PYTHON NUMPY KÜTÜPHANESİ NOTLARI (kaynak: onuraltuntas.net)

Numpy bir python kütüphanesidir. Bu kütüphane matematiksel işlemler yaparak veri işlemleri gerçekleştirir.

numpy kütüphanesi np kısaltma adıyla aşağıdaki gibi eklenir:

```
import numpy as np
```

Dizi (array) oluşturma:

```
dizi = np.array( [1, 2, 3, 4, 5, 6] )
dizi
Output: [1 2 3 4 5 6]
```

Dizinin veri tipini öğrenme: Yukarıdaki dizinin tipi numpy veri tiplerinden ndarray tipindedir. Python'da olduğu gibi numpy kütüphanesinin de kendine özgü veri tipleri vardır. ndarray bunlardan biridir.

```
type(dizi)
Output: numpy.ndarray
```

Dizi içindeki verilerin tipini öğrenme: Yukarıdaki dizinin içindeki veriler int32 veri tipindedir.

```
dizi.dtype
Output: dtype('int32')
```

Numpy içindeki verilerin tipi en geniş anlamda tanımlanır: Örneğin dizinin içinde hem tam sayı hem de float türü sayı varsa bu durumda dtype değeri float olarak belirlenir. Çünkü float veri türü tam sayıları da kapsar. Kısacası NumPy dizileri, bellek verimliliği ve performans nedeniyle homojen veri tipleri gerektirir bu nedenle bir dizi için sadece tek bir veri tipi tanımlanır. Örnek:

```
dizi = np.array( [1, 1.5, 2, 2.5, 3, 3.5, 4] )
dizi.dtype
Output:    dtype('float64')
```

Soru: float64 ve int32 ne anlama gelmektedir?

Cevap: float64 veri tipi, 64 bit uzunluğunda bir ondalık sayı veri tipidir. 64 bit uzunluğunda olduğu için, bu veri tipi 2^64 farklı sayıyı temsil edebilir. Bu sayıların aralığı, yaklaşık olarak -1.8 x 10^308 ile 1.8 x 10^308 arasındadır. Bu, float64 veri tipinin, oldukça büyük veya küçük ondalık sayıları veya çok hassas

ondalık sayıları temsil etmek için yeterince büyük bir aralığa sahip olduğu anlamına gelir. 64 bit uzunluğunda olduğu için, float64 veri tipi, daha küçük boyutlu float veri tiplerine göre daha fazla bellek kullanır. Ancak, daha büyük boyutlu ondalık sayıları daha doğru bir şekilde temsil edebildiği için, bazı uygulamalarda daha uygun bir seçim olabilir. Örneğin, bilimsel hesaplamalarda veya finansal uygulamalarda, float64 veri tipi daha doğru sonuçlar vermek için tercih edilebilir. Aynı şekilde int32'de 32 bit uzunluğunda bir tam sayı veri tipidir.

Soru: Numpy kütüphanesi genel olarak hangi veri tiplerini destekler?

Cevap: NumPy kütüphanesi birçok farklı veri tipini destekler. Bunlar:

1. int8, int16, int32, int64: işaretli tam sayılar
2. uint8, uint16, uint32, uint64: işaretsiz tam sayılar
3. float16, float32, float64, float128: kayan nokta sayıları
4. complex64, complex128, complex256: karmaşık sayılar
5. bool: boolean değerler
6. object: Python nesneleri için referanslar
7. string_: sabit uzunluklu karakter dizileri için veri tipi

Ayrıca, kullanıcının özelleştirilmiş veri tipleri tanımlamasına da izin veren bir dtype sınıfı vardır. Bu, NumPy'nin genişletilebilirliğini artırır ve özellikle özel veri tipleri gerektiren uygulamalarda faydalı olabilir.

Soru: Özelleştirilmiş veri tiplerine örnek olarak ne verilebilir?

