yeniden biçimlendirme(reshape ve newaxis) İşlemleri:

Bezen elimizde bulunan tek boyutlu array'leri matrislere dönüştürmek isteyebiliriz. Bu işlem için reshape() fonksiyonu kullanılmalıdır. Bir adet tek boyutlu array oluşturarak bunu matrise dönüştürelim

a=np.arange(1,10).reshape(3,3)

a

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Ya da

a=np.arange(1,10)

a.reshape(3,3)

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Şimdi ise randint fonksiyonu ile oluşturduğumuz array yapısına reshape işlemi yapalım.

a=np.random.randint(1,10,9).reshape(3,3)

a

```
array([[1, 6, 1],
       [2, 4, 4],
       [1, 1, 5]])
```

Dönüştürme işlemleri yaparken çok sık hata ile karşılaşılmaktadır. Her iki örneğimizde de 9 elemanlı vektörümüz matrise dönüştürülmüştür. Ancak eleman sayısı 9 dan farklı olsaydı 3X3 lük matris oluştururken hata ile karşılaşacaktık.

a=np.arange(1,9).reshape(3,3)

a

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-69-7f26b954198d> in <module>
----> 1 a=np.arange(1,9).reshape(3,3)
      2 a

ValueError: cannot reshape array of size 8 into shape (3,3)
```

Resim 7

Görüldüğü üzere boyut uyuşmazlığı olduğu için dönüştürme işlemi yaparken hata ile karşılaştık. Burada 9 değeri hesaba katılmadığı için 0-8 arası array oluşturulmuştur.

Numpy array'lerinde en çok karıştırılan konu dönüştürme işlemleridir. Şimdi tek boyutlu bir array oluşturarak bunu matrise dönüştürme işlemi yapalım.

x=np.array([1,2,3])

y=x.reshape(1,3) # 1 satır 3 sütunluk matrise dönüştür

y

```
array([[1, 2, 3]])
```

oluşan array'in boyut sayısını inceleyelim, ilk bakışta array tek boyutluymuş gibi algılanabilir, ancak

y.ndim

2

Görüldüğü üzere tek boyutlu bir dizi gibi görülse bile iki boyutlu bir matrise dönüştürüldü.

a=np.arange(1,13)

a.reshape(3,4)

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

```
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Şeklinde de yapılabilir.

Array'lerde Birleştirme İşlemleri:

Tek Boyutlu Arrayleri Birleştirme İşlemleri:

concatenate() fonksiyonu ile tek boyutlu dizilerde birleştirme işlemlerini kolaylıkla gerçekleştirebiliriz.

a=np.array([1,2,3,4])

b=np.array([5,6,7,8])

şeklinde iki adet dizi oluşturalım. Bu iki diziyi birleştirmek için concatenate() fonksiyonunu kullanılmalıdır.

np.concatenate([a,b])

```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

görüldüğü üzere iki array uç uca ekleme yöntemi ile birleştirilmiştir. Oluşan yeni array tek boyutludur. İstersek 3. Hatta 4. Array ekleyerek uç uca çok sayıda dizi eklenebilir. Bunun için farklı array'ler tanımlayıp,

np.concatenate([a,b,c,d])

şeklinde kullanabiliriz.

Burada her iki array'de tek boyutlu olduğu için uç uca ekleme işlemi yapabildik. Bazen uç uca ekleme metodu yerine alt alta ekleme işlemi de yapılabilir. Bunun için array'lerin iki boyutlu olması gerekmektedir.

İki Boyutlu Array'leri Birleştirme İşlemleri:

Eğer elimizde iki boyutlu array'lar varsa bunları birleştirmek için iki farklı metod kullanabiliriz.

a=([1,2,3],
[4,5,6])

b=([7,8,9],
[10,11,12])

Şeklinde iki adet matrisimiz olsun

np.concatenate([a,b])

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

Görüldüğü üzere array'ler tek boyutlu ise uç uca; çift boyutlu ise alt alta ekleme işlemi yaptık. Peki iki boyutlu array'leri uç uca eklemek istersek ne yapabiliriz? Bu durumda concatenate() fonksiyonuna axis metodunu eklememiz gerekmektedir.

np.concatenate([a,b],axis=1)

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

Burada kullandığımız axis() metodu birleştirme işleminin satır basında veya sütun bazında yapılacağını belirlemektedir. Önceki örnekte biz axis() metodu kullanmadığımız halde alt alta ekleme işlemi yapmıştı. Bunun nedeni axis() metodunun varsayılan olarak 0 değeri almasıdır. Eğer 0 değeri girilirse altalta ekleme, 1 değeri verilirse uç uca ekleme işlemi yapmaktadır. Bu değeri görmek için jupyter'de çalışırken concatenate() fonksiyonunda parantez içindeyken klavyeden "shift tab" tuş kombinasyonuna basılarak fonksiyonun alabileceği parametre ve metodları görebiliriz.

