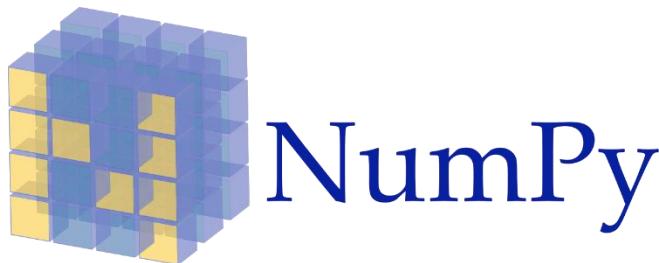


# Numpy 개요

# 이번 수업에서는

1. Numpy란
2. Numpy의 특징과 장점
3. 유용한 링크

- "Numerical Python" or "Numeric Python"
- The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.



출처 Cheat Sheet at DataCamp.com

## NumPy Arrays

### 1D array

1	2	3
---	---	---

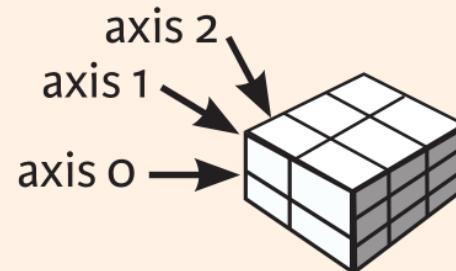
### 2D array

axis 1 →

axis 0 →

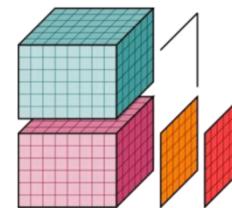
1.5	2	3
4	5	6

### 3D array





StatsModels  
Statistics in Python



xarray



scikits-image  
image processing in python



scikit-learn  
machine learning in Python



And many,  
many more...



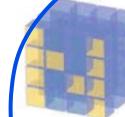
pandas  
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



matplotlib



bokeh



NumPy



IP[y]:  
IPython

python™



## Numpy 의 특징과 장점

- Python은 느리다. 왜? 그래서...
- 
- 강력한 N 차원 배열 객체. List 보다 빠르다. (메모리 구조)
  - 정교한 브로드캐스팅(Broadcast) 기능. 반복문을 쓰지 않음
  - 유용한 선형 대수학, 푸리에 변환 및 난수 기능
  - C/C ++ 및 포트란 코드 통합 도구
- 
- a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities

- 강력한 N 차원 배열 객체. List 보다 빠르다.
- 정교한 브로드캐스팅(Broadcast) 기능. 반복문을 쓰지 않음
- 유용한 선형 대수학, 푸리에 변환 및 난수 기능
- C/C ++ 및 포트란 코드 통합 도구

출처: Inside NumPy: Preparing for the Next Decade

What does NumPy need in order to thrive?

- Sustained funding at a higher level than today.  
Our estimate: **10 full-time people**

### Official Numpy website

- <https://www.numpy.org/> 또는  
<https://docs.scipy.org/doc/numpy/>

### Official Numpy tutorial

- <https://numpy.org/devdocs/user/index.html>
  - [https://numpy.org/devdocs/user/absolute\\_beginners.html](https://numpy.org/devdocs/user/absolute_beginners.html)
  - <https://numpy.org/devdocs/user/quickstart.html>
- <https://numpy.org/devdocs/reference/index.html>

### Source code

- <https://github.com/numpy/numpy>

- <https://www.python-course.eu/numpy.php>
- [http://taewan.kim/post/numpy\\_cheat\\_sheet/](http://taewan.kim/post/numpy_cheat_sheet/)
- **Youtube**
  - Introduction to Numerical Computing with NumPy | SciPy 2019
  - Python Basics for Data Science | edx
- **Youtube**
  - TeamLab X Inflearn
  - Park 널널한 교수

Numpy 는 무엇의 약자인가요?

Python에서 Numpy 가 왜 필요하죠?

Numpy 가 빠른 이유는 무엇인가요

- python 코드가 아니라, Numpy는 C코드로 작성되어 있습니다.
- 메모리 저장구조가 다릅니다
- 반복문을 사용하지 않고 배열을 통해 바로 계산됩니다.
- 벡터라이즈와 브로드캐스팅이 사용됩니다.

# 이번 수업에서는

1. import 의 역할
2. 배열(array)이란 . shape, 표기법. 용어
3. 배열(array) 생성하기. 배열객체의 속성
4. Python vs Numpy 데이터타입

### ➤ import

```
import numpy as np
```

- 모듈(라이브러리)을 호출하여 속성과 메서드를 사용한다
- numpy.sum( )을 간단히 별칭(alias)을 사용해 np.sum( )으로 사용

## 배열(Array)이란

- `ndarray ( n dimensional array )`

### NumPy Arrays

#### 1D array

1	2	3
---	---	---

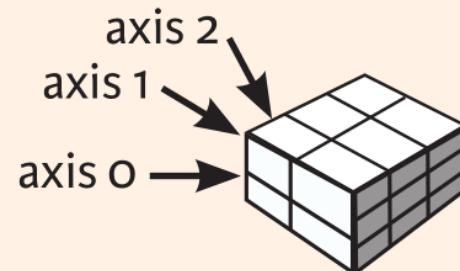
#### 2D array

axis 1 →

axis 0 →

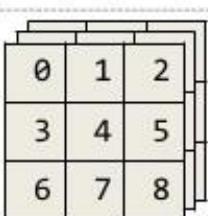
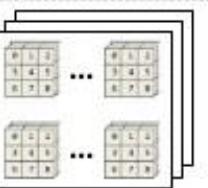
1.5	2	3
4	5	6

#### 3D array



- 각 차원을 축(axis)라고 합니다
- 축의 개수는 차원의 개수인데 rank 라고도 합니다. `ndim`
- 배열의 차원을 `shape`라고 하고 tuple로 표시합니다. `(3, )`, `(2, 3)`
- `shape` 안의 숫자는 각 차원에 있는 원소의 개수입니다.
- 전체 원소의 개수는 `size`라고 합니다.

## Array vs Matrix

Dimensions	Example	Terminology									
1	<table border="1"> <tr><td>0</td><td>1</td><td>2</td></tr> </table>	0	1	2	Vector						
0	1	2									
2	<table border="1"> <tr><td>0</td><td>1</td><td>2</td></tr> <tr><td>3</td><td>4</td><td>5</td></tr> <tr><td>6</td><td>7</td><td>8</td></tr> </table>	0	1	2	3	4	5	6	7	8	Matrix
0	1	2									
3	4	5									
6	7	8									
3		3D Array (3 <sup>rd</sup> order Tensor)									
N		ND Array									

## Array versus Matrix Operations

- Consider two  $2 \times 2$  images

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{ and } \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- Array Product is:

$$\begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

- Matrix Product is:

$$\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

[https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/linear\\_algebra.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/linear_algebra.html)

## Array Creation

```
a = np.array([0, 1, 2, 3])  
print(a)  
  
a  
  
type(a)
```

→ [0 1 2 3]

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

numpy.ndarray

- array 함수는 np 라이브러리에 들어 있다
- 입력값으로 list 가 들어간다
- 동일한 데이터형이어야 한다

## 리스트를 array로 바꾸기

```
a = [0, 1, 2, 3]
```

```
type(a)
```

⇨ <class 'list'>

- 리스트가 있다면 그 리스트를 입력값으로 넣어주기만 하면 된다

```
array_a = np.array(a)
```

```
type(array_a)
```

```
array_a
```

⇨ <class 'numpy.ndarray'>

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

## Arra y의 attribute(속성)

a.dtype

a.ndim

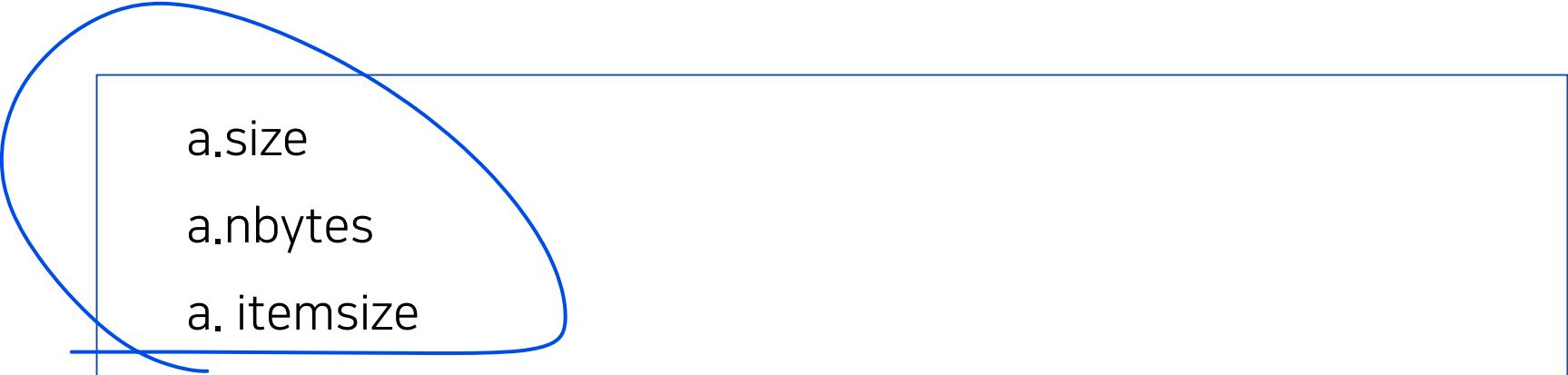
a.shape

→ dtype('int64')

1

(4, )

- 8 bit = 1 byte 입니다.
- 64 bit 이므로 8byte가 한 칸입니다.



```
a.size  
a.nbytes  
a.itemsize
```

→ 4

32

8

- size는 원소(element)의 개수입니다

## data type

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

```
a.dtype
```

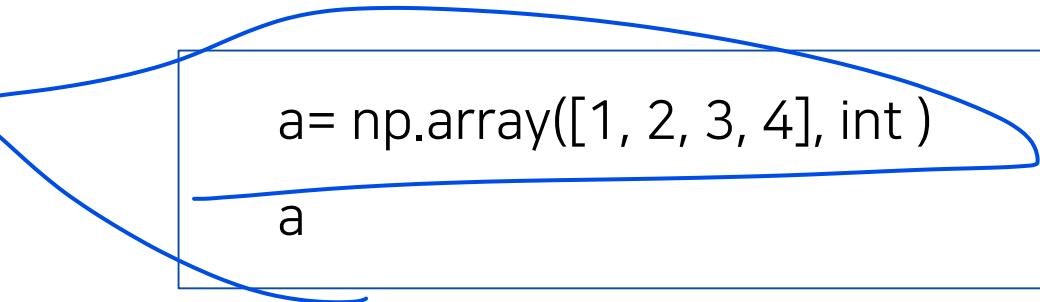
→ dtype('int64')

```
a= np.array([1, 2, 3.5, 4.1])
```

```
a.dtype
```

→ dtype('float64')

- 데이터형은 자동으로 할당된다



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

a

→ array([1, 2, 3, 4])

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

b

→ array([1., 2., 3., 4.])

```
a.dtype
```

→ dtype('int64')

```
b.dtype
```

→ dtype('float64')

```
np.array([1, 2, 3.5, 4.1], int )
```

→ array([1, 2, 3, 4])

## Python vs Numpy

```
np.array([1, 2, 3.5, 4.1], float64)
```

→ NameError: name 'float64' is not defined

- Python의 데이터형은

int, float, str, boolean, complex number, list, tuple, set, dict 이다

Data type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

## 마무리 간단퀴즈

설치된 Numpy 라이브러리를 사용하려면 가장 먼저 해야할 일은?