Cevap: NumPy'deki özelleştirilmiş veri tipleri, özellikle özel veri türleri gerektiren bilimsel hesaplama ve veri işleme uygulamaları için oldukça faydalıdır. Özelleştirilmiş veri tipleri, ndarray nesnelerinde saklanabilen öğelerin veri tipini belirleyen dtype sınıfının alt sınıflarıdır. Bazı örnekler aşağıda verilmiştir: Görüntü İşleme: Renkli görüntüler, piksel değerleri R (Kırmızı), G (Yeşil) ve B (Mavi) kanallarında saklanır. Her piksel, 8 bitlik (0-255 aralığında) bir tamsayı veya 0 ile 1 arasında bir kayan noktalı sayı olarak kodlanabilir. Bu nedenle, NumPy'de özel olarak tanımlanmış bir dtype sınıfı, örneğin "RGBImage" adında, 3 kanalın her biri için farklı bir veri tipi kullanarak görüntülerin saklanmasına izin verebilir.

Finansal veri işleme: Finansal piyasalarda, özellikle opsiyon fiyatlamada, karmaşık matematiksel modeller kullanılır. Bu modeller, matris hesaplamalarında kullanılmak üzere özel veri tipleri gerektirir. Örneğin, bir dtype sınıfı, özel bir karmaşık sayı veri tipi ve bir dizi finansal işlemin özelliklerini saklayan bir yapısal veri tipi (record type) içerebilir.

Bilimsel hesaplama: Bazı bilimsel hesaplamalar, özel matris türleri için özel veri tipleri gerektirir. Örneğin, bir "SparseMatrix" adlı özel bir matris türü, matrisin büyük bir çoğunluğunun sıfır olduğu durumlarda kullanılır. Bu matrisler, bellek kullanımını azaltmak için özel bir veri tipi kullanarak saklanabilir.

Bu örnekler, özelleştirilmiş dtype sınıflarının kullanımına örnek olarak gösterilebilir. NumPy, özelleştirilmiş veri tiplerinin tanımlanması ve kullanılması için gelişmiş bir destek sunar ve kullanıcıların kendi özel veri tiplerini tanımlamasına ve uygulamalarında kullanmasına olanak tanır.

Soru: uint32 ne anlama gelmektedir?

Cevap: uint32, 32 bit uzunluğunda, işaretli olmayan tam sayıların tutulduğu bir veri tipidir. "uint" kısaltması, "unsigned integer" anlamına gelir ve işaretli olmayan tam sayıları ifade eder. İşaretli olmayan bir veri tipi, negatif veya pozitif olabilecek herhangi bir sayı değerini ifade edebilen bir tamsayıdır, ancak işaret bilgisi olarak kullanılmaz. uint32, 0 ile (2^32)-1 arasındaki tam sayı değerlerini depolayabilir, yani 0'dan en fazla 4.294.967.295'e kadar olan tamsayıları temsil edebilir. uint32, özellikle düşük seviyeli bilgisayar programlaması gibi uygulamalarda sıklıkla kullanılır. Örneğin, önbellek adresleri, bellek boyutları ve ağ protokolleri gibi yerlerde uint32 veri tipi kullanılabilir. NumPy'da, uint32 veri tipi, np.uint32 olarak ifade edilir ve NumPy dizilerinde kullanılabilir.

Soru: İşaretli olmayan tam sayı ne demektir?

Cevap: İşaretli olmayan bir tamsayı, negatif veya pozitif olabilecek herhangi bir sayı değerini ifade edebilen bir tamsayıdır, ancak işaret bilgisi olarak kullanılmaz. Bu, işaretli veri tiplerinin aksine, bir sayının pozitif mi yoksa negatif mi olduğunu belirtmek için ayrılmış bir biti olmadığı anlamına gelir. Örneğin, 8 bitlik bir işaretli veri tipinde, ilk bit genellikle işaret biti olarak kullanılır ve bu bit, sayının pozitif mi yoksa negatif mi olduğunu belirler. İşaretli bir veri tipi, -128 ile 127 arasındaki tam sayıları temsil edebilir, ancak işaretli olmayan bir 8 bitlik tamsayı veri tipi, 0 ile 255 arasındaki tam sayıları temsil edebilir. İşaretli olmayan veri tipleri genellikle, sayıların işareti ile ilgilenmeyen veya işaret bilgisi gerektirmeyen uygulamalarda kullanılır. Bu uygulamalar arasında, özellikle mikrodenetleyiciler, donanım yazılımları, ağ protokolleri ve veri sıkıştırma algoritmaları yer alır.