```
In [ ]: np.concatenate([a,b])

In [ ]:  Docstring:
         concatenate((a1, a2, ...), axis=0, out=None)

In [ ]:  Join a sequence of arrays along an existing axis.
```

Görüldüğü üzere axis() metodu varsayılan olarak 0 değeri almaktadır.
Kullandığımız bu “shift tab” tuş kombinasyonunu numpy kütüphanesinin tüm fonksiyonları için kullanabiliriz.

Farklı Boyutlu array’leri Birleştirme İşlemleri:

Farklı boyuttaki array’leri birleştirmek istediğimizde aşağıdaki gibi bir hata ile karşılaşacağız.

x=([1,2,3],
[4,5,6])
y=([7,8,9])
np.concatenate([x,y])

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-41-14616cdb4093> in <module>
----> 1 np.concatenate([x,y])

ValueError: all the input arrays must have same number of dimensions
```

Resim 9

Görüldüğü üzere dizi boyutları aynı olmadığı için hata mesajı aldık.
Boyutları farklı dizileri birleştirmek için vstack() ve hstack() fonksiyonlarını kullanmamız gerekir. Dikey olarak birleştirme işlemi yapmak için vstack() “vertical-dikey” fonksiyonu, yatay olarak birleştirmek için ise hstack() “horizontal-yatay” fonksiyonunu kullanmamız gerekmektedir.

Vsctack Metodu ile Dikeyde Birleştirme İşlemleri:

x=([1,2,3],
[4,5,6])
y=([7,8,9])
np.vstack([x,y])

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Burada vtack() fonksiyonu yerine htcak() fonksiyonu ile birleştirme işlemi yapmak istersek boyut uyuş mazlığından dolayı hata ile karşılaşacağız. Yukarıda vtack() fonksiyonu ile birleştirme işlemi yapmak ist ediğimizde her iki array’in sütun sayıları eşit olduğu için kolaylıkla alt alta ekleme işlemi yaptık. Ancak burada hstack() fonksiyonu ile birleştirme işlemi yapmak istersek

Resim 10

Görüldüğü üzere 7, 8 ve 9 değerlerinin altındaki alanlar boş kaldığı için boyut uyuşmazlığı oluştuğunda n dolayı array'leri birleştirme işlemi yapılamamaktadır.

Hsctack Metodu ile Yatayda Birleştirme İşlemleri:

Aynı şekilde array'leri yatayda birleştirmek için hstack() fonksiyonu ile birleştirmek istersek array'leri mizin aşağıdaki gibi iki boyutlu olması gerekir.

x=([1,2,3],
[4,5,6])
y=([7,8],
[9,10])

şimdi bu iki array'i hstack() fonksiyonu ile birleştirmek istersek.

np.hstack([x,y])

```
array([[ 1,  2,  3,  7,  8],
       [ 4,  5,  6,  9, 10]])
```

Uç uca ekleyerek birleştirme işleminin yapıldığı görülmektedir. Array'lerin satır sayısı aynı olmakla birlikte sütun sayıları farklı olduğundan dolayı bu iki array üzerinde de vstack() fonksiyonunu kullanırsak hata ile karşılaşırız.

1	2	3	
4	5	6	
7	8	–	
9	10	–	

Burada da 3 ve 6 değerlerinin olduğu sütunda alt taraftaki değerler boş kaldığı için boyut uyuşmazlığı olmuştur ve array'lar bu hali ile vstack() fonksiyonu ile birleştirilemez.

Array'leri Bölme işlemleri

Tek Boyutlu Array'leri Bölme(parçalama) İşlemi

Array'leri birleştirdiğimiz gibi ayırma işlemleri de yapabiliriz.

a=np.random.randint(1,20,12)

```
array([ 5, 17,  4,  5,  9,  8, 16,  3, 10,  3, 13,  2])
```

şimdi oluşturduğumuz diziyi farklı boyutlarda olacak şekilde ayıralım. Hatırlanacağı üzere python'da listelerde çalışırken split() fonksiyonu ile belirlediğimiz kriterlere göre listelerimizi bölebiliyorduk. Array'leri ayırma işlemleri yaparken de split() fonksiyonu kullanabiliriz. Sprlit() fonksiyonunun özelliklerini incelersek

```
np.split(a)

Signature: np.split(ary, indices_or_sections, axis=0)
Docstring:
Split an array into multiple sub-arrays.
```

Birinci parametre olarak ayırma işlemi yapılacak array'in girildiği, ikinci parametre olarak ayırma işleminin nasıl yapılacağı, üçüncü parametre olarak ise hangi eksende ayırma işleminin yapılacağının belirtildiği görülmektedir.

np.split(a,[5])

[array([5, 17, 4, 5, 9]), array([8, 16, 3, 10, 3, 13, 2])]

Bu yaptığımız işlemle array'i ikiye bölerek iki farklı array oluşturmuş olduk. İstendiği takdirde daha fazla array'de oluşturulabilir. Bunun için

np.split(a, [2,5])

