

```
# from IPython.display import Image
from numpy.random import randn
import numpy as np
```

▼ The Numpy ndarray

- 다차원 배열 객체
- [] 2차원 [[]] 3차원

```
# numpy 안의 랜덤 randn
```

```
data = randn(3,3) # 2행 3열
data
```

```
array([[ -0.89447894,  1.97766009, -1.78146971],
       [-0.04059574,  0.16287911,  0.3650751 ],
       [-1.3989726 ,  1.22993693,  0.95777659]])
```

```
data*10
```

```
array([[ -8.94478943,  19.77660093, -17.81469713],
       [-0.40595743,   1.62879107,   3.65075099],
       [-13.98972596,  12.29936932,   9.57776592]])
```

```
data+data
```

```
array([[ -1.78895789,  3.95532019, -3.56293943],
       [-0.08119149,  0.32575821,  0.7301502 ],
       [-2.79794519,  2.45987386,  1.91555318]])
```

```
data.shape
```

```
(3, 3)
```

```
data.dtype
```


```
dtype('float64')
```

▼ Creating ndarrays

```
data1 = [5, 7.5, 8, 0, 1] #리스트타입
arr1 = np.array(data1)    #array 타입으로 변환 int형으로 변환됨
arr1
```

```
array([5. , 7.5, 8. , 0. , 1. ])
```


```
data2 = [[1,2,3,4],[5,6,7,8]]
arr2 = np.array(data2)
arr2
```

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


```
arr2.ndim    #dimend = 몇 차원인지
```

 `2`


```
arr2.dtype
```

 `dtype('int32')`

```
np.zeros((3,6))
```

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

```
np.empty((2,3,2))    # 2개 / 3행 2열  
#np.empty((4,3,2))  # 4개 / 3행 2열
```

 `array([[[1.18078783e-311, 3.16202013e-322],
 [0.00000000e+000, 0.00000000e+000],
 [1.33511969e-306, 5.40294136e-066]],


 [[1.85390169e-051, 8.78045605e-071],
 [3.99623187e+175, 2.03957991e+184],
 [5.55527260e+170, 7.35369136e-043]])])`

```
np.arange(1,15)
```


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

▼ Data Types for ndarrays


```
arr1 = np.array([1,2,3], dtype=np.float64)  
arr1.dtype
```

 `dtype('float64')`


```
arr2 = np.array([1,4,5], dtype=np.int32)  
arr2.dtype
```

 `dtype('int32')`


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

 dtype('int32')


```
arr_1 = arr.astype(np.float64)    # .astype(np.변경할문자형)으로 변경된 결과 복사
arr_1.dtype
```

 dtype('float64')

```
arr = np.array([4.9,-2.8,3.3,1.0,56.3,-88])
arr
```

 array([4.9, -2.8, 3.3, 1. , 56.3, -88.])


```
arr_2 = arr.astype(np.int32)
arr_2
```

 array([4, -2, 3, 1, 56, -88])


▼ [Tip] astype

astype을 호출하면 새로운 dtype이 이전 dtype과 같아도 항상 새로운 배열을 생성(데이터를 복사)한다. float64나 float32 같은 부동소수점은 근사값이라는 사실을 염두에 두는게 중요하다.

```
# astype를 사용 숫자로 변환
arr.astype(np.int32)
```

 array([4, -2, 3, 1, 56, -88])


```
numeric_strings=np.array(['1.25','-9.5','33'],dtype=np.string_)
numeric_strings
```

 array([b'1.25', b'-9.5', b'33'], dtype='<S4')

```
numeric_strings.astype(float)
```

 array([1.25, -9.5 , 33.])


```
int_array = np.arange(10)
int_array
```

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


```
calibers = np.array([.22,.270,.355,.142],dtype=np.float64)
calibers
```

 `array([0.22 , 0.27 , 0.355, 0.142])`

```
int_array.astype(calibers.dtype)
```


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

```
# u4는 uint32와 동일
empty_uint32 = np.empty(8, dtype='u4')
empty_uint32
```


 `array([3264175145, 1070344437, 343597384, 1070679982, 3951369912,
1071036497, 1443109011, 1069690126], dtype=uint32)`

▼ 배열과 스칼라 간의 연산


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

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


```
arr+arr
```

 `array([[2., 4., 6.],
[8., 10., 12.]])`


```
arr-arr
```

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

```
1/arr
```


 `array([[1. , 0.5 , 0.33333333],
[0.25 , 0.2 , 0.16666667]])`

```
arr**0.5
```

 `array([[1. , 1.41421356, 1.73205081],
[2. , 2.23606798, 2.44948974]])`

▼ 색인과 슬라이싱


```
arr = np.arange(10)
arr
```

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

```
arr[5]
```

 5

```
arr[5:9]
```

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

```
arr[5:9] = 10
```


```
arr
```

