파이썬으로 시작하는 데이터 분석

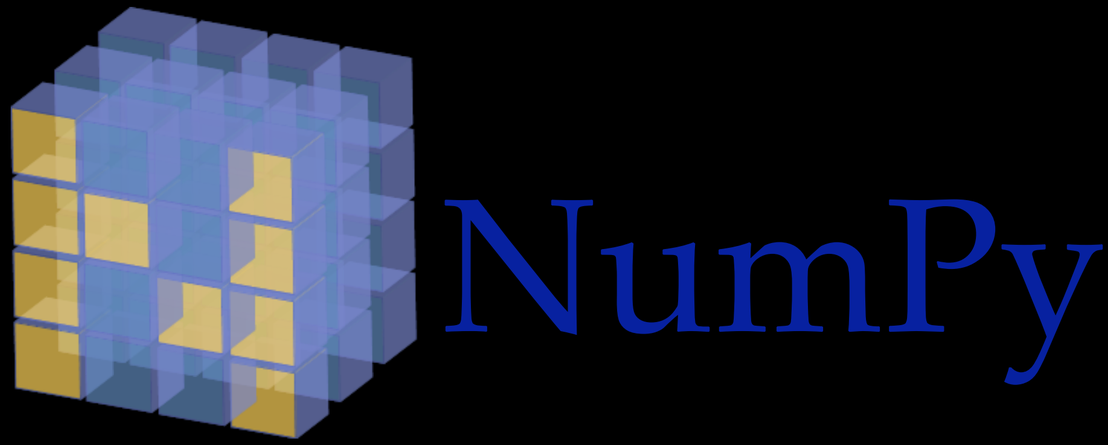
02 NumPy 사용해보기

목차 02 NumPy 사용해보기

1. NumPy 소개

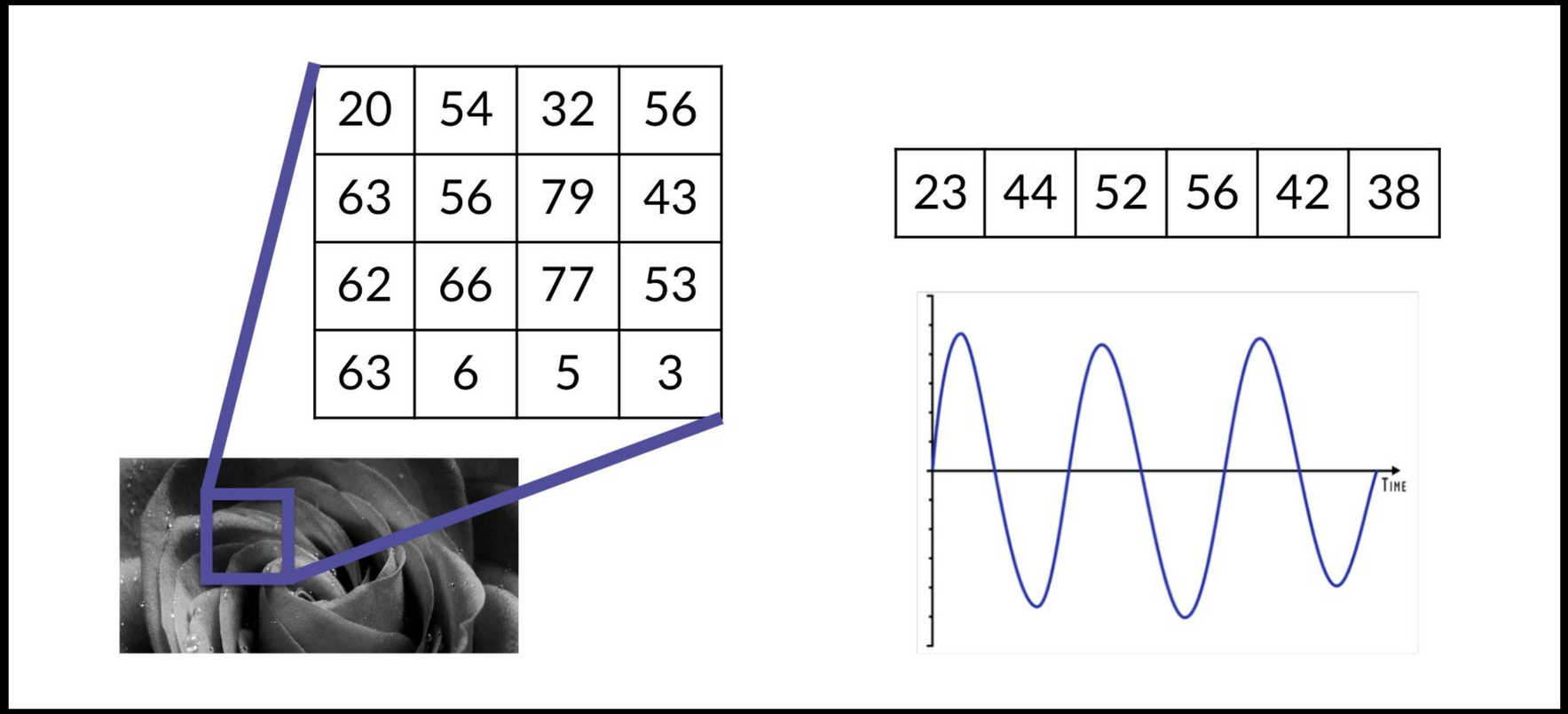
NumPy란

- NumPy : Numerical Python
- Python에서 대규모 다차원 배열을 다룰 수 있게 도와주는 라이브러리



NumPy를 사용하는 이유

- 데이터의 대부분은 숫자 배열로 볼 수 있다.
- NumPy는 파이썬 리스트에 비해 빠른 연산을 지원하고 메모리를 효율적으로 사용한다.



NumPy 배열 만들기


```
list(range(10))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

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

NumPy 배열 만들기


```
np.array([1, 2, 3, 4, 5])
# array([1, 2, 3, 4, 5])
np.array([3, 1.4, 2, 3, 4])
# array([3. , 1.4, 2. , 3. , 4. ])
np.array([[1, 2],
          [3, 4]])
# array([[1, 2],
#        [3, 4]])
np.array([1, 2, 3, 4], dtype='float')
# array([1., 2., 3., 4.])
```

NumPy 배열 데이터 타입 - dtype

- Python List와 다르게 array는 단일 타입으로 구성

```
arr = np.array([1, 2, 3, 4], dtype=float)
arr  # array([1., 2., 3., 4.])
arr.dtype
# dtype('float64')
arr.astype(int)
# array([1, 2, 3, 4])
```

NumPy 배열 데이터 타입 - dtype

dtype	설명	다양한 표현
int	정수형 타입	i, int_, int32, int64, i8
float	실수형 타입	f, float_, float32, float64, f8
