

## ▼ NumPy

- NumPy **array** yapısı ile çalışmamıza olanak tanıyan bir Python kütüphanesidir.

### Neden NumPy Kullanırız?

- NumPy array yapısı bellek yönetimini listelere göre daha hızlı yapar.
- Liste yapısından daha kullanışlıdır.
- Listeler üzerindeki birçok farklı boyutlu işlemi gerçekleştirmemizi sağlar.

NumPy kütüphanesini bilgisayarımıza yükledikten sonra geliştirme ortamımıza bunu **import** ile dahil ederiz.

```
import numpy
```

NumPy'ın özelliklerini daha hızlı ve daha etkili bir şekilde kullanabilmek için **np** kısaltması ile çağırabiliriz.

```
import numpy as np
```

Versiyon kontrolünü **version** özelliği ile yapabiliriz.

```
import numpy as np  
  
print(np.__version__)
```

NumPy ile array objesi oluşturma işlemi **ndarray** dediğimiz, n dimensional array ifadesinin kısaltılmış hali ile yapılır.

Arrayler ile ilgili en çok dikkat etmemiz gereken şey boyut kavramıdır.

```
import numpy as np  
  
arr = np.array([15, 91, 37, 68, 34])  
  
print(arr)  
  
print(type(arr))
```

Arrayleri sadece `[ ]` kullanarak değil tuple(demet) yapısı ile de oluşturabiliriz.

```
import numpy as np
```

```
arr = np.array((15, 91, 37, 68, 34))  
  
print(arr)
```

## Arraylerde Boyut Kavramı

*Nested array*: Array yapısını eleman olarak içinde barındıran arraylere verilen addır.

İç içe geçmiş array yapısında 0, 1, 2 ya da 3 boyutlu arraylerimiz olabilir.

Arrayin kaç boyutlu olduğuna arraylerin içindeki en atomik seviyedeki arrayi bularak karar verebiliriz.

1D Array

3	2
---	---

2D Array

1	0	1
3	4	1

3D Array

1	7	9
5	9	3
7	9	9

### ▼ 0-D Arrayler

```
import numpy as np  
  
arr = np.array(42)  
  
print(arr)
```

### ▼ 1-D Arrayler

```
import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5])  
  
print(arr)
```

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

## ▼ 2-D Arrayler

```
import numpy as np

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

print(arr)
```

## ▼ 3-D Arrayler

```
import numpy as np

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

print(arr)
```

## ▼ Arraylerimizin Kaçıncı Boyutta Olduğunu Nasıl Bileceğiz?

**ndim** özelliği kaçıncı boyutta olduğumuzu görmemize olanak sağlar.

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

## ▼ Arrayleri Oluştururken Boyutları Nasıl Belirleyeceğiz?

**ndmin** özelliği ile oluşturduğumuz arrayin boyutunu belirleyebiliriz.

```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)
```

```
print(arr)
print('number of dimensions :', arr.ndim)
```

## Challenge Zamanı 🏆

Beyin fırtınası yapalım!

Tek mi çift mi?

-Bir fonksiyon oluşturacağız.

-Fonksiyon listedeki elemanların toplamı olan sayının tek mi çift mi olduğunu bulacak.

-Adı **tek\_cift** olacak.

-Fonksiyon yalnızca iki değer döndürecek. *TEK* ya da *ÇİFT*

**İpucu** ▲: Önce listedeki sayıların toplamına sahip olmamız gerekir.

## ▼ NumPy Array Indexleme

Indexleme işlemi yaparken, indexlerin 0'dan başladığını unutmamamız gerekir.

```
import numpy as np

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

print(arr[0])
```

Indexleme işleminde, listelerin elemanlarına ulaşırız ve bu elemanlarla toplama işlemi yapabilmemiz mümkündür.

```
import numpy as np

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

print(arr[2] + arr[3])
```

İki boyutlu arrayler üzerinde boyutlar arasında da bir index işlemi vardır.

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

Aynı işlemi 3 boyutlu arrayler üzerinde de gerçekleştirebiliriz.

```
import numpy as np

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

print(arr[0, 1, 2])
```

## ▼ NumPy Array Slicing

Slicing işlemi, indexler üzerinden arraylerin belli bir kısmını 'dilimleyebileceğimiz' bir işlemdir.

Indexleme işleminin yapısı şu şekildedir: [başlangıç:son]

Indexleme işlemi adım atlayarak yapmak istersek şu yapıda olacaktır: [başlangıç:son:adım]

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])
```

## ▼ Negative Slicing

Slicing işlemi arrayin sonundan gerçekleştirmek istediğimizde negative slicing işlemi yapabiliriz.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```

## ▼ NumPy Array Iterating

Elemanlar üzerinde yaptığımız gezinme işlemine iterating denir. Bunu daha öncesinde aşına olduğumuz döngü yapısıyla yapabiliriz.

```
import numpy as np

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

for x in arr:
    print(x)

1
2
3
```

Aynı gezinme işlemi 2 boyutlu arrayler üzerinde de uygulayabiliriz. Ancak indexlere ve boyutlara dikkat etmemiz gerekir.

```
import numpy as np

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

for x in arr:
    print(x)
```

İç içe döngüler yazarak boyut içine girebilir ve iki boyutlu bir array içinde önce arraye sonra da elemanlarına ulaşarak elemanlar üzerinde gezinti yapabiliriz.

```
import numpy as np

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

for x in arr:
    for y in x:
        print(y)

import numpy as np

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

for x in arr:
    for y in x:
        for z in y:
            print(z)
```

## ▼ Skaler Elemanlar Üzerinde Gezinmek

NumPy kütüphanesi sayesinde **nditer()** fonksiyonuyla en basit işlemlerden en karmaşık işlemlere iteration yapabiliriz.

Yaptığımız diğer işlemlerden farkı ise en atomik ya da daha net bir ifadeyle skaler elemanlar üzerinde geziniyor olmamızdır.

```
import numpy as np

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):
    print(x)
```

[Colab'ın ücretli ürünleri](#) - Sözleşmeleri buradan iptal edebilirsiniz