Aarray([0, 1, 2, 3]) 와 같이 출력되도록 1차원 배열을 만들어 봅시다.

Aarray 객체에서 속성의 종류는 무엇이 있었습니까? 3개 이상 말해 봅시다.

Python의 데이터타입이 아닌, Numpy 만의 데이터 타입을 5가지 이상 말해 봅시다.

# 이번 수업에서는

1. Creation 2D Array , 3D Array
2. Numpy Operation

## 2차원, 3차원 배열 만들기

- np.array( )함수를 써서 만들기
- arange( )와 reshape( )를 써서 만들기
- 3차원 배열 만들기

## 2차원 배열 만들기 1

```
arr1 = np.array([[1,2], [3,4]])
```

```
arr2 = np.array([[5,6], [7,8]])
```

```
arr1
```

→ array([[1, 2],  
 [3, 4]])

1	2
3	4

5	6
7	8

```
arr2
```

→ array([[5, 6],  
 [7, 8]])

- 중첩리스트

## 2차원 배열 만들기 2

- arange와 reshape로 만들기

```
a = np.arange(8)
```

```
a.shape
```

```
a.reshape(2, 4)
```

→ (8, )

```
array([[ 5, 12],  
       [21, 32]])
```

```
np.arange(8).reshape(2, 4)
```

```
np.arange(8).reshape(4,2)
```

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

```
np.arange(6, dtype= np.float32).reshape(2,3)
```

```
→ array([[0., 1., 2.],  
         [3., 4., 5.]], dtype=float32)
```

- arange( )를 쓰면 데이터타입도 정할 수 있습니다.

## 3차원 배열 만들기

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

→ array([[[1, 2],  
 [3, 4]],

[[5, 6],  
 [7, 8]]])

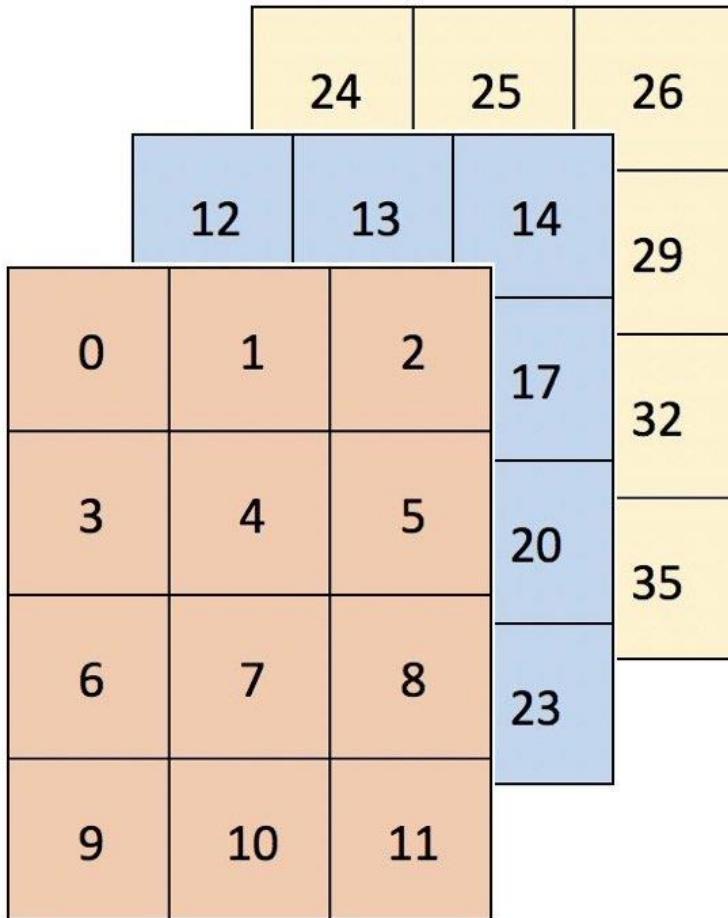
```
np.arange(8).reshape(2, 2, 2)
```

```
np.arange(24).reshape(2, 3, 4)
```

```
→ 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]]])
```

```
np.arange(24).reshape(2,3,4).ndim
```

```
arr = np.arange(36).reshape(3, 4, 3)
```



출처 <https://realpython.com/numpy-array-programming/>

## 배열의 연산( Operation)

- 리스트의 연산과 비슷하지만, 약간 다르다.
- numpy 함수를 쓰는 것보다 객체.함수( )가 편리하다
- Numpy의 연산은 원소들끼리 이루어진다. element wise
- array의 형태(shape)가 안 맞으면, 자동으로 맞추어 주기도 한다.  
broadcasting

## List Operation 연습

```
a =[1, 2, 3, 4]  
b = [5, 6, 7, 8])  
print(a, b)  
type(a)
```

→ [1, 2, 3, 4] [5, 6, 7, 8]

list

a + b

→ [1, 2, 3, 4, 5, 6, 7, 8]

a - b

→

```
a * 3
```

```
↪ [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

```
a[1]
```

```
↪ 2
```

```
a[1] = 10
```

```
a
```

```
↪ 2
```

## 1D Array Operation

### ➤ 1 D Array

- 사칙연산 + , - , \* , /
- 거듭제곱, 제곱근
- 지수, 로그
- 삼각함수

## 1D Array Operation

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

→ [1, 2, 3, 4] [5, 6, 7, 8]

numpy.ndarray

- a.shape
- a.size

➤ 1차원 array 의 사칙연산

a + b , np.add(a,b)

↳ array([ 6, 8, 10, 12])

7

1	2	3	4
5	6	7	8

a - b, np.subtract(a,b)

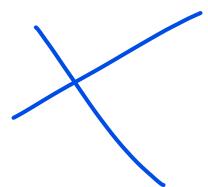
↳ array([-4, -4, -4, -4])

- element wise

a \* b ,

np.multiply(a,b)

[→ array([ 5, 12, 21, 32])



1	2	3	4
---	---	---	---

5	6	7	8
---	---	---	---

a / b ,

np.divide(a,b)

[→ array([0.2 , 0.33333333, 0.42857143, 0.5 ])

a \* b ,

np.multiply(a,b)

[→ array([ 5, 12, 21, 32])

1	2	3	4
---	---	---	---

5	6	7	8
---	---	---	---

a \*\* b

[→ array([ 1, 64, 2187, 65536])

```
a * 3
```

```
[→ array([ 3, 6, 9, 12])
```

1	2	3	4
---	---	---	---

```
a + 1
```

```
[→ array([2, 3, 4, 5])
```

- broadcasting

## 2D Array Operation

```
arr1 = np.array([[1,2], [3,4]])
```

```
arr2 = np.array([[5,6], [7,8]])
```

```
arr1
```

1	2
3	4

5	6
7	8

➤ 2차원 array의 곱셈

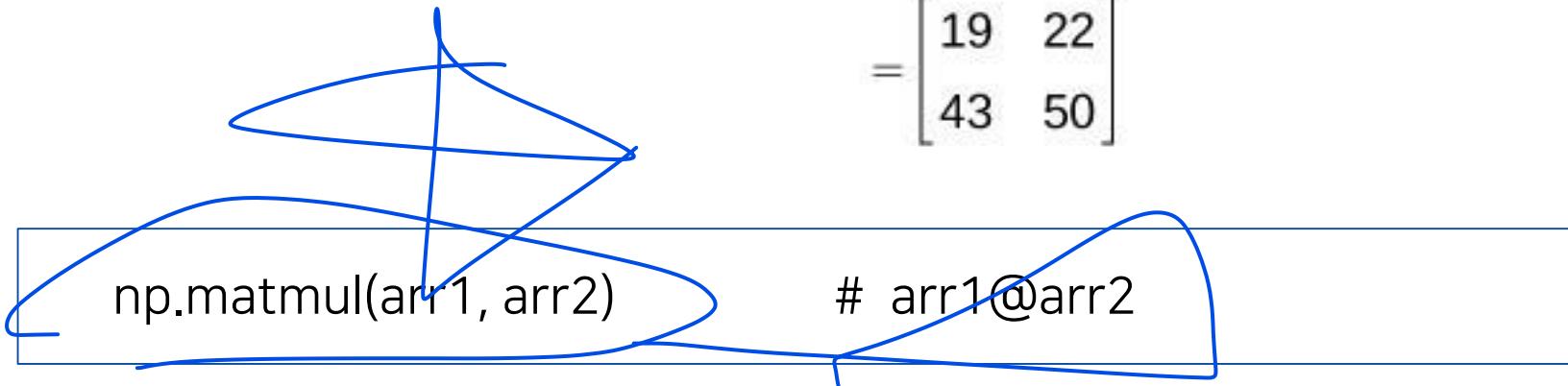
```
np.multiply(arr1, arr2) # arr1*arr2
```

→ array([[ 5, 12],  
[21, 32]])

## 2D Array Operation

### ➤ matrix 의 곱셈

$$\begin{aligned}AB &= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \\&= \begin{bmatrix} 1(5) + 2(7) & 1(6) + 2(8) \\ 3(5) + 4(7) & 3(6) + 4(8) \end{bmatrix} \\&= \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}\end{aligned}$$



