NUMPY

Bu Defter Hakkında

- Defter, öğrendiğim bilgilerden, gördüğüm eğitimlerden ve NumPy dökümantasyonundan derlenmiştir.
- Defterin kapsamı, pek çok işlem için kâfîdir; fakat NumPy farklı amaçlar için de kullanılan bir kütüphâne olduğundan (misal, kriptolojik rastgele sayı üretimi) defter kimi durumlar için yetersizdir; belki ileride yine genişletme çalışması yapabilirim.
- NumPy matrisleri yazdırıldığında elemanlar arasında boşluk olur; fakat virgül olmuyor. Daha iyi anlaşılması için ekran çıktısına virgül ekledim; gerçekte virgül konulmuyor, yanıltmış olmayayım.
- Yazar : Mehmet Akif SOLAK

Genel Bilgiler

- NumPy matris ve vektörler üzerinde matematiksel işlemleri kolayca yapabilmek için tasarlanmış bir yardımcı kütüphânedir.
- İşlemlerin performansını arttırma, esnek işlem kâbiliyyeti ve yardımcı fonksiyonlarla veri biliminde çokça tercih edilen bir kütüphânedir.
- Ücretli bir yazılım olan MatLab yazılımının yerine Python'ın kullanılmasını sağlayan bir kütüphânedir.
- BSD lisansıyla sunulan bir açık kaynak kütüphânedir.
- pip ile yüklemek için `pip install numpy` kodu çalıştırılabilir.
- Anaconda ile yüklemek için `conda install numpy` çalıştırılmalıdır.
- NumPy kütüphânesinin temel yapıtaşı dizilerdir. Diziler Python listelerine benzemektedir; fakat farklıdır. Tek boyutlu NumPy dizisinin sınıfı `numpy.array`'dir; çok boyutlu NumPy dizilerinin (aslında matrislerin) ismi ise `ndarray`'dir.
- NumPy dizilerinin indis kullanımı Python'daki gibi esnektir; hemen hemen aynıdır.
- Kütüphânenin sürüm bilgisi `numpy.__version__` kodu ile öğrenilebilir.

1) Numpy Dizisi oluşturma

- Python listesiyle dizi oluşturmak için ilgili sınıfa Python listesini verin:

```
import numpy as nm
veri = [234, 334, 34]
numpyDizisi = nm.array(veri)
print(numpyDizisi)# array([234, 334, 34])
print(type(numpyDizisi))# <class 'numpy.ndarray'>
```

- Çok boyutlu diziyi de bu şekilde oluşturabilirsiniz; veri tipini ayarlayabilirsiniz:

```
matrix = [[234, 4535, 5], [45, 6, 66]]
numMatrix = nm.array(matrix)
boolMatrix = nm.array(matrix, dtype = 'bool')
# Dilerseniz diğer matris üzerinden kolayca oluşturabilirsiniz:
boolMatrix = numMatrix.astype('bool')
```

- Dizi oluşturmak için NumPy'ın fonksiyonlarını kullanabiliriz:

1. `arange(baslangic, bitis, atlama)` : Verilen aralıktaki sayılardan bir dizi oluşturmak için kullanılır:

```
dizi = nm.arange(0, 5)# [0, 1, 2, 3, 4]
dizi = nm.arange(0, 10, 2)# [0, 2, 4, 6, 8]
dizi = nm.arange(0, 10, 3)# [0, 3, 6, 9]
```

2. `zeros(uzunluk, ...)` : Verilen boyut ve uzunlukta sıfır matrisi döndürür:

```
nm.zeros(3)# [0, 0, 0]
nm.zeros((2, 4))# 2 boyutlu, 3 uzunluklu sıfır matrisi
# [[0, 0, 0, 0]
# [0, 0, 0, 0]]
```

3. `ones(uzunluk, ...)` : Verilen boyut ve uzunlukta birler matrisi oluşturur:

```
nm.ones(3)# [1, 1, 1]
nm.ones((2, 4))# 2 boyutlu, 3 uzunluklu bir birler matrisi
# [[1, 1, 1, 1]
# [1, 1, 1, 1]]
```

4. `linspace(baslangic, bitis, adet)` : 'baslagic' ve 'bitis' arasında eşit mesafede 'adet' kadar değişken içeren dizi döndürür bi iznillah. Üçüncü girdi opsiyoneldir; eğer girilmezse varsayılan olarak 50 alınır.

```
nm.linspace(0, 15, 3)# [0, 7.5, 15]
```

5. `eye(boyut)` : Verilen 'boyut' boyutunda birim matris üretir:

```
nm.eye(3)
# [[1, 0, 0],
# [0, 1, 0],
# [0, 0, 1]]
```

6. `random.randn(sayi)` : 'sayi' uzunluğunda karışık bir dizi döndürür. İkinci bir girdi verilerek karışık float sayılardan oluşan bir matris elde edilebilir:

```
nm.random.randn(3)# [-0.0275184 , -0.89346649,  0.39125001]
```

7. `randint(altSinir, ustSinir, adet)` : 'ustSinir' dahil olmamak üzere verilen aralıktan rastgele seçilmiş 'adet' tâne tam sayıdan oluşan bir dizi döndürür. Alt sınır verilmezse sıfır atanır, adet verilmezse bir tâne tam sayı döndürür:

```
nm.random.randint(0, 5, 2)# [2, 4]
```

2) İndis işlemleri ve NumPy dizilerinin bâzı farkları

- NumPy dizilerinin indis işlemleri Python'daki gibi esnektir. Bâzı kullanım örnekleri:

```
dizi[2:5]# dizinin 2'den 5'e kadar elemanlarını getirir
dizi[:-2]# dizinin dondan ikinciye kadar olan elemanları getirir
dizi[:2]# dizinin ilk iki elemanı getirir
```

- numPy dizilerinin Python'daki listelerden bir farkı tek bir atama ile belirtilen tüm elemanlara aynı değeri atayabilmemizdir:

```
dizi[0:3] = 0# ilk üç elemana sıfır değerini atar
dizi[:] = 2# Tüm dizi elemanlarının değeri 5 yapılır
```