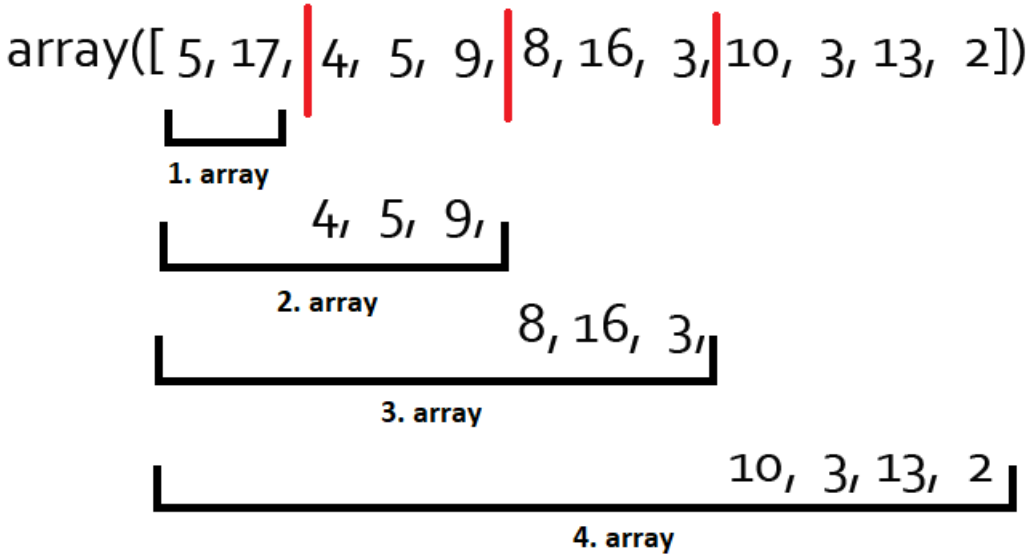
```
[array([ 5, 17]), array([4, 5, 9]), array([ 8, 16,  3, 10,  3, 13,  2])
]
```

Burada yapılan işlem ile array'in ilk iki elemanından 2 elemanlı bir array oluşturuldu, ardından ilk 5 elemanından 2. Array oluşturuldu ancak ilk elemandan birinci array oluşturulduğu için 2. Array için sadece 3 eleman alındı. Kalan elemanlardan da 7 elemanlı bir array daha oluşturulmuştur.

np.split(a, [2,5,8])

```
[array([ 5, 17]),
 array([4, 5, 9]),
 array([ 8, 16,  3]),
 array([10,  3, 13,  2])]
```

burada yapılan işlem aşağıdaki şekilde gösterilmiştir.



Resim 13

Burada ayırdığımız array'leri kullanmak istersek farklı değişkenlere gönderebiliriz.

x,y,z,q=np.split(a, [2,5,8])

a

```
array([ 5, 17])
```

b

```
array([4, 5, 9])
```

c

```
array([ 8, 16,  3])
```

d

```
array([10,  3, 13,  2])
```

İki Boyutlu Array'leri Bölme İşlemi

Tek boyutlu array'leri bölerken yeni oluşacak dizilerin kaçar elemandan oluşacağınız girerek ayırma işlemini yapabiliyorduk. Eğer array'imiz iki boyutlu olursa kümeleri birbirinden ayırmak suretiyle bölme işlemi yapabiliriz.

a=np.random.randint(0,20,12).reshape(3,4)

a

```
array([[18, 19, 16,  6],
       [ 1, 14,  2,  1],
       [17, 16,  8, 11]])
```

Şeklinde bir dizi olduğunu varsayalım,

np.split(a,[1])

```
[array([[18, 19, 16, 6]]), array([[ 1, 14,  2,  1], [17, 16,  8, 11]])]
```

Bu işleme dikeyde bölme işlemi denilmektedir.. bölme işleminden dönen değerleri değişkenlere atarsak,

x,y=np.split(a,[1])

x

```
array([[18, 19, 16, 6]])
```

y

```
array([[ 2,  2,  1,  0],
       [15,  5,  4,  4]])
```

Şeklinde bölündüğü görülmektedir.

Array'i 3 parçaya bölmek istersek,

np.split(a,[1,2])

```
[array([[18, 19, 16,  6]]),
 array([[ 1, 14,  2,  1]]),
 array([[17, 16,  8, 11]])]
```

aslında burada yapılan işlem vsplit() olarak geçmektedir. Yani **np.split(a,[1,2]) işlemi ile np.vsplit(a,[1,2]) aynı görevi yerine getirmektedir.**

np.vsplit(a,[1,2])

```
[array([[18, 19, 16,  6]]),
 array([[ 1, 14,  2,  1]]),
 array([[17, 16,  8, 11]])]
```

array'leri bölme işlemini dikeyde değil yatayda silme işlemi yapmak istersek hsplit() fonksiyonunu kullanmamız gerekmektedir.

np.hsplit(a,[2])

```
[array([[18, 19],
        [ 1, 14],
        [17, 16]]), array([[16,  6],
        [ 2,  1],
        [ 8, 11]])]
```

array'lerde Sıralama İşlemleri

sort() metodu ile sıralama işlemi

array'lerde sıralama işlemleri pythonda listelerde kullandığımız gibi sort() metodu ile yapılabilir.

a=np.random.randint(0,15,5)

a

```
array([ 2, 13, 12,  3,  6])
```

elimizde yukarıdaki gibi karışık bir array olduğunu varsayalım. Sort() metodu ile listemize sıralama işlemi uygulayabiliriz

np.sort(a)

```
array([ 2,  3,  6, 12, 13])
```

Python'da listelere sort() metodu uyguladığımızda liste'de yapılan işlem kalıcı olarak uygulanmış oluyordu. Burada sort işlemi yaptıktan sonra array'i tekrar görüntülemek istersek

a

```
array([ 2, 13, 12,  3,  6])
```

yapılan değişikliklerin array üzerinde uygulanmadığını görürüz. Sıralama işleminden sonra array'de kalıcı değişikliklerin olmasını istersek tekrar atama yapmamız gerekmektedir.

a=np.sort(a)

a

```
array([ 2,  3,  6, 12, 13])
```

argsort() metodu ile sıralama işleminden önceki indisleri öğrenmek:

Veri analizi yaparken verileri sıralama işlemi yaptıktan sonra, sıralama yapmadan önceki indis değerlerine sıklıkla ihtiyaç duyacağız. Bunun için agsort() metodunu kullanabiliriz.

a=np.random.randint(0,15,5)

a

```
array([ 2, 13, 12,  3,  6])
```

b=np.argsort(a)

b

```
array([0, 3, 4, 2, 1], dtype=int64)
```

burada argsort() metodu ile sort() metoduna maruz kalan array'in sıralama işleminden önce hangi indiste olduğu gösterilmektedir.

```
array([ 2, 13, 12,  3,  6])
```

listeye baktığımızda "2" değerinin 0. İndiste, "3" değerinin 3. İndiste, "6" değerinin 4. İndiste, "12" değerinin 2. İndiste ve "13" değerinin 1. İndiste olduğunu görüyoruz. Yani burada yaptığımız argsort() metodu ile sıralama işleminden önceki indis değerlerine ulaşmış oluruz.

Array() Eleman İşlemleri

Array'lerle çalışırken python listelerinde olduğu gibi indexleme ve sıralama işlemleri yapılabilir.

Tek boyutlu array() elemanlarına erişim işlemleri

Tek boyutlu array() lerde elemanlara erişim ve elemanlar üzerinde değişiklik yapma işlemleri python'daki liste işlemlerine benzemektedir.

a=np.random.randint(10,size=10)

a

```
array([1, 9, 7, 6, 4, 6, 2, 0, 8, 6])
```

a[0]

```
1
```

a[5]

```
6
```

a[-2]

```
8
```

Görüldüğü üzere python listelerinde olduğu gibi indis değerleri girilerek array'lerde de elemanlara ulaşılabilir.

Aynı listelerdeki gibi her bir indisteki değerler değiştirilebilir.

a[1]=10

a

```
array([ 1, 10,  7,  6,  4,  6,  2,  0,  8,  6])
```

İki boyutlu array() elemanlarına erişim işlemleri

İki boyutlu array() lerde elemanlara erişim işlemleri için bir adet matris oluşturalım.

a=np.random.randint(12,size=(3,4))

a

```
array([[ 3,  4, 10,  0],
       [11, 10,  0,  8],
       [ 6,  1,  0,  8]])
```

Tek boyutlu array'lerden farklı olarak köşeli parantez içerisine iki adet parametre girilmesi gerekmektedir.

a[0,0]

```
3
```

Burada kullandığımız 1. Parametre ile satır 2. Parametre ile sütun seçilmektedir.

a[1,3]

```
8
```

Yine indislerdeki değeri değiştirmek için

a[1,1]=2

a

```
array([[ 3,  4, 10,  0],
       [11,  2,  0,  8],
       [ 6,  1,  0,  8]])
```

Tek ve çift boyutlu array'lerde olmayan bir indeks görüntülenmek istenirse

a[5,4]

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-66-f542350ab3d9> in <module>
----> 1 a[5,4]

IndexError: index 5 is out of bounds for axis 0 with size 3
```

Out of bounds hatası ile karşılaşılır. Yani belirtilen indis array üzerinde bulunmamaktadır.

Tek boyutlu array() dilimleme işlemleri

Array() yapılarında tıpkı python'da olduğu gibi dilimleme işlemleri yapılabilir. Bunun için önce bir dizi oluşturalım.

a=np.arange(0,10)

a

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

array'in ilk iki elemanını listelemek istersek

a[0:2]

```
array([0, 1])
```

görüldüğü üzere aynı python'da listeleri dilimlediğimiz gibi dilimle işlemini gerçekleştirdik. Başlangıç değeri alındı ancak bitiş değeri alınmadı.

Başlangıç değeri vermeden listelemek istesek

a[:2]

```
array([0, 1])
```

yine aynı işlem yapılacaktır.

Başlangıç değeri verilip bitiş değeri verilmeden listeleme işlemi yapılabilir.

a[4:]

```
array([4, 5, 6, 7, 8, 9])
```

Python'daki gibi atlamalı olarak listeleme işlemi yapılabilir.

a[::2]

```
array([0, 2, 4, 6, 8])
```

İki boyutlu array() dilimleme işlemleri

İki boyutlu array'lerde dilimleme işlemi biraz daha karışıktır.

a=np.arange(16). reshape(4,4)

a

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Önce elemanlara erişim işlemleri yapalım

a[0,0]

```
0
```

0. satır 0. Sütundaki değere ulaştık. Burada satır ve sütunlari birbirinden ayırmak için “,” işareti kullanılmaktadır. Başka bir değere ulaşmak için

a[2,3]

```
11
```

2. satır 3. Sütundaki değere ulaşmış olduk.

Şimdi de satırlara veya sütunlara komple erişim işlemi yapalım.

a[1,:]

```
array([4, 5, 6, 7])
```

burada virgülden önce verdiğimiz 1 parametresi ile 1. Satırı seçmiş olduk, virgülden sonra kullandığımız “:” işareti ile tüm sütunları seçmiş olduk. Yani 1. Satırdaki tüm verilere ulaşmış olduk.

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Resim 15

Aynı işlemleri sütun bazlı yapmak isteyebiliriz bunun için 0. Sütunu seçmek isteyelim,

a[:,0]

```
array([ 0,  4,  8, 12])
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Resim 16

virgülden önceki ifade ile “:” tüm satırları seçtik, virgülden sonra ise 0. Sütunu seçerek sadece 0. Sütunda bulunan tüm elemanlara ulaşmış olduk.

a[2,:]

```
array([ 8,  9, 10, 11])
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Resim 17

2. satırdaki tüm elemanlara ulaştık.

3. sütundaki tüm elemanlara ulaşmak için

a[:,3]

```
array([ 3,  7, 11, 15])
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Resim 18

şeklinde kullanabiliriz.