Soru: Bir NumPy dizisi içinde string veri türleri nasıl kullanılır?

Cevap: Bir NumPy dizisi içinde stringler de olabilir. NumPy, dizi elemanları için farklı veri tipleri sağlar ve string veri tipleri de bunlardan biridir. NumPy dizilerinde string veri tipleri, 'S' veya 'U' karakterleriyle tanımlanır. 'S' tipi veri tipleri, byte dizileri içindir. 'S10' gibi bir veri tipi, her elemanın en fazla 10 byte uzunluğunda bir byte dizisi olduğunu belirtir. Örneğin, "hello" kelimesi 'S5' tipinde bir dizi elemanıdır. 'U' tipi veri tipleri, Unicode karakter dizileri içindir. 'U10' gibi bir veri tipi, her elemanın en fazla 10 karakter uzunluğunda bir Unicode karakter dizisi olduğunu belirtir. Örneğin, "Merhaba" kelimesi 'U7' tipinde bir dizi elemanıdır. NumPy dizilerinde string veri tipleri, çeşitli uygulamalarda kullanılır. Örneğin, metin verileri, dosya adları, etiketler ve sınıf adları gibi veriler string veri tipleri kullanılarak depolanabilir.

Soru: Bir numpy dizisi içinde hem sayı hem de string veri türleri olabilir mi? Bu durumda dtype olarak ne tür veri tanımlanması yapılması gerekir?

Cevap: Bir NumPy dizisi içinde hem sayılar hem de stringler bulunabilir. Ancak, bu tür bir dizi oluşturmak pek önerilmez, çünkü dizinin elemanlarındaki veri türleri aynı olmalıdır ve bu nedenle tamsayı ve stringleri aynı dizi içinde birleştirmek, bu kurala aykırıdır. Örneğin, aşağıdaki gibi bir dizi tanımlamaya çalıştığımızda bir hata alırız:

```
import numpy as np
arr = np.array([1, 'hello', 2, 'world'])
```

Bu kod, TypeError hatası verir çünkü dizi içindeki öğelerin veri türleri farklıdır. Eğer yine de tamsayılar ve stringler içeren bir NumPy dizisi oluşturmanız gerekiyorsa, her bir elemanı string veri tipinde saklayabilirsiniz. Bunu yapmak için, NumPy dizisini oluştururken dtype parametresini 'U' (Unicode string) olarak tanımlayabilirsiniz.
Örneğin:

```
import numpy as np
arr = np.array(['1', 'hello', '2', 'world'], dtype='U')
```

Bu dizi, elemanların hepsini string olarak tutar, ancak sayısal değerler de hala bu dizide saklanabilir ve gerektiğinde string olarak dönüştürülebilir.

Dizi içindeki verilerin tipini belirleme: Dizi içindeki verilerin tipini aşağıdaki gibi önceden belirleyebiliriz.

```
dizi = np.array( [1 ,2, 3, 4, 5, 6] , dtype=float)
dizi
Output: [1. 2. 3. 4. 5. 6.]
```

İki boyutlu bir dizi oluşturma:

```
dizi = np.array([ [1, 2, 3] , [4, 5, 6] ])
dizi
Output:
[[1 2 3]
 [4 5 6]]
```

---

3 satır ve 4 sütundan oluşan iki boyutlu bir dizi aşağıdaki gibi oluşturulur.

```
dizi = np.array([ [1, 2, 3, 4] , [ 5, 6, 7, 8] , [9, 10, 11, 12]  ])
dizi
Output:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9  10 11 12]]
```

Dizilerin kaç boyutlu olduğunu öğrenme: Yukarıdaki dizi 2 boyutludur.

```
dizi.ndim
Output: 2
```

Üç boyutlu dizi oluşturma:

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

dizi
```

```
Output:
[[[ 1  2  3]
  [ 4  5  6]]
 [[ 7  8  9]
  [10 11 12]]]
```

Dizilerin satır ve sütun sayısını öğrenme: Aşağıdaki dizi 2 satır ve 3 sütuna sahiptir.

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

dizi.shape
```

```
Output: (2, 3)
```

Tek bir diziden shape oluşturma:

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

# yukarıdaki diziyi 2 adet satır ve 3 adet sütuna dönüştürüyoruz.

dizi.reshape(2, 3)
```

```
Output:

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

Ek bilgi: shape oluştururken satır ve sütun sayılarının çarpımı dizi eleman sayısına eşit olmalıdır.

Otomatik sayı dizisi oluşturma:

```
# 0'dan başlayarak 30'a kadar (30 dâhil değil) dizi oluşturma

dizi = np.arange(30)

dizi
```

```
Output:

[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29]
```

Sabit aralıklı otomatik sayı dizisi oluşturma:

```
# 0'dan başlayarak 30'a kadar üçer sıra atlayarak dizi oluşturma
dizi = np.arange(0, 30, 3)
dizi
Output: [0 3 6 9 12 15 18 21 24 27]
```

Bir diziyi kopyalama: Yukarıdaki diziyi kopyalayalım.

```
dizi2 = dizi.copy()
dizi2
Output: [ 0 3 6 9 12 15 18 21 24 27]
```

Rastgele sayılardan oluşan bir dizi oluşturma:

```
# dört adet rastgele sayıdan oluşan tek boyutlu bir dizi oluşturma
dizi = np.random.random((4))
dizi
Output: [0.67853348 0.71996834 0.95678727 0.18654227]
```

Rastgele tamsayılardan oluşan tek boyutlu bir dizi oluşturma

```
dizi = np.random.randint(20, size=4)
dizi
# 20 değeri üretilecek sayıların 20'den küçük olmasını sağlar.
# size özelliği kaç adet sayı üretileceğini belirler
Output:  [10  5  8 14]
```

---

2 satırlı, 4 sütunlu rastgele sayılardan oluşan iki boyutlu bir dizi oluşturma

```
dizi = np.random.random( (2, 4) )
dizi
Output:

[[0.58402638 0.35442156 0.0298901  0.66604586]
 [0.86611044 0.27354201 0.65279721 0.2384682 ]]
```

2 satırlı, 4 sütunlu rastgele tam sayılardan oluşan iki boyutlu bir dizi oluşturma

```
# 0 ve 10 arası tam sayı üretelim

dizi = np.random.randint(0, 10, size=(2, 4))
```

```
Output:

[[3 7 0 0]
 [5 1 6 1]]
```

Dizilerde indis verilerine ulaşma:

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

# İlgili dizinin ilk satırını döndürme

dizi[0]
```

```
Output: [1 2 3]
```

```
# İlgili dizinin ikinci satırını döndürme

dizi[1]
```

```
Output: [4 5 6]
```

Reshape bir dizinin şeklini değiştirmek demektir. Dizinin her boyutundaki elemanların sayısını aşağıdaki gibi belirleyebiliriz.

```
dizi = np.arange(8)

dizi = dizi.reshape(4, 2)

dizi

# yukarıdaki kodlar çalıştırıldığında 4 satırlı ve 2 sütunlu aşağıdaki gibi bir dizi elde edilir.
```

```
Output:

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

Bu dizi referans alınarak aşağıdaki işlemler yapılabilir:

```
# dizinin ilk iki satırını döndürme

dizi[0 : 2]
```

Output:

```
[[0 1]
 [2 3]]
```

Dizinin ilk sütununu döndürme. Virgülün solu satırları, sağı ise sütunları belirler. Bu örnekte tüm satırlar seçilmiş ancak sadece sıfırıncı indisteki sütunlar seçilmiştir.

```
dizi[ : , 0]
```

Output: [0 2 4 6 8]

Dizinin üçüncü satırının ikinci değerini döndürme

```
dizi[2, 1]
```

Output: 5

Matris oluşturma:

```
# 5 satır ve 3 sütundan ve sıfırlardan oluşan matris örmeği

dizi= np.zeros((5, 3))

dizi
```