str	문자열 타입	str, U, U32
bool	부울 타입	?, bool_

다양한 배열 만들기

```
np.zeros(10, dtype=int)
# array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

np.ones((3, 5), dtype=float)
# array([[1., 1., 1., 1., 1.],
#        [1., 1., 1., 1., 1.],
#        [1., 1., 1., 1., 1.]])
np.arange(0, 20, 2)
# array([ 0, 2, 4, 6, 8, 10, 12, 14, 16, 18])
np.linspace(0, 1, 5)
# array([0. , 0.25, 0.5 , 0.75, 1. ])
```

난수로 채워진 배열 만들기


```
np.random.random((2, 2))
# array([[0.30986539, 0.85863508],
#        [0.89151021, 0.19304196]])

np.random.normal(0, 1, (2, 2))
# array([[ 0.44050683,  0.04912487],
#        [-1.67023947, -0.70982067]])

np.random.randint(0, 10, (2, 2))
# array([[3, 9],
#         [3, 2]])
```

배열의 기초

```
x2 = np.random.randint(10, size=(3, 4))
# array([[2, 2, 9, 0],
         [4, 2, 1, 0],
         [1, 8, 7, 3]])

x2.ndim  # 2
x2.shape  # (3, 4)
x2.size  # 12
x2.dtype  # dtype('int64')
```

$$\begin{bmatrix} 2 & 2 & 9 & 0 \\ 4 & 2 & 1 & 0 \\ 1 & 8 & 7 & 3 \end{bmatrix}$$

2. Indexing / Slicing

찾고 잘라내기 - Indexing

- Indexing : 인덱스로 값을 찾아냄