```
[→ array([[19, 22],  
          [43, 50]])
```

- 2x2 여서 np.dot(arr1, arr2) 도 위와 같다.

## 마무리 간단퀴즈

다음과 같이 출력되도록 2차원 배열을 2가지 방법으로 만들어 봅시다.

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

그림과 같이 0부터 35까지로 이루어진 3차원 배열을 만들어 봅시다

배열의 곱셈과 행렬의 곱셈은 어떻게 다른가요?

0	1	2	17	29	32	24	25	26
3	4	5	20		35			
6	7	8	23					
9	10	11						

# 이번 수업에서는

1. Numpy 내장함수를 이용하여 배열 만들기
2. \*\*\*\_like ( ) 함수로 배열 만들기
3. 복사본을 만드는 두가지 방법

# Numpy 내장함수를 이용하여 array 만들기

- arange
- linspace
- zeros
- ones
- empty
- full
- tile
- \*\*\_like

## Ones and zeros

`empty(shape[, dtype, order])`  
`empty_like(prototype[, dtype, order, subok, ...])`  
`eye(N[, M, k, dtype, order])`  
`identity(n[, dtype])`  
`ones(shape[, dtype, order])`  
`ones_like(a[, dtype, order, subok, shape])`  
`zeros(shape[, dtype, order])`  
`zeros_like(a[, dtype, order, subok, shape])`  
`full(shape, fill_value[, dtype, order])`  
`full_like(a, fill_value[, dtype, order, ...])`

## Numerical ranges

`arange([start,] stop[, step,][, dtype])`  
`linspace(start, stop[, num, endpoint, ...])`  
`logspace(start, stop[, num, endpoint, base, ...])`  
`geomspace(start, stop[, num, endpoint, ...])`

## arange

➤ `arange( start, stop, step, dtype)`

```
b = np.arange(1, 9, 2)
```

```
b
```

↪ `array([1, 3, 5, 7])`

```
a = np.arange(9, dtype = np.float32)
```

```
a
```

## linspace

➤ `linspace( start, stop, numbers of element )`

```
c = np.linspace(0, 1, 6)
```

```
c
```

↳ `array([0., 0.2, 0.4, 0.6, 0.8, 1.])`

```
d = np.linspace(0, 1, 5, endpoint=False)
```

```
d
```

↳ `array([0., 0.2, 0.4, 0.6, 0.8])`

np.ones

→ ones( shape [, dtype, order] )

a = np.ones((3, 3), int)

a

# shape (3, 3) is a tuple

→ array([[1, 1, 1],  
[1, 1, 1],  
[1, 1, 1]])

a\*3

## np.zeros

➤ zeros( shape [, dtype, order] )

```
b = np.zeros((3, 3))
```

```
b
```

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

```
b[2,0] = 3    # sparse matrix
```

```
b
```

## np.diag

- Extract a diagonal or construct a diagonal array.

➤ diagonal은 대각선

```
a = diag([1,2,3])
```

```
a
```

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

```
c = np.arange(9).reshape(3,3)  
np.diag(c)
```

```
[[0 1 2]  
 [3 4 5]  
 [6 7 8]]
```

```
↳ array([0, 4, 8])
```

## np.identity

- Return the identity array.

➤ np.identity( )는 정사각행렬이 정해져 있다

```
d = np.identity(4)
```

```
d
```

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

```
np.identity(3)
```

## np.eye

- Return a 2-D array with ones on the diagonal and zeros elsewhere.

➤ np.eye( 4 )와 np.identity(4)는 같다

```
d = np.eye(4, 3)
```

```
d
```

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

```
np.eye(4, 3)
```

## np.tile

- Reapeating A the number of times given

```
a = np.array([0, 1, 2])
np.tile(a, 2)      # repeat a 2 times
```

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

```
np.tile(a, (3, 2))  # 3행 = 뒤에 것의 3뭉치
```

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

## np.tile

```
a = np.array([0, 1, 2])
np.tile(a, 2)          # repeat a 2 times
```

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

```
np.tile(a, (3, 2))  # 3행 = 뒤에 것의 3뭉치
```

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

## np.full

```
np.full((2,2), 3)
```

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

```
np.full((4, 3), 2)      # np.full( shape, fill_value)
```

## np.empty

```
np.empty((2,3)) # null에 해당하는 주소값, 초기화 안함
```

```
array([[3.1673758e-316, 0.0000000e+000, 0.0000000e+000],  
       [0.0000000e+000, 0.0000000e+000, 0.0000000e+000]])
```

## \*\*\*\_like( ) 함수로 배열만들기

- zeros\_like( )      **zeros\_like**  
                        Return an array of zeros with shape and type of input.
- ones\_like( )        **ones\_like**  
                        Return an array of ones with shape and type of input.
- empty\_like()       **empty\_like**  
                        Return an empty array with shape and type of input.
- full\_like()        **full\_like**  
                        Fill an array with shape and type of input.

## zeros\_like

- 형태와 데이터타입을 유지하면서 0으로 채움

### `zeros_like`

Return an array of zeros with shape and type of input.

```
a = np.arange(6).reshape(2,3)
a
```

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

```
b = np.arange(3, dtype=np.float)
print(b)
b.dtype
```

```
[0. 1. 2.]
dtype('float64')
```

```
np.zeros_like(a)
```

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

```
np.zeros_like(b)
```

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

## ones\_like, empty\_like

- 형태와 데이터타입을 유지하면서 .....

### ones\_like

Return an array of ones with shape and type of input.

### empty\_like

Return an empty array with shape and type of input.

```
np.ones_like(a)
```

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

```
np.empty_like(a)
```

```
array([[71919968, 0, 0],  
       [0, 0, 0]])
```

```
np.ones_like(b)
```

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

```
np.empty_like(b)
```

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

## full\_like

- 형태와 데이터타입을 유지하면서 .....

### full\_like

Fill an array with shape and type of input.

```
np.full_like(a)      # 당연히..
```

```
-----  
TypeError                                Trace  
<ipython-input-98-87a6b2956ef1> in <module>()  
----> 1 np.full_like(a)      # 당연히..
```

```
np.full_like(a, 3)
```

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

```
np.full_like(b, 3.1)
```

```
array([3.1, 3.1, 3.1])
```

## copy( )의 두가지 방법

- 기본 배열로부터 새로운 배열을 생성하기 위해서는 copy 함수로 명시적으로 사용해야 함
- copy 함수로 복사한 배열은 원본 배열과 완전히 다른 새로운 객체입니다.
- slice, indexing은 새로운 객체가 아닌 기존 배열의 뷰(View)입니다.
- 반환한 배열의 값을 변경하면 원본 배열이 변경된다.

## copy 의 두가지 방법

- 방법 1

```
B = A.copy()      # B = np.copy(A)
```

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

B[3] = 0

B

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

A

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

```
A = np.arange(1, 6)
```

A

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

0 1 2 3 4  
X

## copy 의 두가지 방법

- 방법 2

```
C = A      # 이렇게 할당  
C
```

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

```
C[3]=0  
C
```

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

```
A
```

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

```
A = np.arange(1,6)  
A
```

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

- View

## 마무리 간단퀴즈

- 다음 코드에 대한 출력을 예상해 봅시다

`np.arange(1, 9, 2)`

`np.linspace(0, 1, 6)`

`np.ones((3, 3), int)`

\*\*\* `_like`에는 어떤 것들이 있습니다?

~~copy와 비교하여 view란 무엇입니까?~~

# 이번 수업에서는

1. 배열의 결합. hstack, vstack, dstack
2. 배열의 분리 hsplit, vsplit
3. 배열의 형태변환 reshape, flatten, ravel, transpose

배열의 결합. stack 으로 만들기

- hstack
- vstack
- dstack
- concatenate

stack 이란 : FIFO . 선착순의 반대

## vstack

```
a = np.arange(5)  
a*10
```

```
array([ 0, 10, 20, 30, 40])
```

```
np.vstack([a * 10, a * 20])
```

```
array([[ 0, 10, 20, 30, 40],  
       [ 0, 20, 40, 60, 80]])
```

```
np.vstack([a * 10, a * 20]).shape
```

```
(2, 5)
```

## hstack

```
np.hstack([a * 10, a * 20])
```

```
array([ 0, 10, 20, 30, 40, 0, 20, 40, 60, 80])
```

```
a = [1,2,3]
b = [4,5,6]
np.vstack([a,b]) # 리스트이거나
```

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

```
a = np.array([1,2,3])
b = np.array([4,5,6])
np.vstack([a,b]) # array 이거나
```

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

## dstack

# stack arrays in sequence depth wise

np.dstack((a,b))

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

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

[[1]  
[2]  
[3]]  
[[2]  
[3]  
[4]]

array([[ [1, 2] ],

      [ [2, 3] ],

      [ [3, 4] ]])

## 배열의 분리 splitting

- 수평분리 hsplit 와 수직분리 vsplit

`np.hsplit(a, 2)`

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	23	24	25

shape: (4, 6)

`a.shape[1] → 6`

2개 그룹으로 분할  
`a[:, :3], a[:, 3:]`

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	23	24	25

shape: (2, 4, 3)

<http://taewan.kim>

`np.vsplit(a, 2)`

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

shape: (4, 6)

`a.shape[0] → 4`

2개 그룹으로 분할  
`a[:2], a[2:]`

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

13	14	15	16	17	18
19	20	21	22	23	24

shape: (2, 2, 6)

[그림출처 http://taewan.kim/post/](http://taewan.kim/post/)

## hsplit

```
a = np.arange(1, 25).reshape((4, 6))
a
```

```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12],
       [13, 14, 15, 16, 17, 18],
       [19, 20, 21, 22, 23, 24]])
```

```
np.hsplit(a, 2)
```

```
[array([[ 1,  2,  3],
       [ 7,  8,  9],
       [13, 14, 15],
       [19, 20, 21]]), array([[ 4,  5,  6],
       [10, 11, 12],
       [16, 17, 18],
       [22, 23, 24]])]
```

```
np.hsplit(a, 3)
```

```
[array([[ 1,  2],
       [ 7,  8],
       [13, 14],
       [19, 20]]), array([[ 3,  4],
       [ 9, 10],
       [15, 16],
       [21, 22]]), array([[ 5,  6],
       [11, 12],
       [17, 18],
       [23, 24]])]
```

## vsplit

```
np.vsplit(a, 2)
```

```
[array([[ 1,  2,  3,  4,  5,  6],  
       [ 7,  8,  9, 10, 11, 12]]), array([[13, 14, 15, 16, 17, 18],  
       [19, 20, 21, 22, 23, 24]])]
```

```
np.vsplit(a, 3)
```

---

TypeError

Traceback (most recent call

[/usr/local/lib/python3.6/dist-packages/numpy/lib/shape\\_base.py](#) in sp