- NumPy dizilerinin başka bir farkı ise indis işlemlerinin sütun bazında da yapılabiliyor olmasıdır:

```
matris = nm.array([[2, 4, 5, 7], [3425, 45, 6346, 11]])
d = matris[0:,3]# Her satırın son elemanını yeni dizi olarak döndürür
print(d)# [7, 11]
```

- İstenilen satırlar şu şekilde alınabilir : `matris[[satirNo,satirNo2,..]]`
- Liste üzerinde basitçe filtreleme yapılabilir: `matris > 5` : Matriste 5'ten büyük sayıların yerine `True` yazılmış; diğerlerine `False` yazılmış bir ndarray matrisi döndürür.
- **!!!** : Python dizilerinde diziler toplama işâretiyle toplandığında diziler peş peşe eklenir; NumPy dizilerinde ise dizilerin karşılıklı elemanları birbirleriyle toplanır:

```
d1, d2 = [2, 45], [442, 4]
nmD1, nmD2 = nm.array(d1), nm.array(d2)
print((d1 + d2))# [2, 45, 442, 4]
print((nmD1 + nmD2))# [444, 49]
# Bu kural çarpma işlemi için de geçerlidir
```

- Matrisler üzerinde işlem yaparken yazım kolaylığı olması açısından ikinci indisi virgül koyarak yazabilirsiniz; bu özellik Python matrislerinde yoktur. Şu iki ifâde birbirine eşittir : `matris[0][2]` ve `matris[0, 2]` Her iki ifâde de matrisin ilk satırının 3. elemanını getirir
- NumPy dizilerinde indisler içerisinde filtreleme yapılabilir:

```
matris = nm.array([2, 253, 11])
filtreMatrisi = matris > 10
print(type(filtreMatrisi))# <class 'numpy.ndarray'>
print(filtreMatrisi)# [False, True, True]
filtrelenmisMatris = matris[filtreMatrisi]# [253, 11]
```

3) NumPy Dizilerinin Metotları

- NumPy dizilerinde bir diziden bir bölüm alıp, o bölüm üzerinde yaptığımız değişiklik o diziye kaydediliyor. Demek ki atama yapmaya çalıştığımızda referans ataması yapılıyor; yeni bir değişken üretilmiyor. Bundan kaçınmanın yolu dizinin `copy()` yöntemini kullanmaktır.
- `copy()` : NumPy dizileri 'değişebilir' ('mutable') yapıdadır ve referans bazlı çalışır. Dolayısıyla bir NumPy dizisinin bâzı elemanlarını indisler aracılığıyla başka bir değişkene

atasanız ve üzerinde değişiklik yapsanız, bu değişiklikler diğer NumPy dizisine de yansır. Böyle olmaması için dizinin referansını değil, değerini kopyalamanız lazım; bu yöntem de bu işe yarıyor.

- `mean(axis=None, dtype=None)` : Dizideki değerlerin ortalamasını döndürür. Eğer ortalamayı sütun bazlı hesaplamak istiyorsak, yanî her sütun için kendi değerlerini toplayıp, sütun uzunluğuna bölünmüş hâlini istiyorsak, `axis` parametresine `0` değeri vermeliyiz; satırların ortalamasını istiyorsak `axis`'e `1` değerini vermeliyiz.

- `max(axis=None)` : Dizi içerisindeki en büyük elemanı getirir. Eksen belirtilirse, o eksene göre hesâplama yapılır. `0` değeri her sütundaki değerlerin en büyüklerinin alınarak oluşturulmuş diziyi döndürür, `1` ise her satırın..:

```
data=nm.array([[235, 5, 6],[15,17,56],[167,73,11]], dtype="uint8")
print(data.max())# 167
print(data.max(axis=0))# [167  73  56]
print(data.max(axis=1))# [  6  56 167]
```

- `min(axis=None)` : Dizi içerisindeki en küçük elemanı getirir; kullanımı `max()` gibidir.

- `argmax(axis=None)` : Dizi içerisindeki en büyük elemanın sıra numarasını getirir.

- `argmin(axis=None)` : Dizi içerisindeki en küçük elemanın sıra numarasını getirir.

- `reshape(boyut1, boyut2)` : Diziyi verilen boyutlarda bir matrise dönüştürür; değişiklikler diziye kaydedilmez, yeni bir dizi olarak döndürülür.

- `astype(veriTipi)` : Dizinin elemanlarını farklı bir veri tipine çevirmek için kullanılır. Burada verilecek parametre hedef veri tipinin sınıfı olabilir. Hedef veri tipi sınıfı `numpy.dtypes` veyâ Python veri tipi ismi olabilir. Bunun yerine Python veri tipi sınıfının ismini verebilirsiniz:

```
import numpy as nm
d = nm.array([[45, 11], [34, 51]])
print(d.dtype)# int32
castedArray = d.astype('float32')
print(castedArray)
# [[45, 11]
# [34, 51]]
# Şu şekilde de yapılabilir:
castedArray = d.astype(nm.dtypes.Float32DType)
```

- `copy()` : Aynı dizinin bir kopyasını oluşturur (bellekte kopyalar, yeni nesnenin referansı farklı olur)
- `diagonal()` : Matrisin sol üstten başlayan köşegen değerlerini döndürür.
- `dot(value)` : Verilen değer ile matrisi çarpar; verilen değer bir sayı ise skaler çarpım uygulanır; verilen değer bir matris ise nokta çarpımı yapılır.
- `fill(value)` : Matrisi verilen değerle doldurur.
- `cumsum()` : Elemanları toplayarak uzunlukta tek boyut dizi oluşturur ve döndürür:

```
import numpy as nm
d = nm.array([[45, 11], [4, 5]])
print(d.cumsum())#[45, 56, 60, 65]
```

- `flatten()` : Diziyi tek boyutlu bir dizi hâline getirir, düzleştirir.
- `repeat(sayi)` : Matrisi düzleştirir, elemanlar verilen 'sayi' - 1 kadar kendisinin peşine tekrarlanır; yanî kendisiyle berâber verilen 'sayi' kadar yer alır:

```
print(d.repeat(2))# [45 45 11 11 4 4 5 5]
```