```
x = np.arange(7)
x[3]
# 3
x[7]
# IndexError: index 7 is out of bounds
x[0] = 10
# array([10, 1, 2, 3, 4, 5, 6])
```

찾고 잘라내기 - Slicing

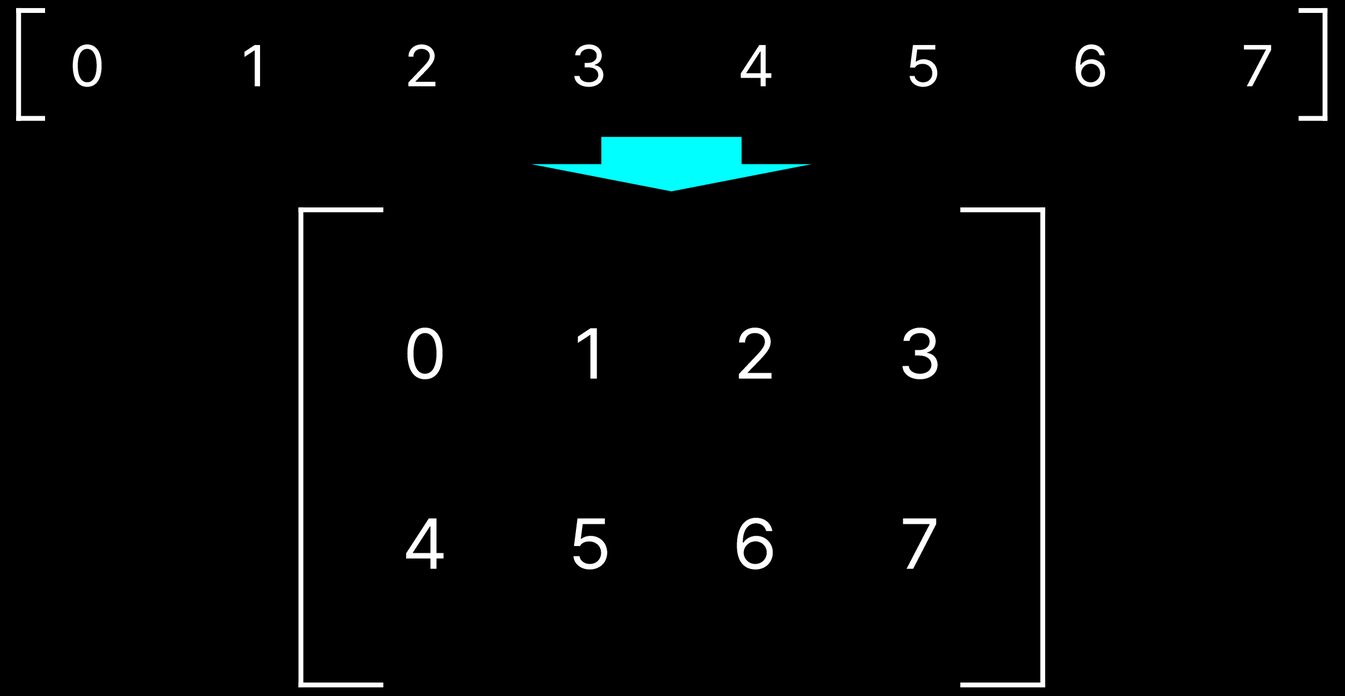
- Slicing : 인덱스 값으로 배열의 부분을 가져옴


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

모양 바꾸기 - reshape

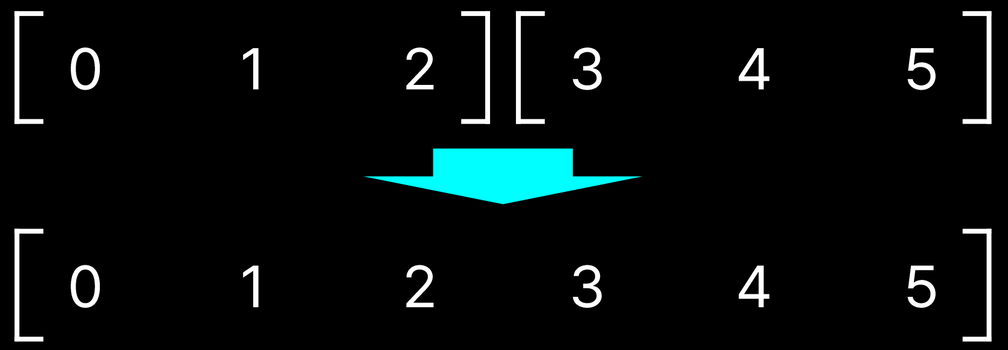
- reshape : array의 shape를 변경