```
np.vsplit(a, 4)
```

```
[array([[1, 2, 3, 4, 5, 6]]),  
 array([[ 7,  8,  9, 10, 11, 12]]),  
 array([[13, 14, 15, 16, 17, 18]]),  
 array([[19, 20, 21, 22, 23, 24]])]
```

## Numpy의 shape 변경 함수

- reshape
- flatten
- ravel

## reshape

```
a = np.arange(20)  
a
```

```
b = np.reshape(a, (4, 5))  
b
```

```
a.reshape(4,5) # better expression
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14],  
       [15, 16, 17, 18, 19]])
```

## reshape

```
a = np.arange(24)  
a.reshape(2, 12)
```

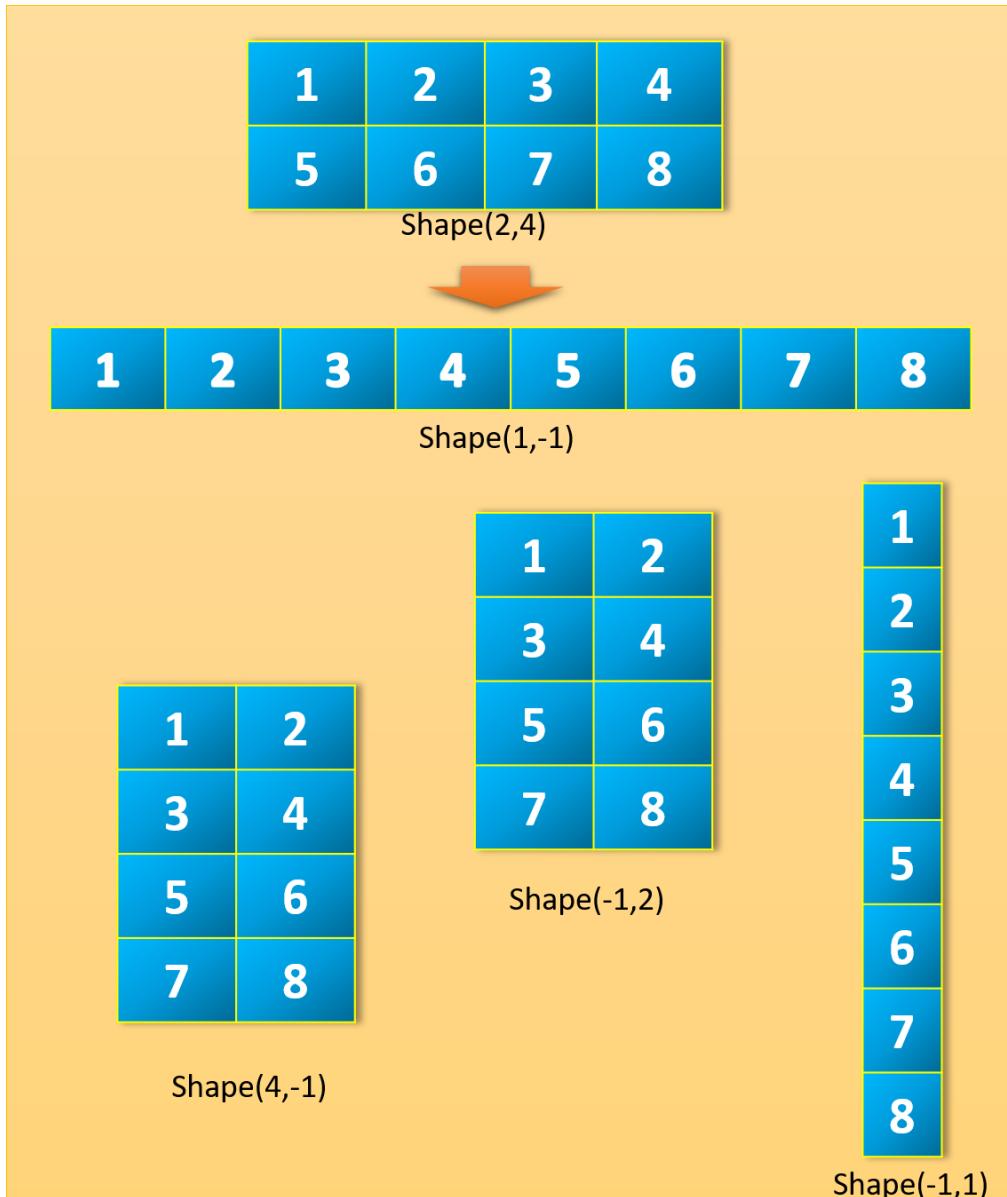
```
a.reshape(2, -1) # -1은 자동으로 맞추어 주라는 뜻
```

```
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]])
```

```
a.reshape(-1, 6)
```

```
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]])
```

## reshape



<https://towardsdatascience.com/5-smart-python-numpy-functions-dfd1072d2cb4>

## flatten

```
c = b.flatten() # return a copy
```

```
c
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,  
       17, 18, 19])
```

```
d=b.ravel() # flatten 과 차이점  
d
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,  
       17, 18, 19])
```

```
c[3] =30  
print(c)
```

```
[ 0  1  2 30  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

```
print(b)
```

```
[[ 0  1  2  3  4]  
 [ 5  6  7  8  9]  
 [10 11 12 13 14]  
 [15 16 17 18 19]]
```

```
d[3] =30  
print(d)
```

```
[ 0  1  2 30  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

```
print(b)
```

```
[[ 0  1  2 30  4]  
 [ 5  6  7  8  9]  
 [10 11 12 13 14]  
 [15 16 17 18 19]]
```

## transpose

$$[1 \ 2]^T = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

- 행렬의 곱셈을 가능하게 해 줍니다.  $1 \times 2$  행렬은 내적 계산

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

`a.T`  
`np.transpose(a)`  
`np.swapaxes(a, 0, 1)`

0	5	10
1	6	11
2	7	12
3	8	13
4	9	14

```
a = np.array([[0, 1, 2, 3]])
```

```
a
```

→ array([[0, 1, 2, 3]])

```
b = a.T
```

```
b
```

# T attribute 사용

→ array([[],  
 [1],  
 [2],  
 [3]])

```
a = np.arange(6).reshape(2, 3)  
a
```

→ array([[0, 1, 2], [3, 4, 5]])

```
np.transpose(a) # transpose() 함수 사용
```

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

## 마무리 간단퀴즈

a = np.array([1,2,3]) 이고 b = np.array([4,5,6]) 일 때,

array([[1, 2, 3],

[4, 5, 6]]) 이 되도록 결합하세요.

np.flatten( ) 과 np.ravel( ) 의 차이점을 말해 보세요

[[1, 2],

[3, 4,]] 인 배열을 만들고 전치하여 [[1, 3], [2, 4]]로 만드세요.

ut

# 이번 수업에서는

1. 비교 Comparison

2. 정렬 Sort

3. 인덱싱 indexing

## 두 배열을 비교하기

- 같다( == )
- 크다, 작다
- 배열 전체를 하나로 비교 (`np.array_equal`)

## Comparison

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

```
print(a, b)  
type(a)
```

```
[1 2 3 4] [3 4 5 6]  
numpy.ndarray
```

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

a == b

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

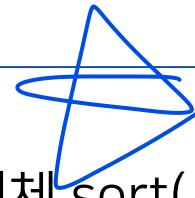
a > b

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

```
np.array_equal(a, b)
```

False

## 정렬하기 sort( )



- numpy.sort()와 객체.sort()는 다르다.
- 1차원 배열의 정렬, 역순으로 정렬
- 2차원 배열의 정렬과 축기준으로 정렬하기

## np.sort( )

- np.sort( ) 함수와 객체.sort()의 차이

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

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

```
np.sort(a) # View
```

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

```
a
```

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

```
a.sort() # 배열 자체를 정렬
```

```
a
```

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

## np.sort( )

- 역순으로 정렬하려면,

```
a = np.array([4, 2, 6, 5, 1, 3, 0])
np.sort(a)[::-1]    ## 정렬한 후 step이 뒤에서 읽어옴
```

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

## np.sort( )

- 축을 지정하여 정렬

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

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

```
np.sort(x, axis=0) # row - wise
```

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

```
np.sort(x, axis = 1) # coloum - wise
```

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

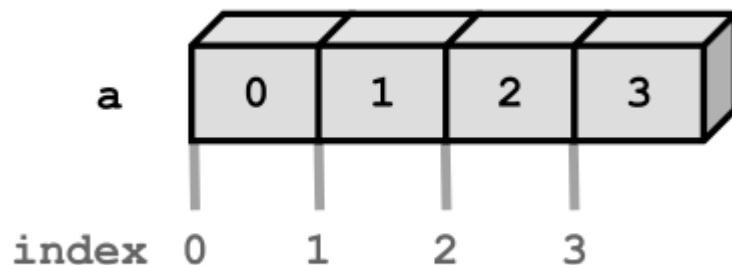
## 인덱싱( Indexing)

- 인덱싱이란

## 1D Indexing

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

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

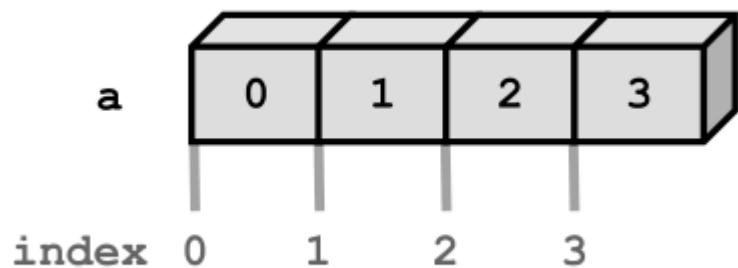


```
a = np.array([0, 1, 2, 3])
a.ndim
a.size
a.nbytes
a.dtype
```

```
dtype('int64')
```

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

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

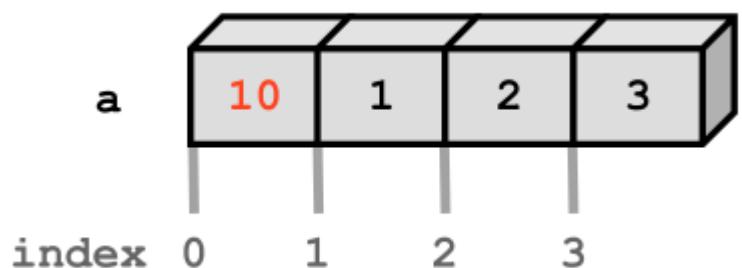