- `reshape((boyut, uzunluk))` : Matrisin verilen boyut ve uzunluk şekline getirilmiş biçimi döndürülür; eğer verilen biçim bilgisi uygun değilse, hatâ verilir. Matrisin aslı üzerinde değişiklik yapılmaz. Eğer matrisin tek boyutlu olması isteniyorsa yalnızca uzunluk bilgisi verilebilir: `d.reshape((1, 4))` = `d.reshape(4)`
- `resize((boyut, uzunluk))` : Matrisin boyutu ve uzunluğu verilen biçime getirilir; değişiklik matrisin aslına yansır; geriye bir şey dönmez. Kullanımı `reshape()` ile aynıdır.
- `round()` : Kesirli sayı barındıran matrislerde verileri yuvarlar; fakat bu değişiklik asıl matrise yansıtılmaz; geriye değerleri yuvarlanmış matris döndürülür. **Önemli bir bilgi**, matrisin değeri 0.5 olduğunda yukarı yuvarlanmaz; fakat 0.51 olduğunda yukarı yuvarlanır.
- `sort(axis = -1, kind = None, order = None)` : Matrisin her satırı kendi içerisinde küçükten büyüğe yuvarlanır. `axis` parametresine `0` değeri verilirse sütuna göre sıralanır.
- `std()` : Verilerin standart sapmasını döndürür.
- `sum(axis = None)` : Elemanları verilen eksen değerine göre toplar, değişiklik asıl matrise yansıtılmaz. Eğer hiç parametre verilmezse matristeki tüm elemanların toplamını döndürür; fonksiyonun başka parametreleri de vardır:

```
d = nm.array([[34, 15], [3, 11]])
print(d.sum(1))# [49 14]
print(d.sum(0))# [18 45]
```

- `take(indices)` : Verilen indis değerlerine göre eleman getirir. Tek bir sayı verildiğinde verilerin düzleştirilmiş hâlindeki sıraya göre eleman getirilir; yanî verilen indis satırların yan yana dizilmesi sonucu kaçıncı eleman ise o getirilir:

```
d = nm.array([[34, 15], [3, 11]])
print(d.take(2))# 3
d.take([0, 1, 1])# [34, 15, 15]
```

- `tolist()` : Dizinin Python `list` tipinde bir nesne hâlini döndürür.
- `transpoze()` : Matrisin transpozunu döndürür.
- `trace()` : Sol üstten başlayan köşegenin toplamını döndürür
- `tofile(fid, sep='', format='%s')` : Matrisi verilen dosya yoluna kaydeder. İlgili yolda bir dosya yoksa oluşturulur; varsa üzerine yazılır. İlk parametre bir dosya, dosya ismi, bir Path değişkeni veyâ dosya yolu metni olabilir; sadece dosya ismi verildiğinde o an çalışılan dizine kaydedilir. Veri varsayılan olarak byte formatında kaydedilir. `sep` parametresi verilerin metîn olarak kaydedildiği durumda elemanların arasına koyulacak işârettir:

```
d = nm.array([[34, 15], [3, 11]])
d.tofile('diziSimdi.txt', sep=' ')
# Dosya içeriği: 1 34 3 11
d.tofile('diziSimdi.txt', sep=',')
# Dosya içeriği: 1,34,3,11
d.tofile('diziSimdi.txt')# Veri, ikili (binary) formatta kaydedilir
```

- ..

> ***NOT :*** `len()` fonksiyonu dizi için çalıştırıldığında dizinin eleman sayısını döndürür; matris için çalıştırıldığında matrisin boyut sayısını döndürür

> ***NOT :*** `argmax()` fonksiyonu matris için çalıştırıldığında matris elemanlarının baştan itibâren sayılı dizi elemanları gibi olduğu durumda ilgili eleman kaçıncı sıradaysa o döndürülür. Misal, 3 uzunluklu 2 boyutlu bir matriste en büyük eleman 2. satırın 2. elemanıysa `argmax()` yöntemi geriye 5 döndürür.

> ***NOT :*** Buradaki fonksiyonların hemen hemen hepsi numpy kütüphânesi üzerinden de çağrılabilir; bu durumda fonksiyonun ilk parametresi ilgili dizi olmalıdır:

```
d = nm.array([[34, 15], [3, 11]])
nm.max(d)# 34
nm.sum(d)# 63
nm.argmax(d)# 0
```

> ***NOT :*** Buradaki metotların bâzısı matrisin aslı üzerinde değişiklik yaparken, bâzısı matrisin üzerinde değişiklik yapılmış hâlini döndürür; asıl matrisi değiştirmez. Metotları kullanırken buna dikkat ediniz.

> ***NOT :*** Numpy dizilerinin metîn olarak kaydedilmesi ve yüklenmesi için yeni sürümde `savetxt()` ve `loadtxt()` metodları vardır.

> ***NOT :*** Numpy'ın `diag()` fonksiyonu sol üstten başlayan köşegen değerlerini almak için kullanılır; eğer köşegenin başlangıç sütunu kaydırılmak isteniyorsa k parametresi pozitif değer olarak verilir; eğer satır indisi belirtilmek isteniyorsa k parametresine negatif değer verilir:

```
d = nm.array([[34, 11, 5], [16, 4, 165], [53, 16, 66]])
print(nm.diag(d))# [34  4 66]
print(nm.diag(d, k=1))# [ 11 165]
print(nm.diag(d, k=-1))# [16 16]
print(nm.diag(d, k=-2))# [53]
```

- ..