```
x = np.arange(8)
x.shape
# (8,)
x2 = x.reshape((2, 4))
# array([[0, 1, 2, 3],
#        [4, 5, 6, 7]])
x2.shape
# (2, 4)
```



이어 붙이고 나누고 - concatenate

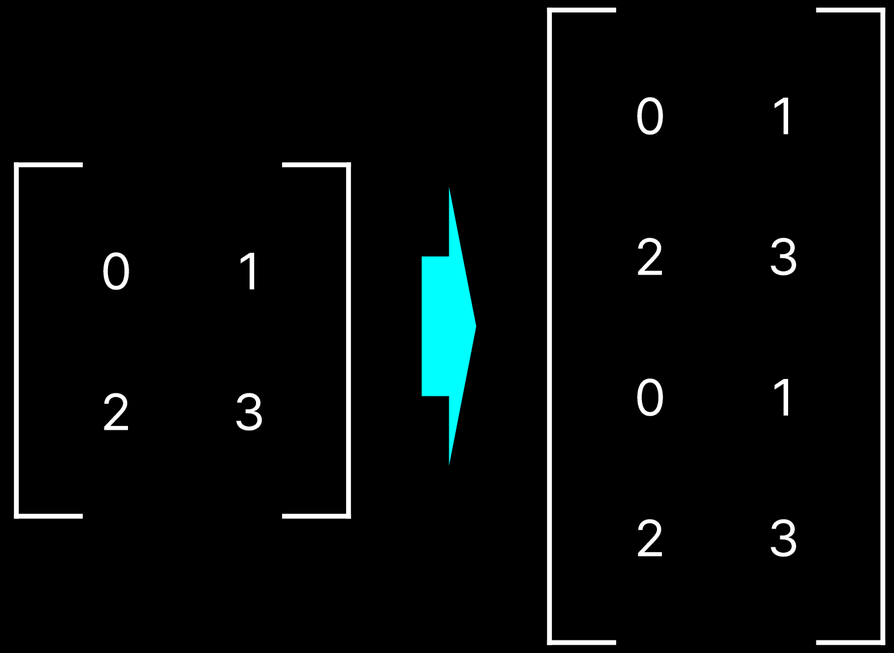
- conactenate : array를 이어 붙임




```
x = np.array([0, 1, 2])
y = np.array([3, 4, 5])
np.concatenate([x, y])
# array([0, 1, 2, 3, 4, 5])
```

이어 붙이고 나누고 - concatenate

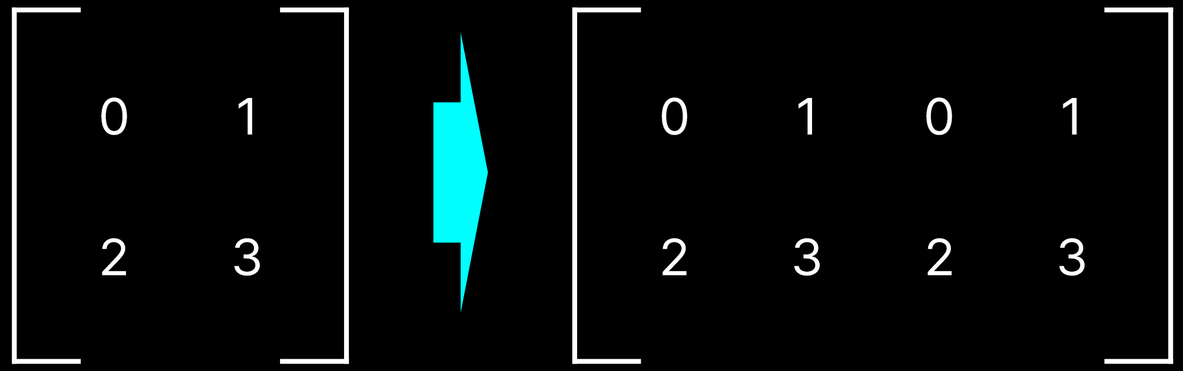
- conactenate : axis 축을 기준으로 이어 붙임
  - axis = 0인 경우