```
a = np.array([0, 1, 2, 3])  
a.ndim  
a.size  
a.nbytes  
a.dtype
```

```
dtype('int64')
```

```
a[0] = 10  
a
```

```
array([10, 1, 2, 3])
```



```
a[0] = 10.5  
a
```

int

## 2D Indexing

```
a = np.array([[0, 1, 2, 3],  
             [10,11,12,13]])
```

```
a
```

```
array([[ 0,  1,  2,  3],  
       [10, 11, 12, 13]])
```

## 2D Array

a	0	1	2	3
	10	11	12	13

## 2D Indexing

a.ndim

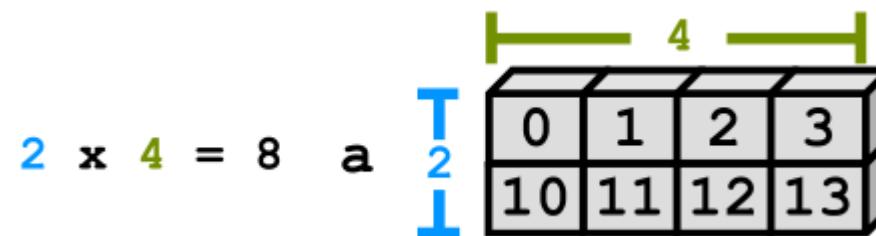
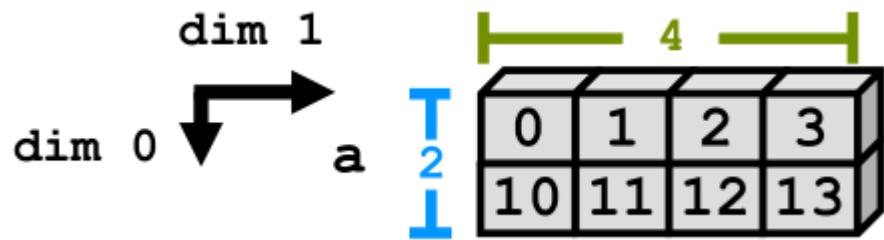
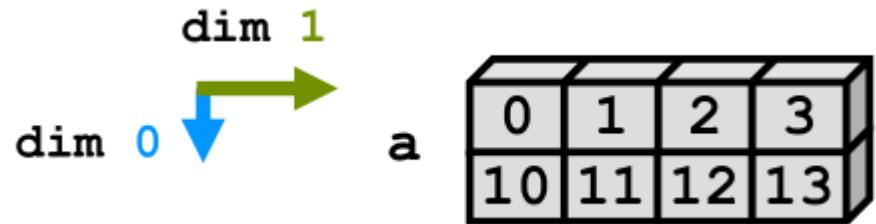
2

a.shape

(2, 4)

a.size

8



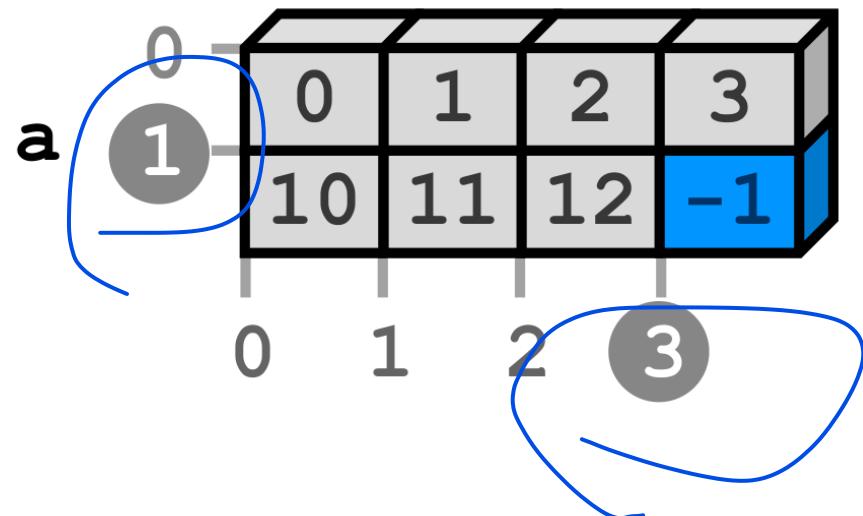
## Element Indexing

**GET**  $a[1, 3]$



**SET**  $a[1, 3] = -1$

```
array([[ 0,  1,  2,  3],  
       [10, 11, 12, -1]])
```



## Row/ Column Indexing

a[1]

2  
3  
4  
5

array([10, 11, 12, -1])

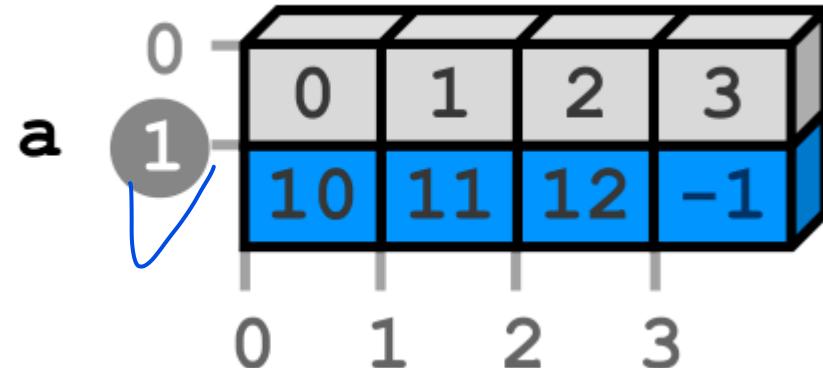
a[1, ]

array([10, 11, 12, -1])

a[  
, 1]

a[:, 1]

array([ 1, 11])



SyntaxError: invalid syntax

a[  
:, 2]

➤ 간단 퀴즈

- 다음과 같이 6x6 array 를 만들고, 빨간색 열을 뽑아보세요.

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

# 이번 수업에서는

1. Slicing
2. Fancy Indexing

슬라이싱 (Slicing)

## **var[lower:upper:step]**

Extracts a portion of a sequence by specifying a lower and upper bound.

The lower-bound element is included, but the upper-bound element is **not** included.

Mathematically: [lower, upper). The step value specifies the stride between elements.

## 1D slicing

- 1차원 배열은 리스트에서의 slicing 과 같다.

```
a = np.array([10,11,12,13,14])  
a
```

```
a[1:3]
```

```
array([11, 12])
```

```
a[1:-2]
```

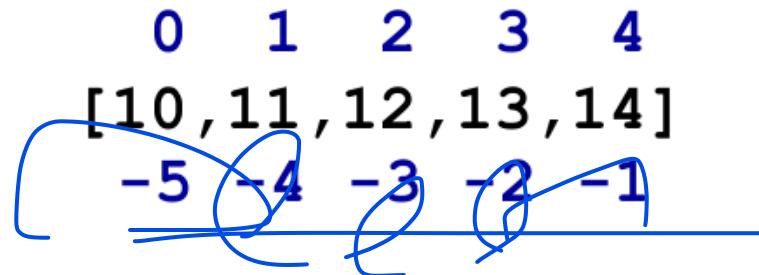
```
array([11, 12])
```

```
a[-4:3]
```

```
array([11, 12])
```

```
a[-2:1]
```

```
array([], dtype=int64)
```



```
a = np.array([10,11,12,13,14])  
a
```

0 1 2 3 4  
[10,11,12,13,14]  
-5 -4 -3 -2 -1

```
a[:3]
```

```
array([10, 11, 12])
```

```
a[-2:]
```

```
array([13, 14])
```

```
a[::2]
```

```
array([10, 12, 14])
```



## 2D slicing

- 2차원 배열은 1차원 배열이 2개 있는 것이다.(and 연산)

a[4:, 4:]

array([[44, 45],  
 [54, 55]])

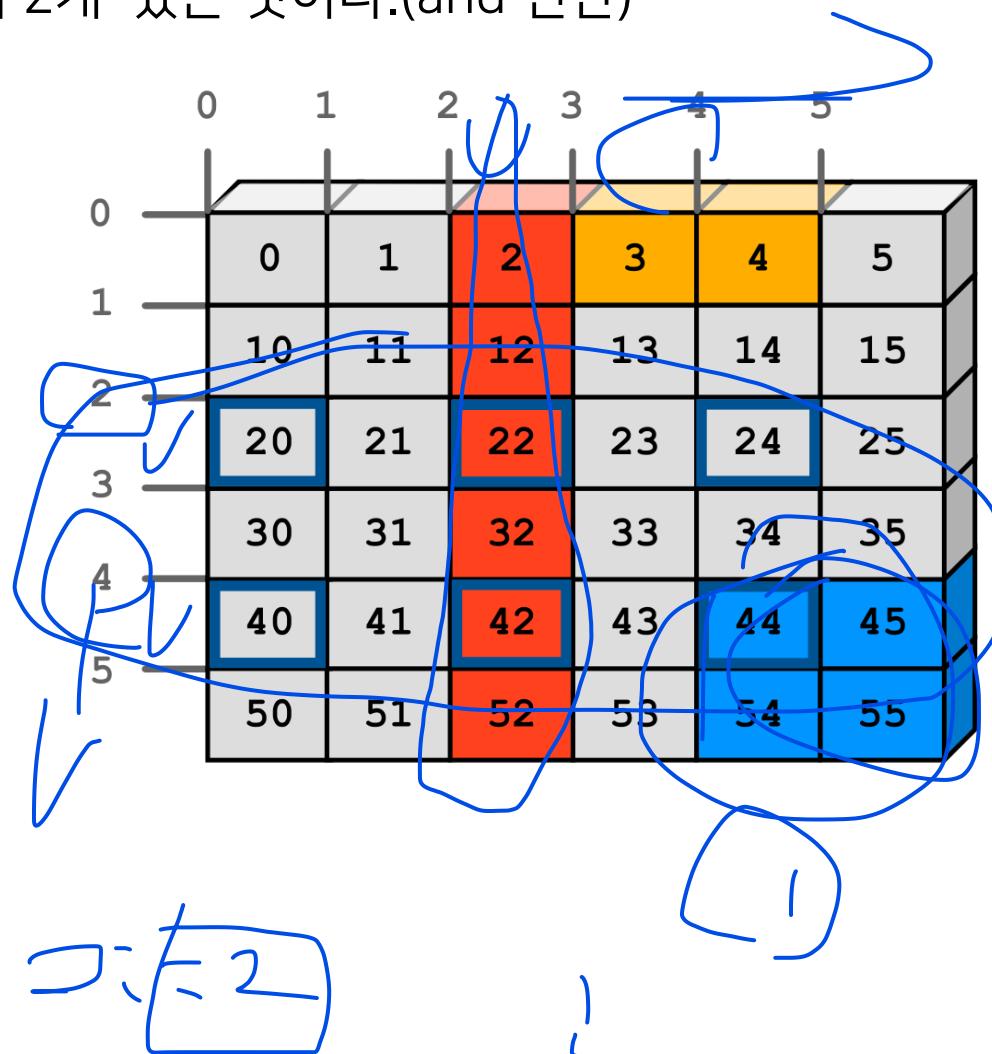
a[:, 2]

array([ 2, 12, 22, 32, 42, 52])

a[2::2, ::2]

array([[20, 22, 24],  
 [40, 42, 44]])

var[lower:upper:step]



## slicing으로 값 수정하기