4) NumPy Dizilerinin Alanları (Özellikleri, 'fields')

- `size` : Dizinin eleman sayısını verir.
- `shape` : Dizinin boyut ve uzunluk bilgisini demet (tuple) biçiminde verir.
- `ndim` : Dizinin boyut bilgisini döndürür; misal matrisler 2 boyutludur.
- `dtype` : Dizinin veri tipini `numpy.dtypes` kitâplığının bir nesnesi olarak döndürür.
- `base` : NumPy dizisi başka bir numpy dizisinin alt dizisi ise, asıl diziyi döndürür. Bu, referans tipli çalışmayla alâkalıdır:

```
import numpy as nm
li = [[34, 14, 11], [15, 55, 16]]
d = nm.array(li)
if d.base is None:
    print("d dizisi sıfırdan oluşturulmuş")
sub = d[:1]
if sub.base is d:
    print("sub dizisi d dizisi kullanılarak oluşturulmuş")
```

- `data` : Dizi verisinin bellekteki adresini döndürür.
- `T` : Matrisin transpozudur.
- `itemsize` : Dizinin her elemanının bellekte kaç byte yer tuttuğu bilgisini döndürür.
- `nbytes` : Dizinin bellekte kapladığı alanı byte cinsinden döndürür.
- `strides` : Bellekte dizinin sonraki elemanına geçmek için kaç byte atlanması gerektiği bilgisini verir. Matrislerde geriye dönen değer demettir. İlk değer bir satırdan diğerine geçmek için gereken kaydırma miktârını ifâde ederken, ikinci değer aynı satırda bir sonraki hücreye geçmek için gereken kaydırma miktârını ifâde eder.

5) NumPy'da Veri Tipleri

- NumPy dizisi oluşturulurken veri tipi otomatik olarak tespit edilir; fakat istersek elle de verebiliriz.
- NumPy dizisinin veri tipini öğrenmek için dizinin `dtype` özelliğine erişebiliriz.
- NumPy'da veri tipleri `numpy.ndtypes` kitâplığı altında bulunmaktadır.
- Numpy'da veri tipi birkaç farklı şekilde belirtilebilir. Birincisi sınıf referansı ile, ikincisi kısa isim ile, üçüncüsü ise `numpy` altındaki tip sâbiti ile:

```
import numpy as nm
# Şu üç satır da aynı işlevi yapmaktadır:
d = nm.array([1, 5, 3], dtype = nm.dtypes.Float16DType)
d = nm.array([1, 5, 3], dtype = nm.float16)
d = nm.array([1, 5, 3], dtype = "float16")
```

- NumPy veri tipleri çok çeşitlidir; fakat temel veri tiplerinin anlaşılması zor değildir.

Tamsayı Veri Tipleri

- Tam sayı veri tipleri işâretli ve işâretsiz olmak üzere 2; farklı bellek boyutu kaplamasına göre 4 çeşittir. Bu sebeple 8 farklı tamsayı veri tipi vardır.
- Tam sayı veri tiplerinin sınıf isimleri ve kısa isimleri şu düzenli ifâde şeklindedir: Sınıf ismi: `[U]Int[8|16|32|64]DType` Kısa isim: `[u]int[8|16|32|64]` Bu yazımda köşeli parantez içerisinde olanlar tercihi olan seçeneklerdir. En baştaki `U` ve `u` `unsigned` kelîmesinin baş harfini yansıtır; sayının işâretsiz olduğunu ifâde eder. İşâretsiz sayılar 0'dan başladığı için pozitif değer aralığı aynı boyuttaki işâretsiz sayının pozitif değer aralığının iki katıdır. Bu işâret yazılmazsa sayı işâretlidir. Sonraki tercihî kısım olan `[8|16|32|64]` kısmı ise sayının bellekte kapladığı alanı belirtmek içindir. Bu 4 değer (8, 16, 32, 64)den birisi yazılmazsa varsayılan olarak 32 geçerlidir.
- Tamsayının değer aralığı bellekte kapladığı alana göre değişkenlik arz etmektedir. İşâretli sayılar için : $[(-2^{byte})/2, ((2^{byte} - 1)/2) - 1$ İşâretsiz sayılar için : $[0, 2^{byte} - 1]$
- Aynı veri tipi farklı isimlerle de ifâde edilebilmektedir. Misal, `Int8DType` veri tipi `ByteDType` ile veyâ 'byte' kısa ismiyle ifâde edilebilir; `Int16DType` ile `ShortDType` aynıdır; bunların kısa isimleri 'int16' ve 'short' da aynıdır.

Kesirli Sayı Veri Tipleri

- Kesirli sayı veri tipleri bellek boyutu bakımından 16, 32 ve 64 olmak üzere 3 çeşittir.
- Kitâplık altındaki sınıf isimleri ve kısa isimleri şu düzenli ifâdedeki gibidir:

  Sınıf ismi : `Float[16|32|64]DType` Kısa isim : float[16|32|64]
- Bâzı veri tipleri farklı isim ve sınıflarla da ifâde edilebilir. Misal `Float64DType` ('float64') veri tipi `LongDoubleDType` ('longdouble') veri tipiyle aynıdır.

Metîn Veri Tipleri

- Metîn veri tipleri temelde `StrDType` sınıfıyla ifâde edilse de, metnin uzunluğuna göre değişkenlik arz etmektedir:

```
s = ['aa', 'bb', 'cc', 'dd']
nS = nm.array(s, dtype=nm.dtypes.StrDType)
print(nS.dtype)#<U2
```

- Yukarıdaki kodun çıktısı `<U2` bize verinin 'unicode' karakterler için 2 ve daha kısa uzunluklu veriler için uygun olduğunu söylüyor. Böyle bir durumda biz veriye daha uzun bir değer atamak istersek, verinin sadece ilk iki karakteri saklanır:

```
nS[0] = 'AkşamSabah'
print[nS]# ['Ak' 'bb' 'cc' 'dd']
```

- Metnin azamî karakter sayısı arttırılabilir:

```
nS = nS.astype('U16')
print(nS.dtype)#<U16
nS[0] = 'AkşamSabah'
print[nS]# ['AkşamSabah' 'bb' 'cc' 'dd']
```

- Eğer elinizdeki verilerin en fazla kaç uzunluklu olduğunu bilmiyorsanız, tüm verinizi bir listede topladıktan sonra numpy dizisini başta gösterildiği gibi oluşturmalısınız.

> ***NOT :*** Bunların dışında `bool` değişkenler için, karmaşık sayılar için, zamân veri tipleri için de numpy veri tipleri vardır.