Output:

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Üç boyutlu sıfırlardan oluşan matris örneği: Burada, (3, 1, 2) tuple'ı, matrisin boyutlarını belirtir. İlk eleman 3, yani matrisin 3 satırı olacak. İkinci eleman 1, yani matrisin sadece 1 sütunu olacak. Son eleman 2, yani her bir hücrenin 2 derinliği olacak.

```
dizi= np.zeros((3, 1, 2))

dizi
```

Output:

```
[[[0. 0.]]

 [[0. 0.]]

 [[0. 0.]]]
```

Üç boyutlu 1'lerden oluşan matris örneği:

dizi= np.ones((2,3,4))

Output:

```
[[[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]]
```

Birim matris oluşturma örneği: x boyutlu bir birim matriste, köşeden çapraz olan değerler birlerden, diğer değerler ise sıfırlardan oluşur. Aşağıdaki örnekte x değeri 3 olarak verilmiştir.

dizi = np.eye(3)

Output:

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Matris birleştirme:

matris1 = np.array([[1, 2, 3] , [4, 5, 6]])

matris2 = np.array([[7, 8, 9] , [10, 11, 12]])

birlesme = np.concatenate([matris1, matris2])

print(birlesme)

Output:

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Matris birleştirmede eğer 'axis' değeri kullanırsa matrisler yan yana diziliş gösterir. Yukarıdaki kodumuzu şu şekilde yazalım ve çıktımızı görelim:

```
birlesme = np.concatenate([matris1,matris2] , axis = 1)
print(birlesme)
```

```
Output:

[[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]
```

Bir dizinin maximum değerini bulma:

```
dizi = np.array([ [1, 2, 3, 4] ,
                  [ 5, 6, 7, 8] ,
                  [9, 10, 11, 12]  ])
dizi.max()
```

```
Output: 12
```

Bir dizinin minumum değerini bulma:

```
dizi = np.array([ [1, 2, 3, 4] ,
                  [ 5, 6, 7, 8] ,
                  [9, 10, 11, 12]  ])
dizi.min()
```

```
Output: 1
```

Bir dizideki tüm değerlerin toplamını bulma:

```
dizi = np.array([ [1, 2, 3, 4] ,
                  [ 5, 6, 7, 8] ,
                  [9, 10, 11, 12]  ])
dizi.sum()
```

```
Output: 78
```

Bir dizideki tüm satırların toplamını bulma:

```
dizi = np.array([ [1, 2, 3, 4] ,
                  [ 5, 6, 7, 8] ,
                  [9, 10, 11, 12]  ])
dizi.sum(axis = 1)
# axis değeri 1 olursa satırların toplamını alır
Output: [10 26 42]
#Burada birinci satırda bulunan sayıların yani 1, 2, 3 ve 4'ün toplamı
10    , aynı şekilde diğer satırların toplamı 26 ve 42'dir.
```

Bir dizideki tüm sütunların toplamını bulma:

```
dizi = np.array([ [1, 2, 3, 4] ,
                  [ 5, 6, 7, 8] ,
                  [9, 10, 11, 12]  ])
dizi.sum(axis = 0)
# axis değeri 0 olursa sütunların toplamını alır
Output: [15 18 21 24]
#Burada birinci sütunda bulunan sayıların yani 1, 5 ve 9'ün toplamı 15,
aynı şekilde diğer sütunların toplamı 18, 21 ve 24'tür.
```

Bir dizideki tüm değerlerin ortalamasını bulma:

```
dizi = np.array([ [1, 2, 3, 4] ,
                  [ 5, 6, 7, 8] ,
                  [9, 10, 11, 12]  ])
dizi.mean()
Output: 6.5
```

Bir dizideki varyansı bulma:

```
dizi = np.array([ [1, 2, 3, 4] ,
                  [ 5, 6, 7, 8] ,
                  [9, 10, 11, 12]  ])
dizi.var()
```

```
Output: 11.916666666666666
```

Soru: Varyans nedir?