```
matrix = np.arange(4).reshape(2, 2)
np.concatenate([matrix, matrix], axis=0)
```

이어 붙이고 나누고 - concatenate

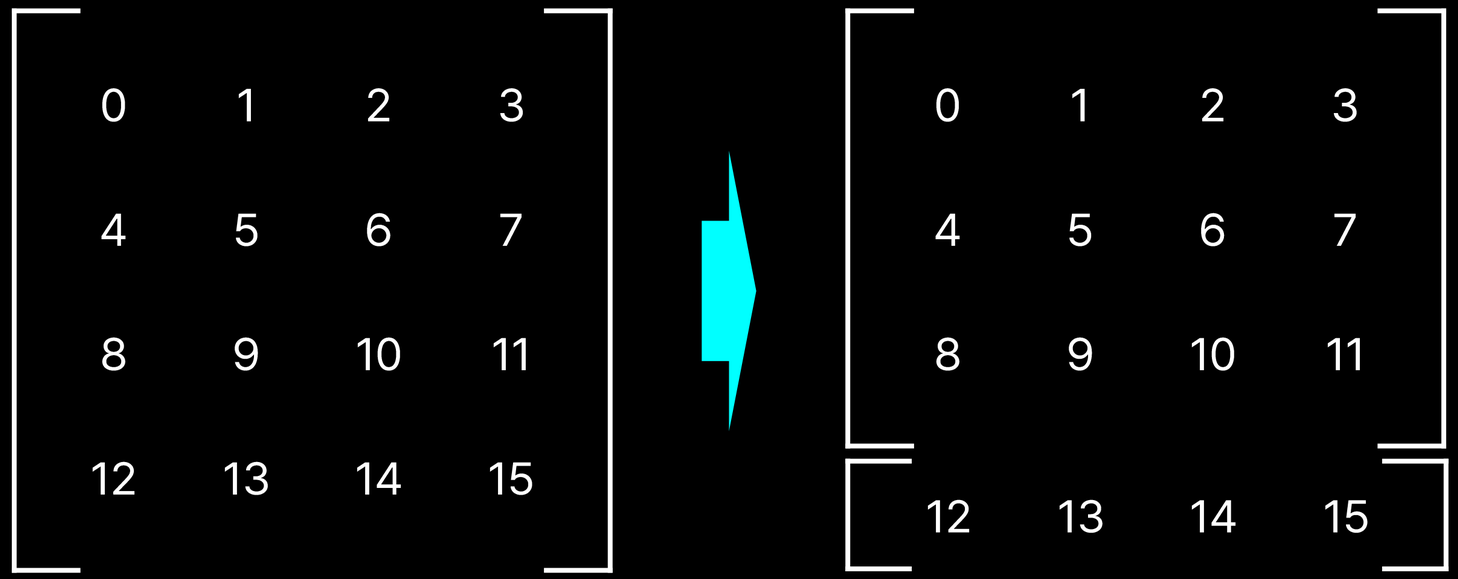
- conactenate : axis 축을 기준으로 이어 붙임
  - axis = 1인 경우




```
matrix = np.arange(4).reshape(2, 2)
np.concatenate([matrix, matrix], axis=1)
```

이어 붙이고 나누고 - split

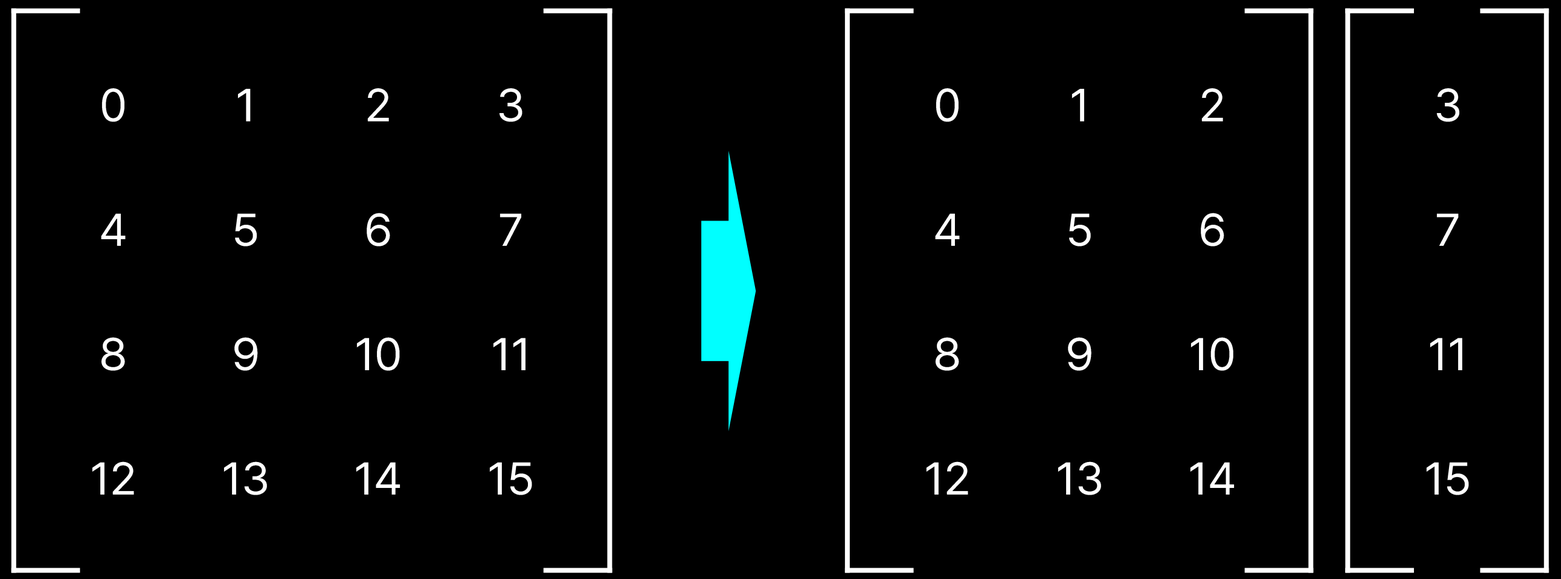
- conactenate : axis 축을 기준으로 분할
  - axis = 0인 경우