Veri Tipleri Tablosu

Sınıf	Metîn olarak	Boyut	Değer aralığı
Int8DType	int8	1 byte	[-128, 127]
ByteDType	byte	1 byte	[-128, 127]
Int16DType	int16	2 byte	[-32768, 32767]
ShortDType	short	2 byte	[-32768, 32767]
Int32DType	int32	4 byte	[-2,147,483648, 2,147,483647]
IntDType	int	4 byte	[-2,147,483648, 2,147,483647]
LongDType	long	4 byte	[-2,147,483648, 2,147,483647]
Int64DType	int64	8 byte	[-9,223,372,036,854,775808, 9...07]
LongLongDType	longlong	8 byte	[-9,223,372,036,854,775808, 9...07]
Float16DType	float16	2 byte	[-65500.0, 65500.0]
Float32DType	float32	4 byte	[-3.4028235e+38, 3.4028235e+38]
Float64DType	float64	8 byte	[-1.7976931348623157e+308, 1.7976931348623157e+308]
LongDoubleDType	longdouble	8 byte	[-1.7976931348623157e+308, 1.7976931348623157e+308]
UInt8DType	uint8	1 byte	[0, 255]
UByteDType	ubyte	1 byte	[0, 255]
UInt16DType	uint16	2 byte	[0, 65536]
UShortDType	ushort	2 byte	[0, 65536]
UInt32DType	uint32	4 byte	[0, 4,294,967296]
UIntDType	uint	4 byte	[0, 4,294,967296]
ULongDType	ulong	4 byte	[0, 4,294,967296]
UIn64DType	uint64	8 byte	[0, 18,446,744,073,709,551616]
ULongLongDType	ulonglong	8 byte	[0, 18,446,744,073,709,551616]
StrDType	str	değişken	Metîn verisi, uzunluğu değişken

Sınıf	Metîn olarak	Boyut	Değer aralığı
BoolDType	bool	1 byte	[False, True]
BytesDType	bytes	değişken	İkili (binary) veri, uzunluğu değişken
Complex64DType	complex64	8	[-3.4028235e+38, 3.4028235e+38]
Complex128DType	complex128	16	[-1.7976931348623157e+308, 1.7976931348623157e+308]
CLongDoubleDType	clongdouble	16	[-1.7976931348623157e+308, 1.7976931348623157e+308]

- Tablodakiler dışında `DateTime64DType` veri tipi târih saat değeri tutmak için, `TimeDelta64DType` veri tipi birimi belirtilmiş iki zamân arasındaki farkı tutmak için kullanılır. Her ikisinin de farklı hassâsiyet çeşitleri vardır.
- ..

6) Rastgele Sayı Üretici Kitâplık : numpy.random

- `numpy.random` kitâplığı rastgele sayı üretme işlevini üstlenir.
- Rastgele sayı üretmek için bâzı yöntemler:
  - `random.random(size=None)` : Hiçbir parametre verilmezse rastgele bir sayı döndürür. Eğer tek bir sayı verilirse, verilen adet kadar rastgele sayıdan oluşuan bir dizi döndürülür; eğer demet şeklinde (boyut, uzunluk) biçimi verilirse, o biçim şeklinde rastgele bir dizi döndürülür:

```
import numpy as nm
print(nm.random.random())# 0.0799815312516513
print(nm.random.random(2))# [0.32859396 0.09735729]
print(nm.random.random((3, 2))
# [[0.34366302 0.32437284]
# [0.88283441 0.72611467]
# [0.43546157 0.61959615]]
```

  - `random.randn(d0, d1, d2, ..., dn)` : Normal dağılımlı rastgele sayı üretir. Parametre verilmediğinde bir sayı üretir. Tek bir sayı verildiğinde verilen sayı adedince rastgele sayı üretir. Çok boyutlu dizi üretilmesini istiyorsak, boyut ve uzunluk sırasıyla **ayrı parametreler olarak** olarak verilmelidir (demet biçiminde verilmemelidir):

```
import numpy as nm
print(nm.random.randn())# -0.7184102550573718
print(nm.random.randn(2)# [-0.54339332 1.65435453]
print(nm.random.randn(3, 2))# Boyut ve uzunluk ayrı parametre!
# [[0.46870054 0.10157999]
# [0.7033265  0.11584723]
# [0.2955605  0.70789846]]
```

- `random.randint(low, high = None, size = None, dtype = int)` : Verilen değer aralıklarında rastgele tamsayı üretir. Tek parametre verilirse o parametre üst sınır olur. Alt sınır değer aralığına dâhildir, üst sınır dâhil değildir. Farklı veri tiplerinde dönüş almak için `dtype` parametresini kullanın:

```
import nmupy as nm
print(nm.random.randint(35, 55, 3))# [45 36 51]
d = nm.random.randint(35, 55, 3)
print(d.dtype)# int32
d = nm.random.randint(35, 55, 3, dtype='int16')
print(d.dtype)# int16
d = nm.random.randint(35, 55, 3, dtype='short')
print(d.dtype)# int16
```

- `random.choice(a, size=None, replace=True)` : Verilen diziden verilen miktar adedince eleman seçilir. Dizi tek boyutlu olmalıdır. `replace` parametresine `False` değeri verilirse aynı eleman yeniden seçilmez. Eğer seçmek istediğiniz eleman sayısı dizinin eleman sayısından fazla ise ve `replace=False` parametre değerini verdiyseniz hatâ alırsınız.

Rastgele Sayı Üreteci

- Dilerseniz kendi sayı üretecinizi oluşturabilirsiniz:

```
rGen = nm.random.default_rng(seed = 11)# seed = Tohum değeri
print("Üretilen sayı - 1 : ", rGen.random())# 0.12857020276919962
print("Üretilen sayı - 2 : ", rGen.random())# 0.49927786244011496
# Aynı tohum değeriyle yeni nesne oluşturursanız, aynı sonuç çıkar:
rGen2 = nm.random.default_rng(seed = 11)# seed = Tohum değeri
print("Üretilen sayı - 1 : ", rGen2.random())# 0.12857020276919962
print("Üretilen sayı - 2 : ", rGen2.random())# 0.49927786244011496
```

- Rastgele sayı üretecinin metotları:

- `random(size=None, dtype='float64')` : 0 - 1 aralığında verilen biçimde verilen veri tipinde rastgele sayı üretir; veri tipi kesirli 'float32' veyâ 'float64' olmalıdır; varsayılan olarak 8 byte'lık bir kesirli sayı döndürüldüğünden veri tipini belirtmek pek çok durumda iyi olabilir. `size` parametresine bir sayı verildiğinde, bu değer uzunluk olarak yorumlanır ve verilen miktar adedince rastgele sayıdan oluşan bir boyutlu dizi döndürülür. Eğer matris üretilmek isteniyorsa, matris boyut ve uzunluğu bir demet şeklinde verilmelidir:

```
rGen = nm.random.default_rng(seed = 11)# seed = Tohum değeri
print(rGen.random(size=3))# [0.13380122, 0.1285702 0.79708064]
print(rGen.random(size=(4, 2), dtype='float32')
# [[0.49927783, 0.5900328],
# [0.6014983, 0.7121722],
# [0.02868897, 0.48550326],
# [0.14792603, 0.40149254]]
```

- `integers(low, high = None, size = None, dtype = 'int64', endpoint = False)` : Tam sayı üretir. `low` parametresi taban değerini, `high` parametresi tavan değerini, `size` parametresi uzunluk veyâ biçim (boyut, uzunluk) değerini, `dtype` parametresi üretilen sayının veri tipini, `endpoint` parametresi tavan değerinin değer aralığına dâhil olup olmayacağını ifâde eder:

```
d = rGen.integers(2, 66, 4, dtype='int8', endpoint=True)
# [2, 66] aralığında 1 byte boyutunda 4 tamsayıdan oluşan
# tek boyutlu dizi döndürülür.
print(d)# [21 53 18 10]
rGen.integers(4)# [0, 4) aralığında bir tam sayı üretilir
d = rGen.integers(4, 16, size=(2, 3), dtype='int8')
print(d)
# [[ 9,  7, 15],
# [10, 10, 14]]
```

- `normal(loc=0.0, scale=1.0, size=None)` : Gauss normal dağılımlı rastgele kesirli sayı üretir.
- `uniform(loc=0.0, scale=1.0, size=None)` : Normal dağılımlı kesirli sayı üretir.
- `poisson(lam=1.0, size=None)` : Poisson dağılımına sâhip tamsayı üretir.
- `shuffe(x, axis=0)` : Verilen x dizisinin elemanlarını karıştırır; varsayılan olarak sütun bazında karıştırma yapar. Satır bazında karıştırma yaptırmak için `axis` parametresine `1` değeri verilmelidir.

- `choice(a, size=None, replace=True, p=None, axis=0, shuffle=True)` : `a` dizisinden verilen parametrelere uygun olarak eleman seçme uygulanır. `size` parametresine 2 verilirse 2 satır verisi çekilir. `axis` parametresine `1` değeri verilirse sütun seçilir.

- `standard_normal()` : Ortalaması sıfır, standart sapması 1 olan normal dağılımlı rastgele dizi, matris oluşturmak için kullanılır.

- Bu kitâplığı bir rastgele sayı üreteci kullanmadan rastgelelik tohumuyla besleyerek aynı tohumla aynı sayıların üretilmesini sağlayabilirsiniz:

```
nm.random.seed(seed=13)
number = nm.random.randn(50)
```

- ..

7) Kullanışlı NumPy Fonksiyonları

- NumPy dizisi metotları dışında olan kullanışlı bâzı `numpy` fonksiyonları:

  - `unique(array)` : Dizi içerisindeki değerleri münferid (tekil) olarak döndürür.

  - `std(a, axis = None, dtype = None, )` : Verilen dizinin standart sapmasını hesaplar. Eğer `axis` değerine `1` verilirse, sırasıyla her satırın standart sapma değeri, `0` verilirse sırasıyla her sütunun standart sapma değeri döndürülür.

  - `flip(array)` : Verilen dizinin elemanlarının ters çevrilmiş hâlini döndürür. Eğer dizi matris ise, düzleştirildiğinde hangi sıradaysa, onun tersi sıralama geri döndürülür.

  - `square(x, dtype=None)` : Verilen değerin karesini döndürür; verilen değer bir dizi ise her elemanın karesini alır; `dtype` dışında başka parametreleri de vardır; fakat çoğu durumda bu iki parametreyi bilmek kâfî olabilir. `dtype` parametresi belirtilmezse, verilen dizi veyâ sayının veri tipinde geri dönüş yapılır. Bâzen ilgili değerin karesi o değerin veri tipinin değer aralığını aşabilir; bu durumda hatâ almazsınız; fakat yanlış sonuç alırsınız:

```
d = nm.array([[34,  1,  6], [17,  6,  6]], dtype=nm.int8)
print(nm.square(d))
# [[-124, 1, 36],
# [33, 36, 36]]
# Satırların ilk elemanları yanlış!
```

- `sum(a, axis = None)` : Verilen değerlerin toplamını döndürür. `axis = 0` için sütunların kendi içinde toplamı, `axis = 1` için satırların toplamı sırasına uygun olarak döndürülür.

- `sqrt(a)` : Verilen değerin karekökünü döndürür; eğer dizi verilirse tüm elemanların karekökünün alınmış hâlini barındıran yeni dizi döndürür.

- `unique(a)` : Verilen dizinin münferit (tekil) elemanlarını tek boyutlu dizi olarak döndürür.

- `sin(), cos(), tan(), sinh(), cosh(), tanh()` : Bunlar sırasıyla, sinüs, kosinüs, tanjant, hiperbolik sinüs, hiperbolik kosinüs, hiperbolik tanjant fonksiyonlarını ifâde eder. Dikkat edilmesi gereken nokta **verilen değerin radyan cinsinden olması** gerektiğidir. NumPy'da dereceyi radyana çeviren yardımcı fonksiyonlar vardır.

- `rad2deg(radianValue)` : Radyan değerini dereceye çevirir.

- `deg2rad(degreeValue)` : Açı değerini radyana çevirir.