Cevap: "Varyans" istatistiksel bir terimdir ve bir veri kümesindeki değerlerin ortalamadan ne kadar sapma gösterdiğini ölçer. Yani varyans, bir veri kümesindeki her bir verinin ortalamadan ne kadar uzaklaştığına bağlı olarak veri dağılımının ne kadar geniş olduğunu ölçer. Varyans, bir veri kümesindeki her bir verinin ortalamadan ne kadar farklı olduğunu ölçmek için kullanılan kareli sapma değerlerinin toplamının n-1'e bölünmesi ile hesaplanır. Burada n, veri kümesindeki toplam örneklemlerin sayısını temsil eder. Varyansın karekökü standart sapma denir ve varyansın ölçüm birimi ile aynı birimde ifade edilir. Daha düşük bir varyans değeri, veri kümesindeki örneklerin ortalamaya daha yakın olduğu anlamına gelirken, daha yüksek bir varyans değeri, örneklerin ortalamadan daha fazla uzaklaştığı anlamına gelir.

Bir dizideki standart sapmayı bulma:

```
dizi = np.array([ [1, 2, 3, 4] ,
                  [ 5, 6, 7, 8] ,
                  [9, 10, 11, 12]  ])
dizi.std()
```

```
Output: 3.452052529534663
```

Soru: Standart sapma nedir?

Cevap: Standart sapma, bir veri kümesindeki değerlerin ortalamadan ne kadar sapma gösterdiğini ölçen bir istatistik terimidir. Standart sapma, veri kümesindeki her bir örneğin ortalamadan ne kadar farklı olduğunu hesaplamak için kullanılır. Standart sapma, veri kümesindeki örneklemlerin her birinin ortalamadan farkının karelerinin toplamının, örneklemlerin sayısının bir eksiğiyle bölümünden hesaplanır. Bu hesaplamayla, standart sapma, veri kümesindeki örneklerin ne kadar yayıldığını, ne kadar değişken olduğunu ölçer. Standart sapma, varyansın karekökü olarak hesaplanabilir. Varyans, standart sapmanın karesidir ve standart sapma varyansın karekökü'dür. Standart sapma, varyans gibi, veri kümesinin ölçüm birimi ile aynı birimde ifade edilir. Bir veri kümesindeki standart sapma daha yüksekse, veri kümesindeki örneklerin ortalamadan daha fazla sapma gösterdiği anlamına gelir ve veri kümesinin daha değişken olduğunu ifade eder. Standart sapma daha düşükse, veri kümesindeki örneklerin ortalamaya daha yakın olduğu ve veri kümesinin daha homojen olduğu anlamına gelir.

Ek bilgi: Diziler arasında matematiksel işlemler gerçekleşebilir ancak, dizilerin boyutu ve şekli birbiri ile uyumlu olmalıdır. Yani aynı satır ve sütun sayılarına sahip olmalıdır.

Farklı dizileri toplama: İki farklı dizimiz olsun. Bu dizilerdeki karşılıklı satır ve sütunları toplayalım. Aşağıdaki örnekte dizi1'deki 1 değeri, dizi2'deki 7 değerini denk geleceği için bu ikisi toplanır ve 8 değerine ulaşılır. Diğer değerler de aynı şekilde toplanır.

```
dizi1 = np.array([ [1, 2, 3] ,
                   [ 4, 5, 6] ])
dizi2 = np.array([ [7, 8, 9] ,
                   [ 10, 11, 12] ])
dizi1 + dizi2
```

```
Output:
 [[ 8 10 12]
  [14 16 18]]
```

Farklı dizileri çıkarma: Farklı bir örnekle dizi2 değerlerini dizi1 değerlerinden çıkaralım.

```
dizi1 = np.array([ [3, 2, 4] ,
                   [ 7, 7, 9] ])
dizi2 = np.array([ [12, 45, 32] ,
                   [ 10, 32, 56] ])
dizi1 - dizi2
```

```
Output:
 [[ 9 43 28]
  [ 3 25 47]]
```