Dilimleme işlemi yaparken satırların ya da sütunların tamamını alma işlemini görmüş olduk, aynı şekilde satır veya sütunlarda bazı bölümler üzerinde de dilimleme işlemi yapabiliriz.

a[:2,:3]

```
array([[0, 1, 2],
       [4, 5, 6]])
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Resim 19

Burada ilk iki satırı ve ilk üç sütunu alarak listeleme işlemi yaptık.

Şimdi de

a[2:,2:]

```
array([[10, 11],
       [14, 15]])
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Resim 20

2. ve 3. Satır ile 2. Ve 3. Sütunu almış olduk.

İlk 3 satır ve ilk 2 sütunu almak istersek

a[:3,:2]

```
array([[0, 1],
       [4, 5],
       [8, 9]])
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Resim 21

İşlemini yaparız. Aslında burada yapılan işlem

a[0:3,0:2]

işlem ile aynıdır.

Tüm sütunları ve ilk 2 satırı seçmek itersek

a[:2,:]

```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Resim 22

tüm satırlar ve son 3 sütunu seçmek için

a[:,1:]

```
array([[ 1,  2,  3],
       [ 5,  6,  7],
       [ 9, 10, 11],
       [13, 14, 15]])
```

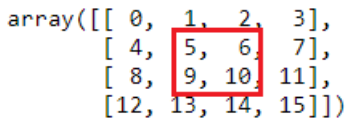
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Resim 23

1. ve 2. Satır ile 1. Ve 2. Sütunu alma işlemi yapalım

a[1:3,1:3]

```
array([[ 5,  6],
       [ 9, 10]])
```



Resim 24

Array'lerde değerler üzerinde değişiklik işlemleri:

Array'lerde tıpkı listelerde olduğu gibi indisler üzerinde değişim işlemleri yapılabilir. Bunun için indis değerini belirmemiz gerekmektedir.

a=np.random.randint(0,20,9)

a

```
array([ 0, 18, 16,  8,  5, 11,  5,  7,  4])
```

a[2]=1

a

```
array([ 0, 18,  1,  8,  5, 11,  5,  7,  4])
```

görüldüğü üzere 2. İndisteki "16" değeri 1 olarak değişmiştir. Aynı şekilde çoklu değer atama işlemleri de yapılabilir.

a[:3]=7

a

```
array([ 7,  7,  7,  8,  5, 11,  5,  7,  4])
```

Bu yaptığımız işlemle array'in ilk 3 indisindeki değeri 7 yapmış olduk.

Array'lerde kopyalama ve Çoğaltma işlemleri

Bir liste oluşturduğumuz zaman bu listeyi kopyalamak için

a=[1,2,3,4,5]

b=a

b[0]=7

a

```
[7, 2, 3, 4, 5]
```

Şeklinde bir metod kullandığımızda b listesinde bir değişiklik yapmak istediğimizde a listesinde de aynı değişiklik oluyordu. Aynı işlemi array'ler üzerinde yaptığımızda da yapılan değişiklik her iki listeye yansıyacaktır.

a=np.arange(1,6)

b=a

b

```
array([1, 2, 3, 4, 5])
```

b[0]=8

a

```
array([8, 2, 3, 4, 5])
```

görüldüğü üzere aynı durum array'ler içinde geçerli oldu. Pythonda bir listeyi kopyalamak için for döngüsü kullanarak append metodu kullanıyorduk ya da copy() fonksiyonundan yararlanıyorduk. Aynı şekilde array'lerde de copy() fonksiyonu ile listeyi çoğaltabiliriz.

c=np.copy(a)

c

```
array([8, 2, 3, 4, 5])
```

a[0]=1

c

```
array([8, 2, 3, 4, 5])
```

görüldüğü üzere birbirinden bağımsız iki liste oluşturduk. Aynı şekilde dilimleme işlemi yaptıktan sonra, seçtiğimiz değerleri farklı bir listeye atama işlemi yapabiliriz.

a=np.random.randint(0,10,(4,5))

a

```
array([[0, 3, 8, 3, 1],
       [7, 7, 1, 9, 2],
       [0, 7, 6, 8, 2],
       [3, 0, 1, 4, 6]])
```

Dilimleme işlemi ile seçtiğimiz değerleri farklı bir değişkene atayalım.

b=a[::,0:2]

b

```
array([[1, 2],
       [7, 7],
       [0, 7],
       [3, 0]])
```

Burada oluşturduğumuz alt kümede(b) bir değişiklik yaptığımız zaman aynı şekilde ana değişken üzerinde de değişiklik olacağı için alt kümeyi copy fonksiyonu ile çoğaltmamız gerekmektedir.

b=a[::,0:2].copy()

artık bağımsız bir alt küme olmuştur ve yapılan değişiklikler a değişkenine yansımaz.