 `array([0, 1, 2, 3, 4, 10, 10, 10, 10, 9])`

```
arr_slice = arr[5:9]    # arr의 5~8까지 자르는  
arr_slice
```

 `array([10, 10, 10, 10])`

```
arr_slice[1] = 12345    #arr_slice의 1번째  
arr
```

 `array([0, 1, 2, 3, 4, 10, 12345, 10, 10,
 9])`

```
arr_slice[:] = 64  
arr
```


 `array([0, 1, 2, 3, 4, 64, 64, 64, 64, 9])`

[Tip] 슬라이스 복사본

만약에 뷰 대신 ndarray 슬라이스의 복사본을 얻고 싶다면 `arr[5:8].copy()`를 사용해서 명시적으로 배열을 복사하면 된다.

▼ 2차원 배열

```
arr2d = np.array([[1,2,3],[4,5,6],[7,8,9]])  
arr2d[2]
```

 `array([7, 8, 9])`

```
arr2d[0,2]
```

 3

```
arr2d[0][2]
```


 3

```
arr2d[2,1]
```

 8


▼ 3차원 배열

```
arr3d = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])  
arr3d
```



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

```
arr3d[0,1]
```




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

```
arr3d[0,0,0]
```


 1

```
arr3d[0] = 42  
arr3d
```




```
array([[[42, 42, 42],  
        [42, 42, 42]],  
       [[ 7,  8,  9],  
        [10, 11, 12]]])
```

```
arr3d[0,1] = 10  
arr3d
```




```
array([[[42, 42, 42],  
        [10, 10, 10]],  
       [[ 7,  8,  9],  
        [10, 11, 12]]])
```

```
arr3d[0] = np.array([[1,2,3],[4,5,6]])
arr3d
```


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

[[7, 8, 9],
[10, 11, 12]])

```
old_values = arr3d[0].copy()
old_values
```

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

```
arr3d[0] = old_values
arr3d
```

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


[[7, 8, 9],
[10, 11, 12]])

▼ 슬라이드 색인


```
arr
```

 array([0, 1, 2, 3, 4, 64, 64, 64, 64, 9])


```
arr[1:6]
```

 array([1, 2, 3, 4, 64])

```
arr2d
```

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

```
arr2d[1,:2]
```

 array([4, 5])


```
arr2d[:2]
```




```
arr2d[2,:1]
```

 array([7])

```
arr2d[:, :1]
```


 array([[1],
[4],
[7]])

```
arr2d[:2,1:] =0  
arr2d
```

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


▼ 불리언 색인(인덱싱)

```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])  
names
```


 array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')

```
data = randn(7,4)
```


```
data
```

 array([[-1.00766818, -2.26320999, 0.23523663, -0.39924381],
[0.7793744 , 0.05219855, 0.58482829, 0.64349462],
[0.76974409, -2.27627079, 0.02299811, -0.69820179],
[3.07552784, -0.54626755, 0.41046104, -0.39563456],
[1.26317167, -0.22958675, -1.14526231, -1.95480864],
[-1.30555998, 0.3463475 , -1.78786867, -0.39672916],
[-1.0031312 , -1.73947791, -1.04028283, -0.27457652]])

```
data[names == 'Will']
```

 array([[0.76974409, -2.27627079, 0.02299811, -0.69820179],
[1.26317167, -0.22958675, -1.14526231, -1.95480864]])

```
data[names == 'Will', 2:]
```

 array([[0.02299811, -0.69820179],
[-1.14526231, -1.95480864]])


```
names != 'Bob'
```

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

▼ [Tip] NumPy Boolean Negative

'~' 연산자는 사용하지 않으며, ~ 혹은 Logical Not 함수를 대신 사용한다.

```
data[~(names == 'Will')] # ~ = !  
# ~(names == bob) : names가 밥이 아닌 것
```

```
array([[ -1.00766818, -2.26320999,  0.23523663, -0.39924381],  
       [ 0.7793744 ,  0.05219855,  0.58482829,  0.64349462],  
       [ 3.07552784, -0.54626755,  0.41046104, -0.39563456],  
       [-1.30555998,  0.3463475 , -1.78786867, -0.39672916],  
       [-1.0031312 , -1.73947791, -1.04028283, -0.27457652]])
```

```
data[(names)!='Will']
```

```
array([[ -1.00766818, -2.26320999,  0.23523663, -0.39924381],  
       [ 0.7793744 ,  0.05219855,  0.58482829,  0.64349462],  
       [ 3.07552784, -0.54626755,  0.41046104, -0.39563456],  
       [-1.30555998,  0.3463475 , -1.78786867, -0.39672916],  
       [-1.0031312 , -1.73947791, -1.04028283, -0.27457652]])
```

```
mask = (names == 'Bob')|(names=='Will')
```

```
mask
```