```
a = np.array([0,1,2,3,4])  
a
```

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

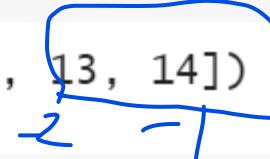
```
a[-2:]
```

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

```
a[-2:] = [13,14]
```

```
a
```

```
array([ 0,  1,  2, 13, 14])
```



```
a[-2:] = 11
```

```
a
```

```
array([ 0,  1,  2, 11, 11])
```

slicing은 원본 View

a = np.array([0, 1, 2, 3, 4])

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

b = a[2:4]

array([2, 3])

b[0] = 10

b

array([10, 3])

a

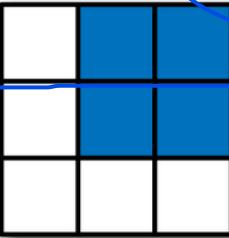
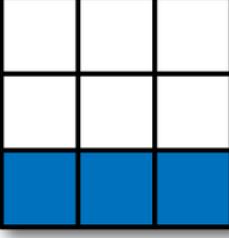
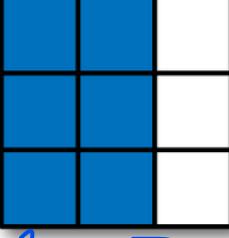
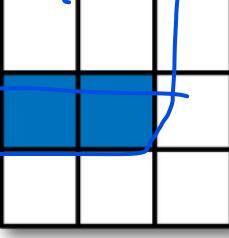
array([0, 1, 10, 3, 4])

- slice 한 것은 복사품이 아니라, 원본의 참조다

✓ 2가지

- changing b change a

## 그림으로 정리

	Expression	Shape
	$\text{arr}[:, :2]$	(3, 2)
	$\text{arr}[2]$ $\text{arr}[2, :]$ $\text{arr}[2:, :]$	(3,) (3,) (1, 3)
	$\text{arr}[:, :2]$	(3, 2)
	$\text{arr}[1, :2]$ $\text{arr}[1:2, :2]$	(2,) (1, 2)

## 마무리 퀴즈

- 다음과 같이 6x6 array 를 만들고, 노란색, 빨간색, 파란색으로 뽑아보세요.

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

팬시 인덱싱( Fancy Indexing) 이란

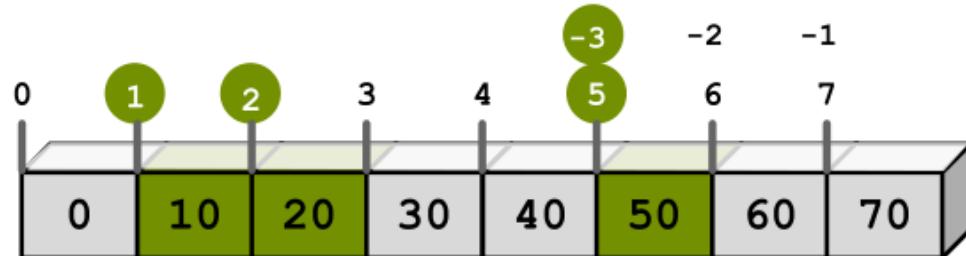
- 정수 배열을 indexer로 사용해서 indexing하는 방법

- indexing by position
- indexing by boolean
- indexing by condition

## Fancy Indexing.position

```
a = np.arange(0,80, step= 10)  
a
```

```
array([ 0, 10, 20, 30, 40, 50, 60, 70])
```



```
a[[1,2,-3]]
```

```
array([99, 99, 99])
```

```
indices = [1,2,-3]  
a[indices]
```

```
array([99, 99, 99])
```

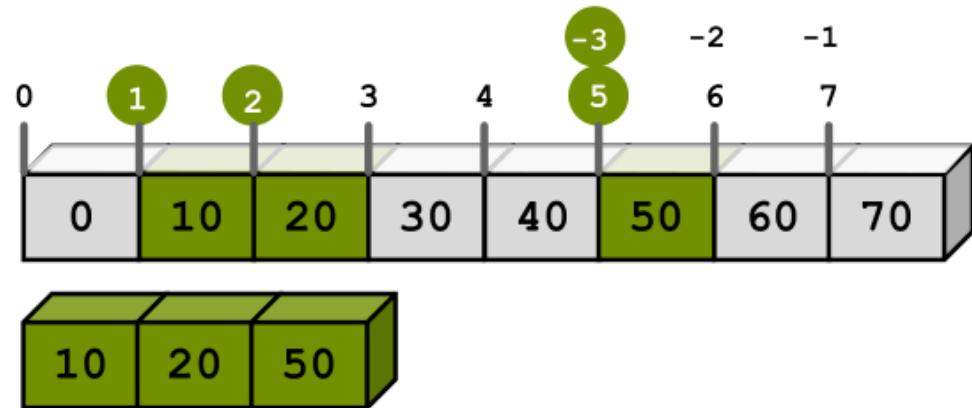
```
a[indices] = 99 # index에 있는 값 바꾸기  
a
```

```
array([ 0, 99, 99, 30, 40, 99, 60, 70])
```

## Fancy Indexing.boolean

```
a = np.arange(0,80, step= 10)  
a
```

```
array([ 0, 10, 20, 30, 40, 50, 60, 70])
```



```
mask = np.array([0, 1, 1, 0, 0, 1, 0, 0], dtype=bool)  
a[mask]
```

```
array([10, 20, 50])
```

```
mask = np.array([False, True, True, False, False, True, False, False])
```

## Fancy Indexing.boolean

```
a = np.arange(0,80, step= 10)  
a
```

```
array([ 0, 10, 20, 30, 40, 50, 60, 70])
```

- indexing by boolean

```
a>40
```

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

```
a[a>40]
```

```
array([50, 60, 70])
```

```
mask = a > 40  
a[mask]
```

```
array([50, 60, 70])
```

## 2차원 팬시 인덱싱

# 미리 알아두어야 할 것

```
a = np.arange(12).reshape(3,4)  
a
```

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

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

[ a[0,1], a[1,2], a[2,0], a[2,3] ]

[1, 6, 8, 11]

a[[0,1,2,2], [1,2,0,3]]

```
array([ 1,  6,  8, 11])
```

4 2 11

4 2 11  
5 0 1 2 1 1

## 2차원 팬시 인덱싱

```
a[[0, 1, 2, 3, 4],  
 [1, 2, 3, 4, 5]]
```

```
array([ 1, 12, 23, 34, 45])
```

```
a[3:, [0, 2, 5]]
```

```
array([[30, 32, 35],  
       [40, 42, 45],  
       [50, 52, 55]])
```

```
mask = np.array([1, 0, 1, 0, 0, 1], dtype=bool)  
a[3:, mask]
```

```
array([[30, 32, 35],  
       [40, 42, 45],  
       [50, 52, 55]])
```

```
a[mask, 2]
```

```
array([ 2, 22, 52])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

## 마무리 퀴즈

- 5 x 5 의 배열을 만들고, 짝수 부분의 원소를 뽑아보세요

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

# 이번 수업에서는

1. Broadcasting

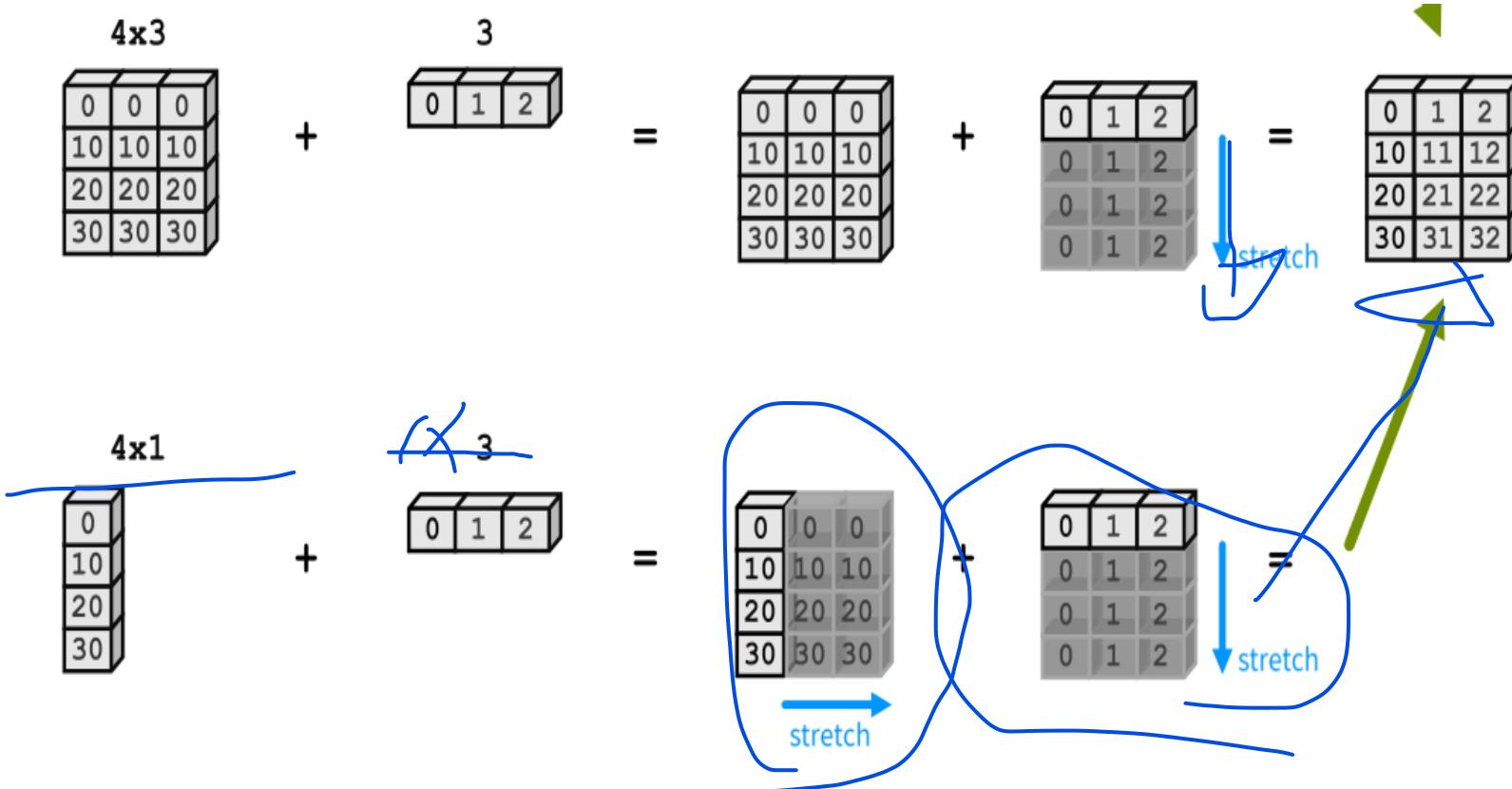
2.