Koşullu İşlemler

Array'lerde karşılaştırma operatörleri kullanılabilir.

a=np.arange(1,8)

a

```
array([1, 2, 3, 4, 5, 6, 7])
```

a>5

```
array([False, False, False, False, False,  True,  True])
```

görüldüğü üzere verdiğimiz koşulu sağlayan değerler için True sağlamayan değerler için False değeri üretiliştir.

a<=5

```
array([ True,  True,  True,  True,  True, False, False])
```

buradaki yapı incelendiğinde sanki sadece yapılan işlemlerden True False değerleri dönüyor görünse de aslında bu değerler üzerinde çeşitli işlemler yapılabilir.

np.sum(a>2)

```
5
```

Bu işlem sonucunda 2 değerinden büyük toplam 5 adet değer olduğu(3,4,5,6,7) görülmektedir.
Yaptığımız işlemi mantıksal operatörleri de ekleyerek genişletebiliriz.

np.sum((a>2) & (a<=5))

```
3
```

Array içerisinde 2'den büyük ve 5 ten küçük veya eşit toplam 3 değer bulunmaktadır.
Hatırlanacağı üzere python'da mantıksal bağlaçları kullanırken "and" ve "or" operatörleri vardı. Array'lerle çalışırken bu bağlaçların yerine &(and), | (or) operatörleri kullanabiliyoruz.
Bu yaptığımız işlemi satır ve sütun bazında da yapabiliriz.

a=np.random.randint(1,10,(3,3))

a

```
array([[4, 8, 7],
       [3, 6, 2],
       [8, 7, 2]])
```

np.sum(a>5)

```
5
```

Yukarıdaki gibi bir işlem yaparsak tüm değerler içerisinde kaç tanesinin 5 ten büyük olduğu bilgisine erişebiliriz. Eğer satır veya sütunları ayrı ayrı yapmak istersek,

np.sum(a>5,axis=1)

```
array([2, 1, 2])
```

1. satırda 2 değer, 2. Satırda 1 değer 3. Satırda 2 değerin 5 ten büyük olduğunu görürüz. Axis değerine 0 verirsek sütun bazında inceleme işlemi yapabiliriz.

np.sum(a>5,axis=0)

```
array([1, 3, 1])
```

1. sütunda 1 değer, 2. Sütunda 3 değer ve 3. Sütunda 1 değer 5 ten büyüktür.

All ve Any ifadeleri

Matrisin tüm elemanlarının bir koşulu aynı anda gerçekleştirip gerçekleştirmediğini kontrol etmek istediğimizde **all** ve **any** ifadeleri kullanılabilir.

a=np.random.randint(1,10,(3,3))

a

```
array([[4, 8, 7],
       [3, 6, 2],
       [8, 7, 2]])
```

np.all(a>3)

```
False
```

Bütün değerler 3 den büyük mü sorgusu yapıldı ve False değeri döndü

np.all(a>1)

```
True
```

Bütün değerler 1 den büyük mü sorgusu yapıldı ve True değeri döndü

np.any(a<5)

```
True
```

Herhangi bir değer 5 den küçükmü sorgusu yapıldı ve array içerisinde en az bir tane 5 den küçük değer olduğu için True değeri döndü.

Koşullu İfadelerde Elemanlara Erişme İşlemleri

Buraya kadarki işlemlerde genellikle indis değerleri üzerinden işlemler yaptık. İstersek elemanlar üzerinde de bazı işlemler yapabiliriz.

a=np.arange(1,6)

a

```
array([1, 2, 3, 4, 5])
```

Array'in yukarıdaki gibi olduğunu varsayalım. Eğer array'de indis değeri üzerinden işlem yapmak istersek

a[2]

```
3
```

Şeklinde indis değerleri üzerinden elemanlara ulaşabiliriz.

Ayrıca koşullu ifadeler ile belirli koşulu sağlayan değerlere de ulaşabiliriz.

a>2

```
array([False, False,  True,  True,  True])
```

şimdi öğrendiğimiz bu iki ifade ile array'deki değerleri bulalım

a[a>2]

```
array([3, 4, 5])
```

bu yaptığımız işlemle koşulu gerçekleştiren değerleri getirmiş olduk.

Farklı bir yöntem kullanmak istersek

a=np.arange(1,10)

liste=a>5

liste

```
array([False, False, False, False, False,  True,  True,  True,  True])
a[liste]
array([6, 7, 8, 9])
```

şeklinde bir filtreleme işlemi de yapılabilir.
İfade de 3'den büyük değerlerin toplamını bulak istersek
np.sum(a[a>3])
```
9
```

Array'ler Üzerinde Matematiksel İşlemler

Array'ler üzerinde bazı matematiksel işlemler yapılabilir.
a=np.arange(10)
a
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```
a-5
```
array([-5, -4, -3, -2, -1,  0,  1,  2,  3,  4])
```
a+3
```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```
a*2
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```
a/2
```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```
a2**
```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81], dtype=int32)
```
görüldüğü üzere array üzerinde toplama çıkarma çarpma bölme gibi işlemler kolaylıkla yapılabilmektedir.