```
matrix = np.arange(16).reshape(4, 4)
upper, lower = np.split(matrix, [3], axis=0)
```

이어 붙이고 나누고 - split

- conactenate : axis 축을 기준으로 분할
  - axis = 0인 경우




```
matrix = np.arange(16).reshape(4, 4)
left, right = np.split(matrix, [3], axis=1)
```

3. NumPy 연산

- array의 모든 원소에 5를 더해서 만드는 함수

```
def add_five_to_array(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = values[i] + 5
    return output

values = np.random.randint(1, 10, size=5)
add_five_to_array(values)
```

- 만약 array의 크기가 크다면?

```
big_array = np.random.randint(1, 100, size=10000000)

add_five_to_array(big_array)
# 5.3 s ± 286 ms per loop (mean ± std. dev. of 7 runs,
5 loops each)

big_array + 5
# 33.5 ms ± 1.94 ms per loop (mean ± std. dev. of 7
runs, 5 loops each)
```

기본 연산

- Array는 +, -, *, / 에 대한 기본 연산을 지원

```
x = np.arange(4)
# array([0, 1, 2, 3])
x + 5
# array([5, 6, 7, 8])
x - 5
# array([-5, -4, -3, -2])
x * 5
# array([ 0,  5, 10, 15])
x / 5
# array([0. , 0.2, 0.4, 0.6])
```

행렬간 연산

- 다차원 행렬에서도 적용 가능

```
x = np.arange(4).reshape((2, 2))
y = np.random.randint(10, size=(2, 2))