Farklı dizileri çarpma: Farklı bir örnekle dizi1 değerlerini dizi2 değerleri ile çarpalım.

```
dizi1 = np.array([ [3, 2, 4] ,
                   [ 3, 8, 5] ])
dizi2 = np.array([ [9, 4, 7] ,
                   [ 1, 2, 6] ])
dizi1 * dizi2
```

```
Output:
 [[27  8 28]
  [ 3 16 30]]
```

Farklı dizileri bölme: Farklı bir örnekle dizi2 değerlerini dizi1 değerlerine bölelim. Sonuçların float olarak çıktığına dikkat edin.

```
dizi1 = np.array([ [3, 2, 4] ,
                   [ 3, 8, 5] ])

dizi2 = np.array([ [9, 4, 8] ,
                   [ 6, 16, 35] ])

dizi1 / dizi2
```

```
Output:
 [[3. 2. 2.]
  [2. 2. 7.]]
```

Farklı diziler arası modül işlemi: Modül işlemi bilindiği üzere bölümlerden kalan sayıyı verir. Bunu aşağıdaki dizi değerleri arasında gösterelim.

```
dizi1 = np.array([ [3, 2, 4] ,
                   [ 7, 7, 9] ])

dizi2 = np.array([ [12, 45, 32] ,
                   [ 10, 32, 56] ])

dizi2 % dizi1
```

```
Output:
 [[0 1 0]
  [3 4 2]]
```

Bir dizi ile bir sayının toplanması: Aşağıdaki işleme göre dizi1'in her değerine 10 değeri eklenir.

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

dizi + 10
```

```
Output:
 [[11 12 13]
  [14 15 16]]
```

Ek bilgi: Bir dizi ile bir sayı nasıl toplanabiliyorsa diğer tüm işlemler de yapılabilir.

Dizilerde trigonometrik işlemler: Dizilerde trigonometrik işlemler de yapılabilir. Aşağıdaki örnekte sinüs alma işlemi gösterilmiştir:

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

np.sin(dizi)
```

```
Output:
 [[ 0.84147098  0.90929743  0.14112001]
  [-0.7568025  -0.95892427 -0.2794155 ]]
```

Dizilerde karekök ve diğer işlemler: Yukarıda gösterilen trigonemetrik işlemi haricinde karekök gibi işlemler de yapılabilir. Bunun için yapmamız gereken sadece sin yerine sqrt komutu yazmaktır. Bunun haricinde exp (e sayısı), log (logaritma) gibi işlemler de aynı yöntemle yapılır.

Matrisleri çarpma: İki matrisin çarpımı numpy.dot() işlevi kullanılarak hesaplanır. Ancak, matrislerin boyutlarına dikkat etmek önemlidir. Örneğin ilk matrisin sütun sayısı ikinci matrisin satır sayısına eşit olmalıdır.

```
dizi1 = np.array([ [3, 2,] , [ 7, 8] ])

dizi2 = np.array([ [9, 4] ,  [ 1, 5] ])

np.dot(dizi1, dizi2)
```

```
Output:
   [[29, 22]
   [71, 68]]
```

Yukarıdaki örnekte matris çarpımı şu şekilde hesaplanır:

İlk satır ve ilk sütun için:

(3*9) + (2*1) = 29

İlk satır ve ikinci sütun için:

(3*4) + (2*5) = 22

İkinci satır ve ilk sütun için:

(7*9) + (8*1) = 71

İkinci satır ve ikinci sütun için:

(7*4) + (8*5) = 68

Ek bilgi: Matris çarpma işlemi için @ işareti de kullanılabilir. Python 3.5 ve üstü sürümlerde geçerli olan bu işaretin Jupyter notebook'da da etkin olabilmesi için bir hücreye aşağıdaki kod yazılır ve çalıştırılır.

```
%config IPCompleter.greedy=True
```

Tranpozisyon işlemi: Transpozisyon işlemi, bir dizi ya da matrisin sütun ve satırlarının yerlerini değiştirir. Yani bir matrisin transpozu, orijinal matrisin sütunlarının satırlara ve satırların sütunlara yer değiştirmiş halidir.