- `pad(array, pad_width, mode='constant')` : Verilen dizinin çevresinde verilen `pad_width` miktarınca 0 doldurur ve bu yeni diziyi döndürür (değişiklikler asıl diziye yansımaz). `mode` ise boşlukların nasıl doldurulacağını ifâde eder:

```
d = nm.array([[ 2,  3], [ 4, 22]])
print(nm.pad(d, 2))
# [[ 0  0  0  0  0  0]
# [ 0  0  0  0  0  0]
# [ 0  0  2  3  0  0]
# [ 0  0  4 22  0  0]
# [ 0  0  0  0  0  0]
# [ 0  0  0  0  0  0]]
print(nm.pad(d, (2,1)))# Sol ve üstte 2, sağ ve altta bir boşluk
# [[ 0  0  0  0  0]
# [ 0  0  0  0  0]
# [ 0  0  2  3  0]
# [ 0  0  4 22  0]
# [ 0  0  0  0  0]]
print(nm.pad(d, ((1, 1), (1, 2))))# Sağda 2, diğerlerinde 1 boşluk
# [[ 0  0  0  0  0]
# [ 0  2  3  0  0]
# [ 0  4 22  0  0]
# [ 0  0  0  0  0]]
padded = nm.pad(d, ((1, 4), (2, 3)))#Üst:1, alt:4, sol:2, sağ:3

# Hangi değerle doldurulacağı seçilebilir:
print(nm.pad(d, 1, constant_values=1))
# [[ 1  1  1  1]
# [ 1  2  3  1]
# [ 1  4 22  1]
# [ 1  1  1  1]]
# Doldurma değerleri mevkîye göre atanabilir:
print(nm.pad(d, 1, constant_values=((17, 14), (1, 5))))
# [[ 1 17 17  5]
# [ 1  2  3  5]
# [ 1  4 22  5]
# [ 1 14 14  5]]
```

- `concatenate(a1, a2, axis=0)` : `a1` ve `a2` dizilerini verilen eksene göre birleştirir. Eksen (`axis`) parametresine `0` verilirse -ki varsayılan değer budur- a2 dizisi, a1'in altına eklenir; yanî satır düzeyinde ekleme / birleştirme yapılır. `axis` parametresine `1` verilirse a2 dizisi, a1'in yanına sütun olarak eklenir.

```
d = nm.array([[ 2,  3], [4, 22]])
e = nm.array([[34, 15], [674, 756]])
print(nm.concatenate([d, e], axis=1))
# [[  2   3  34  15]
# [  4  22 674 756]]
print(nm.concatenate([d, e], axis=0))
# [[  2   3]
# [  4  22]
# [ 34  15]
# [674 756]]
```

- `expand_dims(x, axis=0)` : Verilen diziye eksen ekler. `axis=0` için yeni eksen satır düzeyini ifâde eder; yanî verilerin her biri yeni bir sütun olur. `axis = 1` için eksen, sütun düzeyini ifâde eder; yanî verilerin her biri yeni bir satır olur. Matris biçimindeki bir veri genişletilirken `axis` parametresi `2` yapılırsa yeni eklenen eksen z katmanını ifâde eder. Z katmanının sonda ifâde edilmesi yaygın bir kullanımdır:

```
data=nm.array([[235, 5, 6],[15,17,56],[167,73,11]], dtype="uint8")
print(data.shape)#(3, 3)
data = nm.expand_dims([data], axis=2)
print(data.shape)#(3, 3, 1)
```

- `squeeze(a, axis=None)` : Tek elemanı olan boyutları kaldırır; yanî eksen siler. Misal yukarıdaki örnek için bu fonksiyonu çalıştırırsak son katmanı silerek, verinin biçimini (3,3) hâline getirir; fakat değiştirilen veri döndürülür, değişiklik asıl matrise yansıtılmaz. Eğer birden fazla tek elemanlı katman varsa onları da siler. Misal, (3,3,1,1) biçimindeki veriyi (3,3) biçimine getirir; böyle olması istenmiyorsa, `axis` parametresine silinmek istenen katmanın indisi verilir.

- `moveaxis(a, source, destination)` : Eğer bir eksendeki veriyi başka eksene taşımak istiyorsak bu fonksiyonu kullanabiliriz. `a` parametresi diziyi, `source` parametresi taşınmak istenen eksenin indisini, `destination` parametresi eksenin taşınmak istendiği indisi ifâde eder:

```
data=nm.array([[235, 5, 6],[15,17,56],[167,73,11]], dtype="uint8")
data = nm.expand_dims(data, axis=2)
print(data.shape)# (3, 3, 1)
data = nm.moveaxis(img, 2, 0)
print(data.shape)# (1, 3, 3)
```

- `nonzero(a)` : Aldığı dizi içerisinde, sıfırdan farklı elemanların indislerini bir dizi biçiminde döndürür. Eğer diziniz çok boyutlu ise dizinizi önce düzleştirmeniz sonucun daha anlaşılır olmasını sağlar.

- `diag_indices(n, ndim=2)` : Sol üstten sağ alta doğru ilerleyen köşegenin indislerini elde etmek için kullanılır. Bu, köşegen üzerindeki verileri değiştirmek gibi durumlarda kullanılabilir. `n` kenar uzunluğunu `ndim` ise boyutu belirtir:

```
data=nm.array([[235, 5, 6],[15,17,56],[167,73,11]], dtype="uint8")
indicesOfDiagonalLine= nm.diag_indices(3)
print(indicesOfDiagonalLine)# (array([0, 1, 2]), array([0, 1, 2]))
data[indicesOfDiagonalLine] = 1
print(data)
# [[ 1   5   6]
# [ 15   1  56]
# [167  73   1]]
```

- `intersect1d(ar1, ar2, assume_unique=False, return_indices=False)` : Verilen iki dizi arasında aynı mevkîde aynı değer barındıranların değerlerini almak için kullanılır; istenilirse `return_indices` parametresine `True` verilerek indisleri de alınabilir.

- `negative(a)` : Verilen sayı veyâ dizinin negatifini döndürür.

- `add(x1, x2, dtype=None)` : Verilenleri toplar. Eğer dizi verilirse karşılıklı olarak aynı indisteki değerleri toplar.

- `divide(x1, x2)` : Bölme işlemi uygular.

- `trunc(x)` : Verilen değerin / dizinin kesirli kısmınının silinmiş hâlini döndürür.

- `abs(x)` : Verilen değerin / dizinin mutlak değerini döndürür.

- `isnan(x)` : Verilen değerin / dizinin geçerli bir sayı olup, olmadığını kontrol eder. `np.nan` olan değerlerin olduğu yere `True` atanır.