Array'ler üzerine uygulanabilen özel fonksiyonlar

- **max() ve min() fonksiyonları**

Array içerisindeki en büyük ve en küçük ifadeleri max() ve min() fonksiyonları ile bulabiliriz.
a=np.arange(10)
a
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```
max(a)
```
9
```
min(a)
```
0
```
Array içerisindeki en büyük ve en küçük değerlerin hangi indislerde olduğunu bulmak için argmax() ve arg() min fonksiyonları kullanılmalıdır.
a=np.random.randint(1,20,10)
a
```
array([ 9,  6, 15, 12,  9,  3, 16, 11, 11, 17])
```
np.argmax(a)
```
9
```
np.argmin(a)
```
5
```

- **sum() fonksiyonu**

Array içerisindeki değeri toplamak istersek sum() fonksiyonu kullanabiliriz.
sum(a)
```
45
```
Aynı işlemi matrisler üzerinde de gerçekleştirebiliriz.
a=np.arange(1,10).reshape(3,3)
a
```
array([[1, 2, 3],
       [4, 5, 6],
```

```
       [7, 8, 9]])
```
a.sum()
```
45
```
Her bir satırın ayrı ayrı toplamlarını bulmak istersek axis değerini değiştirebiliriz.
a.sum(axis=1)
```
array([ 6, 15, 24])
```
her bir sütunun ayrı ayrı toplamını bulmak için ise axis değeri 0 yapılmalıdır.
a.sum(axis=0)
```
array([12, 15, 18])
```

- **mean() fonksiyonu**

Array'ler içerisindeki değerlerin ortalamasını bulmak için mean() fonksiyonu kullanılmalıdır.
a.mean()
```
5
```

- **median() fonksiyonu**

array içerisindeki değerlerin median'ının bulmak için kullanılır. (tüm veri noktalarını sıralayarak ve ortadakini seçer)
np.median(a)
```
5.0
```

- **std() fonksiyonu**

array içerisindeki değerlerin standart sapmasını bulmak için kullanılır.
a.std()
```
2.581988897471611
```

Array'lerde Birleştirme İşlemleri

Aynı Boyutlu Array'lerde Matematiksel İşlemler

Array'lerde dilimleme parçalama işlemleri yapılabileceği gibi birleştirme işlemleri de yapılabilmektedir.
a=np.arange(1,4)
b=np.arange(4,7)
a
```
array([1, 2, 3])
```
b
```
array([4, 5, 6])
```
a+b
```
array([5, 7, 9])
```
görüldüğü üzere iki adet array üzerinde toplama işlemi yapılmış oldu. Aynı şekilde çıkartma, çarpma ve bölme işlemleri de kolaylıkla yapılabilir. Eğer array'ler aynı boyutlu ise kolay bir şekilde birleştirme işlemleri yapılabilmektedir. Array'lerin boyutları aynı değilse hata verecektir.
c=np.array([10,11,12,13])
c
```
ValueError                                Traceback (most recent call last)
<ipython-input-8-e200917bc55f> in <module>
----> 1 a+c

ValueError: operands could not be broadcast together with shapes (3,) (4,)
```
Resim 25
Burada a 3 elemanlı, c ise 4 elemanlı array'ler olduğu için boyut uyuşmazlığından dolayı toplama işlemi gerçekleştirilememiştir. Birden fazla array üzerinde matematiksel işlemler yapılabileceği gibi tek bir array üzerine de aritmetiksel işlemler uygulanabilir
Örnek olarak a array'ini 5 değeri ile toplayabiliriz.

a+5

```
array([6, 7, 8])
```

a*3

```
array([3, 6, 9])
```

burada array'in her bir elemanı yapılan işleme maruz kalmış oldu

array'ler üzerinde temel matematiksel fonksiyonları kullanabiliriz. Array içerisinde bulunan en küçük değer

np.min(a)

```
1
```

Maksimum değer

np.max(a)

```
3
```

Değerlerin ortalaması

np.mean(a)

```
2.0
```

Farklı Boyutlu Array'lerde Toplama İşlemleri

Farklı boyutlu array'lerde birleştirme işlemleri yaparken

a=np.arange(1,4)

a

```
array([1, 2, 3])
```

b=np.arange(1,10).reshape(3,3)

b

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

a+b

```
array([[ 2,  4,  6],
       [ 5,  7,  9],
       [ 8, 10, 12]])
```

Görüldüğü üzere tek boyutlu array, matrisle sanki aynı boyutlucaymışcasına her bir satır üzerinde toplama işlemi yapılmış oldu. Yani tek boyutlu array kendisinden farklı boyuttaki array üzerine eklenmiş oldu. Satırlar eşit sütunlar eşit değilse,

c=np.arange(1,4).reshape(3,1)

c

```
array([[1],
       [2],
       [3]])
```

a+c

```
array([[2, 3, 4],
       [3, 4, 5],
       [4, 5, 6]])
```

Burada boyutlar uyuşmadığı halde b isimli array'e ekleme işlemi yaparak toplama işlemi yapılmıştır. Aslında yukarıdaki 1 satır 3 sütunlu array'le yapılan işlemden farkı yoktur.