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

```
data[mask]
```

```
array([[ -1.00766818, -2.26320999,  0.23523663, -0.39924381],  
       [ 0.76974409, -2.27627079,  0.02299811, -0.69820179],  
       [ 3.07552784, -0.54626755,  0.41046104, -0.39563456],  
       [ 1.26317167, -0.22958675, -1.14526231, -1.95480864]])
```

```
data[data<0] = 0  
data
```

```
array([[0.          , 0.          , 0.23523663, 0.          ],  
       [0.7793744 , 0.05219855, 0.58482829, 0.64349462],  
       [0.76974409, 0.          , 0.02299811, 0.          ],  
       [3.07552784, 0.          , 0.41046104, 0.          ],  
       [1.26317167, 0.          , 0.          , 0.          ],  
       [0.          , 0.3463475 , 0.          , 0.          ],  
       [0.          , 0.          , 0.          , 0.          ]])
```

```
data[names!='Joe'] = 7
data
```

```
array([[7.          , 7.          , 7.          , 7.          ],
       [0.7793744 , 0.05219855, 0.58482829, 0.64349462],
       [7.          , 7.          , 7.          , 7.          ],
       [7.          , 7.          , 7.          , 7.          ],
       [7.          , 7.          , 7.          , 7.          ],
       [0.          , 0.3463475 , 0.          , 0.          ],
       [0.          , 0.          , 0.          , 0.          ]])
```

▼ 팬시 색인

- 팬시 색인은 정수 배열을 사용한 색인을 설명하기 위해 NumPy에서 차용한 단어다.

```
arr = np.empty((8,4))
for i in range(8):
    arr[i] = i
arr
```

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

```
arr[4,3]
```

```
4.0
```

- 특정한 순서로 로우를 선택하고 싶다면, 그냥 원하는 순서가 명시된 정수가 담긴 ndarray나 리스트를 넘기면 된다.

```
arr[[4,3,0,6]]
```

```
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [0., 0., 0., 0.],
       [6., 6., 6., 6.]])
```

```
arr[[-3,-5,-7]]
```

```
array([[5., 5., 5., 5.],
       [3., 3., 3., 3.],
       [1., 1., 1., 1.]])
```

- 다차원 색인 배열을 넘기는 것은 조금 다르게 동작
- 각각의 색인 튜플에 대응하는 1차원 배열을 선택한 후, reshape 시킨다.

```
arr = np.arange(32).reshape((8,4))
arr
```

```
array([[ 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, 30, 31]])
```

```
arr[(1,5,7,2),(0,3,1,2)]
```

```
array([ 4, 23, 29, 10])
```

```
arr[[1,5,7,2]]
```

```
array([[ 4,  5,  6,  7],
       [20, 21, 22, 23],
       [28, 29, 30, 31],
       [ 8,  9, 10, 11]])
```

```
arr[[1,5,7,2]][:,[0,3,1,2]] # 1,5,7,2 행을 가져와 모든 행에서 0,3,1,2열로 배열
```

```
array([[ 4,  7,  5,  6],
       [20, 23, 21, 22],
       [28, 31, 29, 30],
       [ 8, 11,  9, 10]])
```


```
arr[np.ix_([1,5,7,2],[0,3,1,2])]
```

```
array([[ 4,  7,  5,  6],
       [20, 23, 21, 22],
       [28, 31, 29, 30],
       [ 8, 11,  9, 10]])
```


▼ 배열 전치와 축 바꾸기

- 배열 전치는 데이터를 복사하지 않고 데이터 모양이 바뀐 뷰를 반환하는 특별한 기능
- ndarray는 transpose 메소드와 T라는 이름의 특수한 속성을 가진다.
- 다차원 배열의 경우 transpose 메소드는 튜플로 축 번호를 받아서 치환한다.


```
arr = np.arange(15).reshape((3,5))
arr
```

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


```
arr.T
```

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

```
arr = np.random.randn(6,3)
arr
```


```
 array([[ 0.96419619,  0.32103303, -0.54816212],
        [ 0.1662321 , -0.32955349,  0.80196088],
        [ 0.96895821,  0.19998857,  2.3244853 ],
        [ 1.04708548, -0.63431152,  0.28717086],
        [ 0.16206295, -0.4777443 , -0.79934935],
        [ 0.6388379 ,  0.33902876, -0.04641755]])
```

```
np.dot(arr.T, arr)
```

```
 array([[ 3.4269537, -0.07648159,  1.99859917],
        [-0.07648159,  0.99719436,  0.20859528],
        [ 1.99859917,  0.20859528,  7.07043593]])
```


- `np.dot` = `arr`을 행과 열을 바꾸고, 본래 `arr`과 곱하는 함수

```
arr = np.arange(16).reshape((2,2,4))
arr
```

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

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]])
```

```
arr.transpose((0,1,2))
```

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

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]])
```

```
arr.swapaxes(1,2)
```



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