- `tile(A, reps)` : Verilen `A` değeri / dizisi `reps` adet tekrarlanır; oluşan dizi döndürülür. Eğer dizi verildiyse, her satırdaki değerler, satır bittikten sonra aynı sırayla tekrarlanır.

- `matmul(x1, x2)` : `x1` ve `x2` dizisi bu sırayla matris çarpımı işlemine sokulur; çıkan sonuç döndürülür. Matris çarpımı, skaler çarpım işleminden farklıdır. Matris çarpımında, `x1`'in sütun sayısı `x2`'nin satır sayısıyla eşit uzunlukta olması gerektiğinden, boyut uyuşmazlığı durumunda hatâ alınır.

- `swapaxes(a, axis1, axis2)` : `a` dizisindeki iki eksenin yerini değiştirmek için kullanılır; `moveaxis()` ile aynı sonucu üretir.

- `insert(arr, obj, values, axis=None)` : Verilen diziye `obj` ile verilen indisten itibâren `values` ile verilen değerleri ekler. `axis` ile hangi eksene veri eklenmek istendiği belirtilir; eğer eksen belirtilmezse biçimi değişmiş dizi döndürülür:

```
d = nm.array([[345, 51, 5], [15, 66, 76]], dtype='int16')
print(d)
# [[345, 51, 5],
# [15, 66, 76]]

# Yeni bir satır olarak ekleme (1 indisli satır olur):
print(nm.insert(d, 1, [34, 15, 15], axis=0))
# [[345  51   5]
# [ 34  15  15]
# [ 15  66  76]]

# Yeni bir sütun olarak ekleme (2 indisli sütun olur):
print(nm.insert(d, 2, [34, 15], axis=1))
# [[345  51  34   5]
# [ 15  66  15  76]]

# En sona yeni satır olarak eklemek istersek:
print(nm.insert(d, len(d), [34, 15, 15], axis=0))

# En son sütun olacak şekilde yeni sütun olarak eklemek istersek:
print(nm.insert(d, d.shape[1], [34, 15, 15], axis=0))
```

- ..

> ***NOT :*** Bâzı fonksiyonların yazılandan çok daha fazla parametresi vardır; yaygın kullanıma veyâ genel ihtiyaca göre kullanılması düşünülen parametreler zikredilmiştir.

- Bunların dışında ileri seviye işlemler için `tensordot()`, `einsum()` gibi fonksiyonlar vardır. Bunlar, tensor çarpımı, tensor toplanması için kullanılan fonksiyonlardır.

8) DİĞER

- NumPy dizisi oluşturulurken verilen veri varsayılan olarak kopyalanır; yanî NumPy dizisi oluşturmak için kullanılan liste üzerinde yapılan değişiklik NumPy dizisine yansıtılmaz; bunun olmamasını istiyorsak, misal çok büyük bir matrisin olduğu durumda bu matrisin kopyalanmamasını istiyorsak dizi oluştururken `copy` parametresine `False` değerini vermeliyiz.

- `numpy`'da sayı olmayan veriler (`None`) `numpy.nan` ile ifâde edilir.
- Matematiksel işlemlerde sayının hesaplanamaz durumda olduğu durumlar olabilir. Bu durumda hatâ oluşmaması için `numpy`'da sonsuz sayıyı simgeleyen bir tip vardır: `numpy.inf`
- NumPy'ın yapılandırmasını görmek için şunu çalıştırın : `numpy.show_config()`
- NumPy pratik kodlama biçimlerini öğrenmek sıkıcı ve gereksiz olabilir; fakat çok ihtiyaç duyulan ve tek adımda yapılmayan bâzı kodlama pratiklerini bilmek iyi olabilir. Bunlardan bir tânesi, resme yeni bir kanal eklemek. Resme yeni bir kanal eklemek için önce resme z ekseni eklenir (eğer yoksa) ve sonra yeni verisi ana veriye eklenir:

```
# 3x3'lük tek kanallı resmimiz olduğunu düşünelim.
# Ana verimiz 3x3 boyutunda, 8 bit kodlamalı bir matris:
img=nm.array([[13, 14, 7], [54, 21, 57], [11, 25, 16]],dtype=nm.uint8)
newChannel = nm.array([[5, 6, 16],[51, 1, 6],[6, 61,9]],dtype='uint8')

# Verilerin biçim bilgisi:
print("Resim biçimi:", img.shape,"\nKanal biçimi:", newChannel.shape)
# Resim biçimi: (3, 3)
# Kanal biçimi: (3, 3)

# Verilere z ekseni eklemeliyiz:
img = nm.expand_dims(img, axis=2)
newChannel = nm.expand_dims(newChannel, axis=2)

# Yeni biçim bilgisi:
print("Resim biçimi:", img.shape,"\nKanal biçimi:", newChannel.shape)
# Resim biçimi: (3, 3, 1)
# Kanal biçimi: (3, 3, 1)
# Katman sırası: x, y, z

# Resimleri birleştirmek için concatenate fonksiyonu kullanılabilir:
newImg = nm.concatenate([img, newChannel], axis=2)# Z ekseni = 2

# Yeni boyut bilgisi:
print("Yeni resmin boyut bilgisi:", newImg.shape)
# Yeni resmin boyut bilgisi: (3, 3, 2)
```

- NumPy'da resimlerle uğraşırken genellikle sonuncu katman kanal katmanı olarak kullanılır; veriyi parçalara ('batch') ayırdığımızda ise en baştaki katman parçaları ifâde eder. Bu biçim zorunlu değildir; fakat hangi katmanın neyi ifâde ettiği dikkat edilmesi gereken bir noktadır.

- `numpy`'da târih ve zamân ile ilgili pek çok işlem yapılabilir; veri tipinin târih - zamân hassasiyeti ayarlanabilir; misal, `np.datetime64('today')` kodu bugünün târihini döndürür; bu tür yardımcı özellikler kullanılabilir; fakat `numpy` genellikle sayısal veriler için kullanıldığından bu konulara değinilmemiştir.

- Numpy'da sınırları zorlamak ve düşük seviye işlemlerle ilgili bilgi edinmek için şu adresi ziyâret edebilirsiniz: https://www.labri.fr/perso/nrougier/from-python-to-numpy/