Satır ve sütunların birbirinden farklı olduğu array'lerde

a=np.arange(4)

a

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

b=np.arange(4).reshape(4,1)

b

```
array([[0],
```

```
       [1],
       [2],
       [3]])
```

a+b

```
array([[0, 1, 2, 3],
       [1, 2, 3, 4],
       [2, 3, 4, 5],
       [3, 4, 5, 6]])
```

Yukarıdaki örneklerde boyutlar uyuşmasa dahi tek boyutlu arrayler kendilerinden farklı boyuttaki array'lerle kolaylıkla birleşmektedir. Bazı durumlarda array'lerin yapısı uyuşmadığı için birleştirme işlemi gerçekleştirilemez

a=np.full((2,3),6)

a

```
array([[6, 6, 6],
       [6, 6, 6]])
```

b=np.ones((3,4))

b

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

a+b

```
ValueError                                Traceback (most recent call last)
<ipython-input-52-ca730b97bf8a> in <module>
----> 1 a+b

ValueError: operands could not be broadcast together with shapes (2,3) (3,4)
```

Resim 26

Array'ler üzerinde iterasyon işlemleri

Array'ler üzerinde for döngüleri ile gezinme(iterasyon) işlemleri yapabiliriz. Tek boyutlu array'lere iterasyon işlemi yapmak listeler üzerinde yaptığımız iterasyon işlemi ile aynıdır.

```
a=np.arange(1,10)
for i in a:
   print(i,end=",")
```

```
1,2,3,4,5,6,7,8,9,
```

İki boyutlu array'lerde iterasyon işlemi yapmak için tek boyutlu dizilere uyguladığımız metodu uygularsak,

```
a=np.arange(1,10).reshape(3,3)
for i in a:
   print(i,end=",")
```

```
[1 2 3]
[4 5 6]
[7 8 9]
```

Şeklinde bir çıktı alırız. Burada satır satır yazdırma işlemi yapılmıştır. Yani önce birinci satır, sonra ikinci satır ve son olarak üçüncü satır ekrana basılmıştır. flat metodu kullanarak tüm elemanları alt alta yazdırma işlemi yapabiliriz.

```
for i in a.flat:
   print(i)
```

```
1
2
```

```
3
4
5
6
7
8
9
```

Örnekler:

1) Numpy kütüphanesini ekleyin.

import numpy as np

2) ([1,2,3,4,5,6]) elemanlarından oluşan bir array oluşturun

np.arange(1,7)

```
array([1, 2, 3, 4, 5, 6])
```

3) içerisinde 6 tane 1 barındıran bir array oluşturun

np.ones(6)

```
array([1., 1., 1., 1., 1., 1.])
```

3) 0 lardan oluşan 3 satır 4 sütunlu bir array oluşturun

np.zeros([3,4])

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

4) 8'lerden oluşan 4 satır 5 sürunlu bir array oluşturun

np.full((4,5),8)

```
array([[8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8]])
```

5) 20 den başlayarak 50 ye kadar 3 er atlamalı bir array oluşturun.

np.arange(20,50,3)

```
array([20, 23, 26, 29, 32, 35, 38, 41, 44, 47])
```

6) 2,9,7,1,4,6,3 değerlerinden oluşan tek boyutlu bir array oluşturun.

np.array([2,9,7,1,4,6,3])

```
array([2, 9, 7, 1, 4, 6, 3])
```

7) 0 ile 20 arasında aynı uzaklıkta 5 elemanlı bir array oluşturun(linspace)

np.linspace(0,20,5)

```
array([ 0.,  5., 10., 15., 20.])
```

8) normal dağılımlı 3 satır 4 sütunlu bir array oluşturun (randn)

np.random.randn(3,4)

```
array([[-0.55207874,  0.59605469,  0.23411568, -0.02846559],
       [ 0.16792082,  1.17885437,  0.46727986,  0.20187447],
       [-0.96881594, -1.0061479 , -1.72933383,  0.50389727]])
```

9) 4 elemanlı birim matris oluşturun (eye)

np.eye(4)

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

10) 0-1 arası 10 adet rasgele sayı oluşturup bu sayıların en büyük ve en küçüğünü bulun.

a=np.random.rand(10)

a

```
array([0.24945433, 0.95785697, 0.67377922, 0.42080402, 0.87089963,
       0.30230424, 0.26856852, 0.20605942, 0.50006395, 0.38619797])
```

max(a)

```
0.957856968949654
```

min(a)

```
0.20605942455124104
```

11) Rasgele sayılardan oluşan 5 satır 4 sütunlu bir matris oluşturun.

a=np.random.randint(1,30,20).reshape(5,4)

a

```
array([[ 2, 20, 16, 23],
       [18,  6, 18,  2],
       [ 7, 14, 19, 12],
       [24,  7, 23, 10],
       [22, 14, 18,  4]])
```

12) 1. Satırdaki tüm değerlere ulaşın

a[1,:]

```
array([18,  6, 18,  2])
```

13) 2. Satır 3. Sütundaki elemana ulaşın

a[2,3]

```
12
```

14) 3. Sütundaki tüm elemanlara ulaşın

a[:,3]

```
array([23,  2, 12, 10,  4])
```

15) for döngüsü ile 2 satır ve 3 sütundan oluşan matrisin elemanlarını alt alta yazınız.

a=np.arange(1,7).reshape(2,3)

```
for i in a.flat:
    print(i)
1
2
3
4
5
6
```

16) 0-1 arası rasgele 6 sayı oluşturup 5 ile çarpın

a=np.random.rand(6)
a*5

```
array([0.25043448, 0.17300977, 4.54521728, 3.83036951, 4.70327729,
       0.16850366])
```

17) array'in sayıları toplamını bulun
sum(a)

```
2.7341623982675354
```

18) array'in median'ınını bulun
np.median(a)

```
0.6037817371739695
```

19) array'in standart sapmasını bulun
np.std(a)

```
0.24551707547157878
```