## 브로드 캐스팅

브로드캐스팅( Broadcasting) 이란

- 차원(dimension)이 다른 두 배열의 연산에서
- 낮은 차원의 배열이 차원을 맞추어 주도록 변화한다
- 데이터의 복사를 하지 않으므로 빠르다

## 브로드 캐스팅



## Broadcasting

```
a = np.ones((3, 5))  
a
```

*3    5*

✓

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

```
b = np.ones((5,))  
b
```

*U1  
  2*

✓

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

~~a + b~~

1 | 1 | 1

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

~~a + 2~~

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

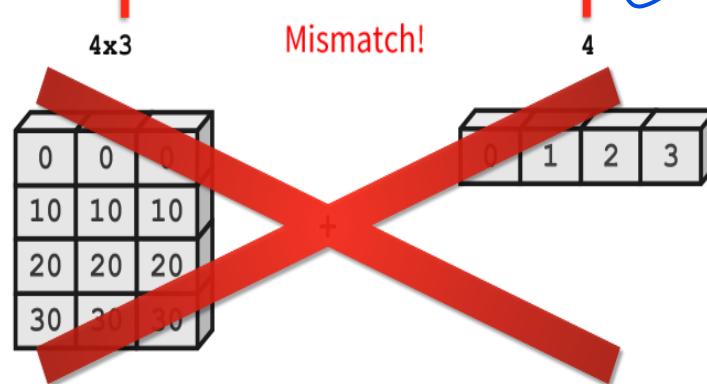
## Broadcasting

```
a = np.ones((3, 5))
```

```
a
```

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

적어도 어느 한쪽은 형태가 같아야 한다.



```
b= np.array([1,2])  
b
```

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

```
a + b
```

could not be broadcast together with shapes (3,5) (2,)

## Broadcasting

a

4x1

0	0	0
10	10	10
20	20	20
30	30	30

stretch

2-D Array

b

3

0	1	2
0	1	2
0	1	2
0	1	2

+

stretch

=

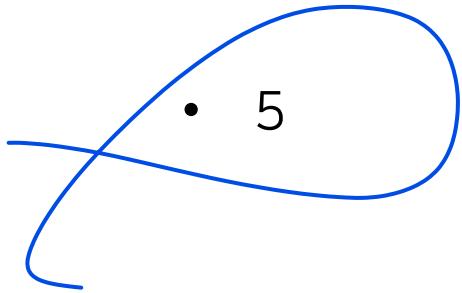
0	1	2
10	11	12
20	21	22
30	31	32

```
a = np.array([[0], [10], [20], [30]])  
b = np.array([0, 1, 2])
```

```
a + b
```

```
array([[ 0,  1,  2],  
       [10, 11, 12],  
       [20, 21, 22],  
       [30, 31, 32]])
```

## 마무리 퀴즈



# 이번 수업에서는

1. Universal functions ( ufuncs )
2. 파이썬은 왜 느린가? 넘파이는 왜 빠른가?

## Array Calculation (= Computation) by Universal Functions

**수학함수** • sum, prod, min, max, argmin, argmax

**통계함수** • mean, std, var

**진리값** • any, all

<https://docs.scipy.org/doc/numpy/reference/ufuncs.html>

## sum( )

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

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

```
np.sum(a)
```

```
21
```

```
a.shape
```

```
(2, 3)
```

```
np.sum(a, axis = 0)
```

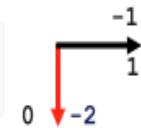
```
array([5, 7, 9])
```

```
np.sum(a, axis =1)
```

```
array([ 6, 15])
```

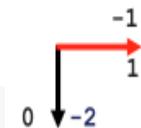
1	2	3
4	5	6

sum(a)



1	2	3
4	5	6

sum(a, axis=0)



5	7	9
---	---	---

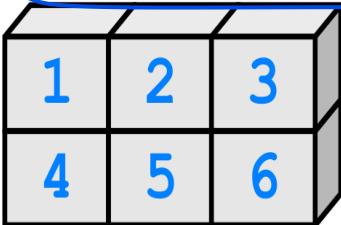
sum(a, axis=-1)



1	2	3
4	5	6

sum(a, axis=-1)

min( ), max( )



np.min(a)

1

np.max(a)

6

np.min(a , axis = 0)

array([1, 2, 3])

np.max(a, axis= 0)

array([4, 5, 6])

np.min(a, axis = 1)

array([1, 4])

np.max(a, axis= 1)

array([3, 6])

## argmin( ), argmax( )

1	2	3
4	5	6

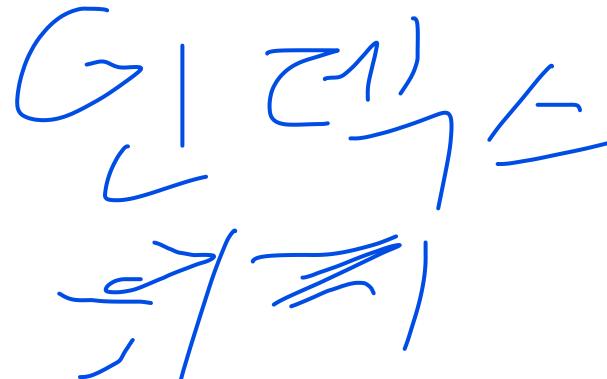
arg\_ methods return the location in 1D on a raveled index of the original array, instead of the value

`np.argmin(a)`

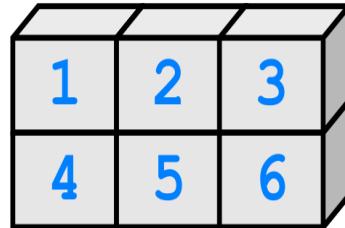
0

`np.argmax(a)`

5



## mean( ), std( ), var( )



```
np.mean(a)
```

```
# a.mean()
```

3.5

```
np.mean(a, axis = 0)
```

```
# a.mean(axis = 0)
```

array([2.5, 3.5, 4.5])

```
np.mean(a, axis = 1)
```

array([2., 5.])

```
a.std()
```

1.707825127659933

```
a.std(axis = 0)
```

array([1.5, 1.5, 1.5])

```
a.std(axis = 1)
```

array([0.81649658, 0.81649658])

```
a.var(axis = 0)
```

array([2.25, 2.25, 2.25])

where

✓

The where( ) function **returns the indices of elements** in an input array where the given condition is satisfied.

$a > 7$

```
a = np.array([6,8,7,9,10], dtype= int)  
a
```

```
array([ 6,  8,  7,  9, 10])
```

```
np.where(a > 7) # return index
```

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

```
index = np.where(a > 7)  
print(index) # index  
a[index] # value
```

```
(array([1, 3, 4]),)  
array([ 8,  9, 10])
```

## where

The `where( )` function **returns the indices of elements** in an input array where the given condition is satisfied.

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

1	2	3
4	5	6

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

```
np.where(a>2)
```

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

```
a[np.where(a>2)]
```

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

## 마무리 퀴즈

- 아래와 같은 array를 만듭니다.

```
a = np.arange(-15, 15).reshape(5, 6) ** 2
```

225	196	169	144	121	100
81	64	49	36	25	16
9	4	1	0	1	4
9	16	25	36	49	64
81	100	121	144	169	196

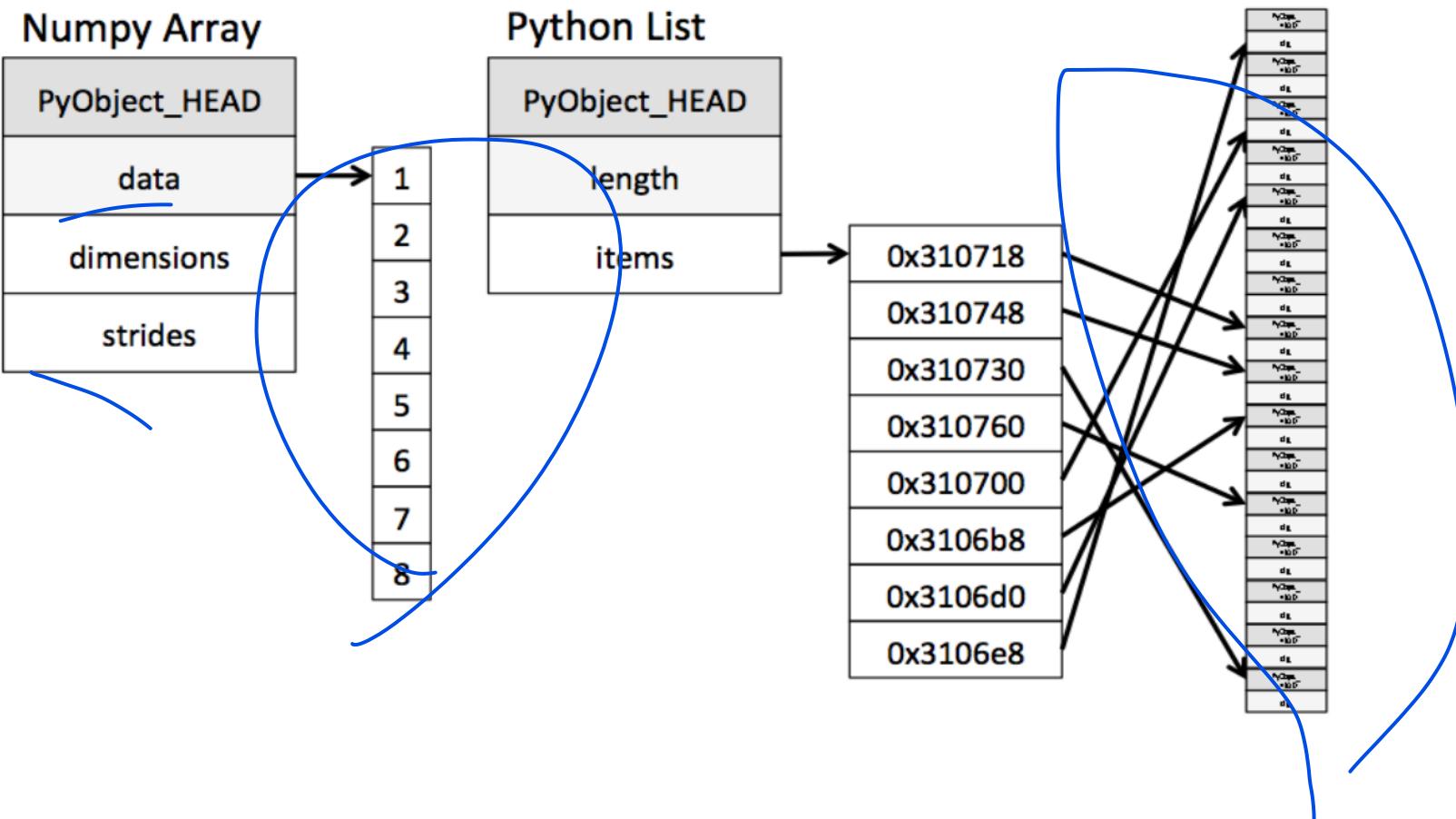
- 각 행별로 최댓값은 각각 얼마입니까?
- 각 열별로 평균은 얼마입니까?