NumPy'da transpozisyon işlemi, transpose() fonksiyonu ile yapılır. Bu fonksiyon, bir dizinin boyutlarını tersine çevirir.

Örneğin, bir 2D dizi (matris) için, transpose() işlemi, sütunları satırlara ve satırları sütunlara yer değiştirir. Örnek:

```
import numpy as np
# Örnek bir matris tanımlayalım
matris = np.array([[1, 2], [3, 4], [5, 6]])
# Matrisin transpozunu alalım
transpoz = matris.transpose()
# Matris ve transpozunu yazdıralım
print("Matris:\n", matris)
print("Transpoz:\n", transpoz)
```

```
Output:
Matris:
 [[1 2]
  [3 4]
  [5 6]]
Transpoz:
 [[1 3 5]
  [2 4 6]]
```

Ek bilgi:

```
transpoz = matris.transpose()   # yerine;
transpoz = matris.T             # şekilnde de yazabiliriz.
```

Dizi ve matrislerde Boolean işlemleri: Koşullu işlemler yapıp dizimizdeki değerlerin doğruluğunu kontrol edip True veya False çıktıları alabiliriz.

```
 # bir dizi oluşturalım
dizi= np.array([[1, 2], [3, 4], [5, 6]])
# kontrol adında bir değişken oluşturup koşulumuzu belirleyelim
kontrol = dizi < 4
# doğruluk değişkenini yazdıralım
print(kontrol)
```

```
Output:
[[ True  True]
 [ True False]
 [False False]]
```

Ek bilgi: Sadece koşulu sağlayan değerleri dizi haline getirebiliriz. Yukarıdaki örneği tekrar baz alırsak;

```
 kosul_degerleri= dizi[dizi < 4]
print(kosul_degerleri)
```

```
Output:
[1, 2, 3]
```

.npy Dosya Formatı:

NumPy dizileri diskte depolamak ve okumak için kullanılabilen bir dosya formatı olan .npy dosyalarını destekler.

.npy dosyaları, NumPy dizilerinin sıkıştırılmamış binary formatta kaydedildiği bir dosya formatıdır. Bu dosyalar, NumPy tarafından sağlanan **save()** ve **load()** fonksiyonları kullanılarak oluşturulur ve okunur. Bu dosyalar, yüksek performanslı veri depolama ve aktarımı sağlar.

Ek bilgi: NumPy ayrıca, .npz uzantılı sıkıştırılmış dosyalar da destekler. Bu dosyalar, birden fazla .npy dosyasını içerebilir ve sıkıştırılmış bir arşiv olarak saklanabilir. Bu dosyaları oluşturmak ve okumak için de NumPy tarafından sağlanan savez() ve loadz() fonksiyonları kullanılabilir.

```
# Örnek bir dizi oluşturalım.
dizi = np.array([1, 2, 3, 4, 5])
# Diziyi "deneme.npy" adlı bir dosyaya kaydedelim
np.save("deneme.npy", dizi)
# "deneme.npy" dosyasındaki verileri yeni_dizi adlı değişkene yükleyelim.
yeni_dizi = np.load("deneme.npy")
# Yüklenen diziyi ekrana yazdıralım
print(yeni_dizi)
```

```
Output:
[1 2 3 4 5]
```

Bir .csv uzantılı dosyanın içindeki verileri çekme: Numpy, .csv dosyalarını okumak için numpy.genfromtxt() fonksiyonunu kullanabilirsiniz. Bu fonksiyon, bir .csv dosyasını numpy dizisi olarak okur. Örneğin, aşağıdaki sayilar.csv adındaki örnek bir dosyanın içindeki verileri kullanalım.

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

```
# dosyamızı bir değişkene atayalım.
dosya_yolu = "ornek.csv"
# dosyamızı ilgili fonksiyonu kullanarak veri adlı değişkene aktaralım.
veri = np.genfromtxt(dosya_yolu, delimiter=",")
# delimiter parametresi, dosyanın nasıl ayrıldığını belirtir. Burada , karakteri kullanıldı.
print( veri)
```

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