x + y
# array([[1, 7],
         [6, 5]])
x - y
# array([[-1, -5],
         [-2,  1]])
```

$$x = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$$

$$y = \begin{bmatrix} 1 & 6 \\ 4 & 2 \end{bmatrix}$$

브로드캐스팅

- Braodcasting : shape가 다른 array끼리 연산

matrix + 5

2	4	2
6	5	9
9	4	7

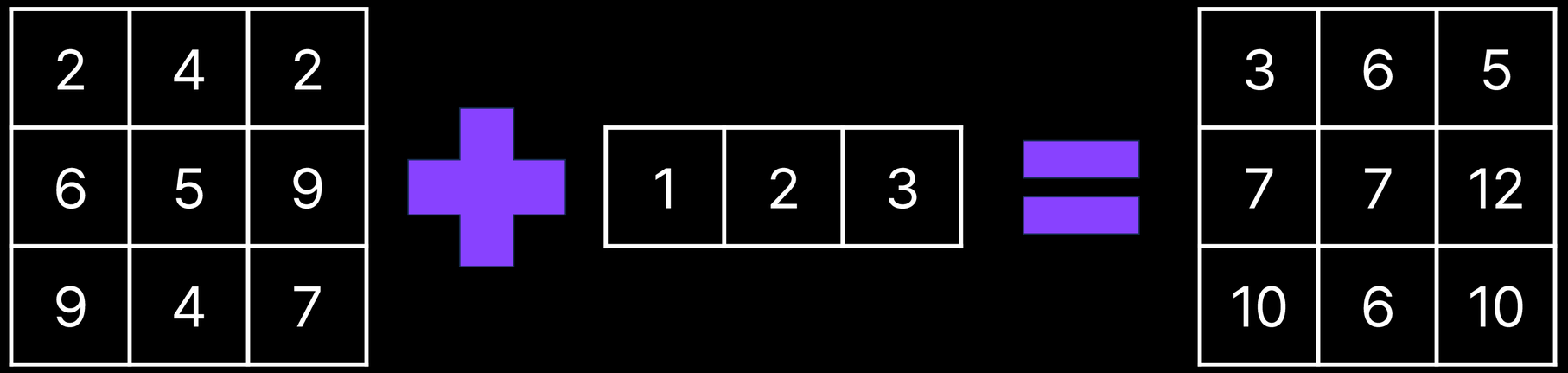
+ 5 =

7	9	7
11	10	14
14	9	12

브로드캐스팅

- Braodcasting : shape가 다른 array끼리 연산

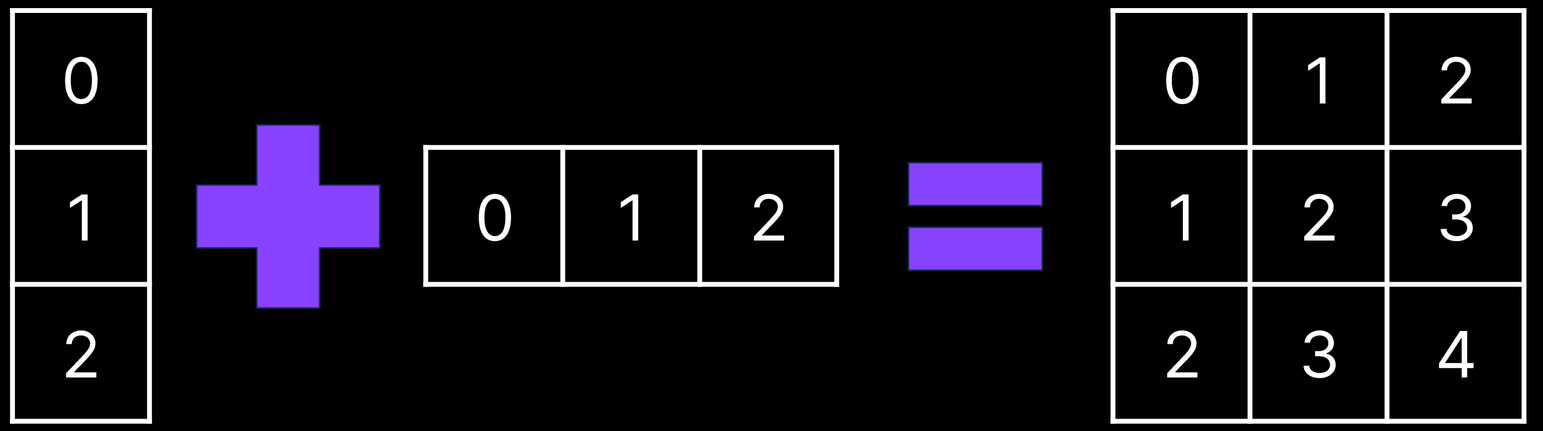
```
matrix + np.array([1, 2, 3])
```



브로드캐스팅

- Braodcasting : shape가 다른 array끼리 연산

```
np.arange(3).reshape((3,1)) + np.arange(3)
```



4. 집계함수 & 마스킹 연산

집계함수

- 집계 : 데이터에 대한 요약 통계

```
x = np.arange(8).reshape((2, 4))
np.sum(x)
# 28
np.min(x)
# 0
np.max(x)
# 7
np.mean(x)
# 3.5
```

집계함수

- 집계 : 데이터에 대한 요약 통계

```
x = np.arange(8).reshape((2, 4))
np.sum(x, axis=0)
# array([ 4,  6,  8, 10])
np.sum(x, axis=1)
# array([ 6, 22])
```

마스킹 연산

- 마스킹 연산 : True, False array를 통해서 특정 값들을 뽑아내는 방법

```
x = np.arange(5)
# array([0, 1, 2, 3, 4])
x < 3
# array([ True,  True,  True, False, False])
x > 5
# array([False, False, False, False, False])
x[x < 3]
# array([0, 1, 2])
```