201 x 1

## 파이썬은 왜 느린가? 넘파이는 왜 빠른가?

- Numpy의 메모리 저장 위치 vs Python
- Numpy does not make new one (strides)
- C is faster than any interpreter language

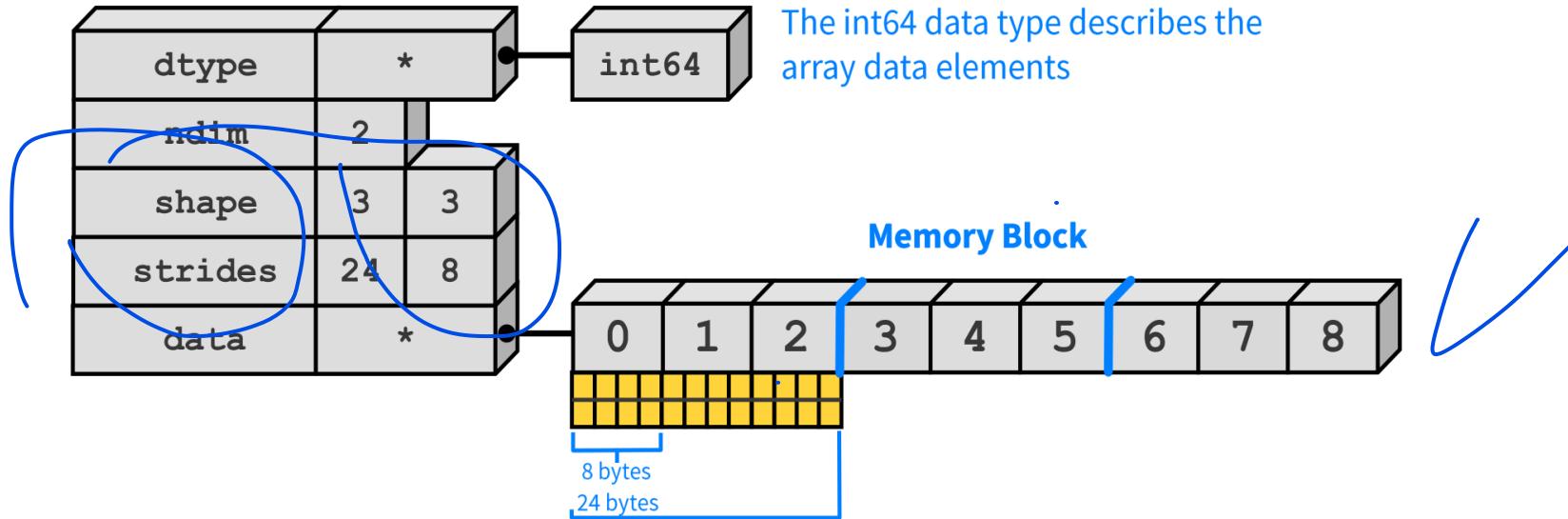
## 메모리 저장위치



출처 <https://cultivo-hy.github.io/python/2019/01/17/파이썬-내부-원리/>

# 데이터 저장구조

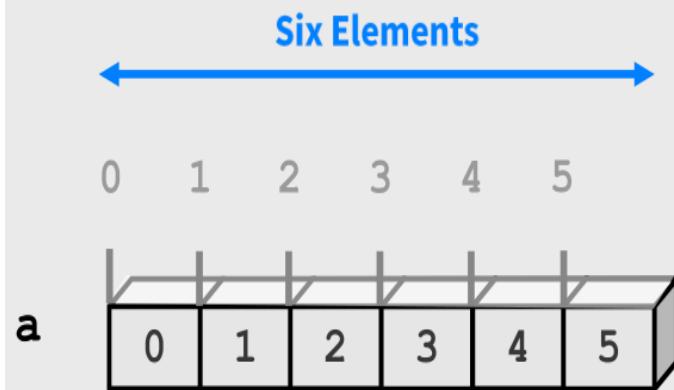
## NDArray Data Structure



## stirde

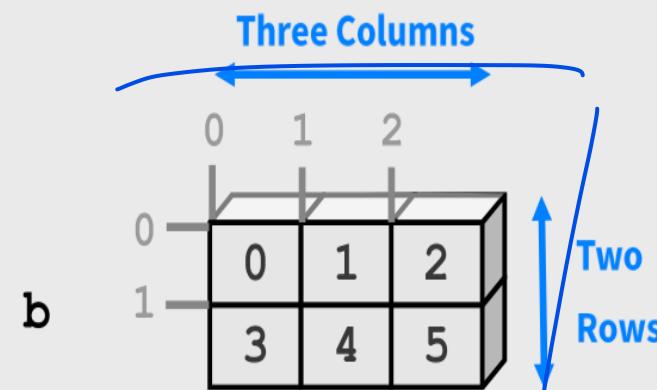
```
>>> a = np.arange(6)
```

```
>>> a
```



```
>>> b = a.reshape(2, 3)
```

```
>>> b
```



This is not a new copy of the data.

The original data does not get reordered.

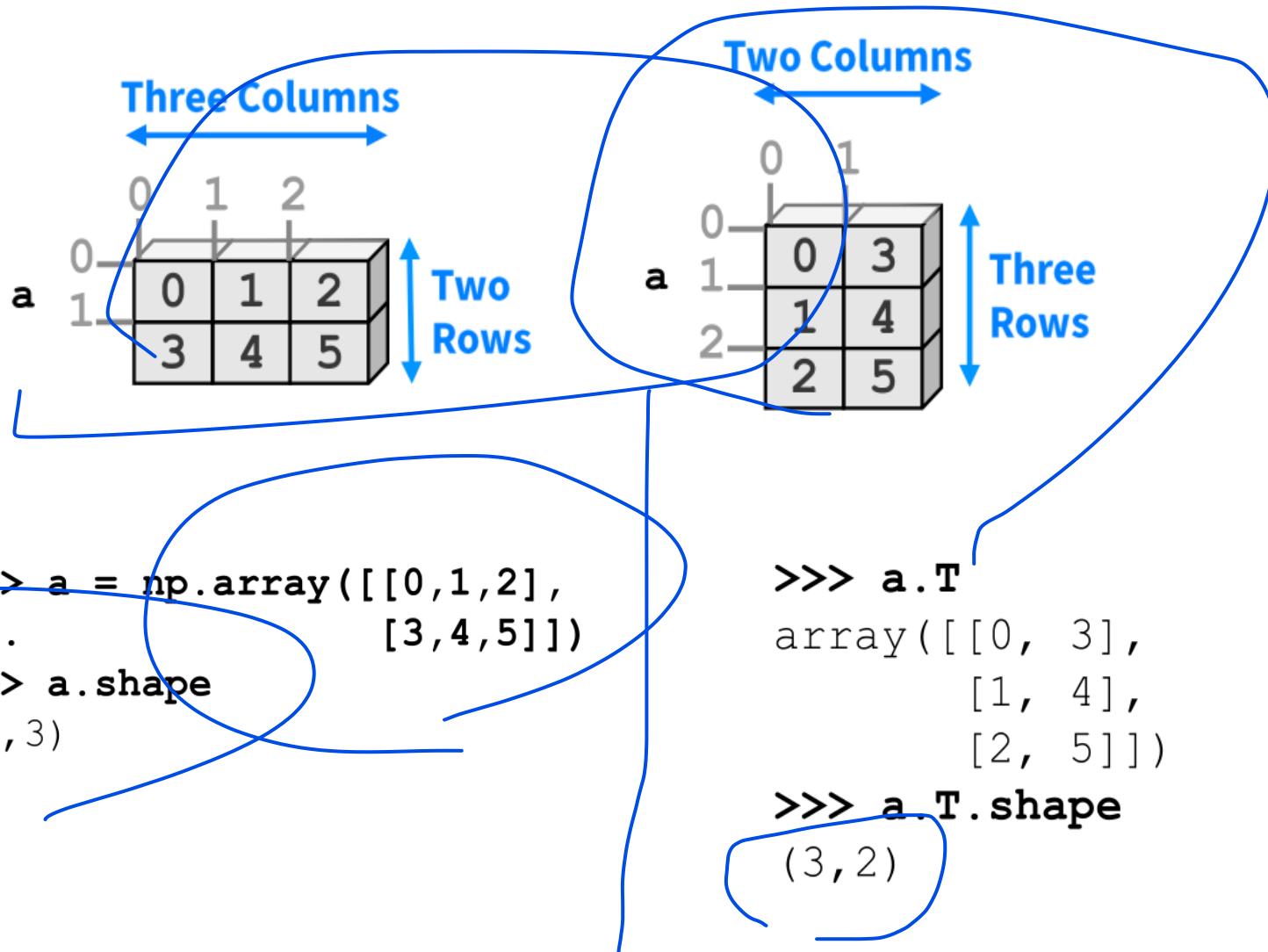
a.strides

(8, )

b.strides

(24, 8)

## 메모리 저장구조



## 메모리 저장구조

