

2주차(3/3)

넴파이 튜토리얼 2

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

넴파이 튜토리얼 2

- 학습 목표
 - 기계학습에서 왜 넴파이를 사용하는지 이해한다.
 - 넴파이 개념과 기본적인 사용법을 익힌다.
- 학습 내용
 - 브로드캐스팅
 - 배열의 축 다루기
 - 난수 배열


브로드캐스팅

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

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

브로드캐스팅

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




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

ValueError

Traceback (most recent call last)

<ipython-input-3-0ad41c013129> in <module>()
 1 a = np.array([1,2,3])
 2 b = np.array([1,2])
----> 3 a + b

ValueError: operands could not be broadcast together with shapes (3,) (2,)



브로드캐스팅

- 브로드캐스팅: “널리 전하다”

브로드캐스팅

- 브로드캐스팅: “널리 전하다”

$$A + B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} + (1 \quad 0 \quad 1)$$

브로드캐스팅

- 브로드캐스팅: “널리 전하다”

$$\begin{aligned} A + B &= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} + (1 \ 0 \ 1) \\ &= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{aligned}$$

브로드캐스팅

브로드캐스팅

- 브로드캐스팅: “널리 전하다”

$$A + B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

```
a = np.array(np.arange(1, 13)).reshape(4, 3)
b = np.array([1, 0, 1])
print(a + b)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```


브로드캐스팅

- 브로드캐스팅: “널리 전하다”
- 다음은 어떻게 브로드캐스팅을 할까요?

$$A + B = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} + (0 \quad 1 \quad 2)$$

브로드캐스팅

- 브로드캐스팅: “널리 전하다”
- 예제: **`shape(3, 1) + shape(1, 3) = shape(3, 3)`**

브로드캐스팅

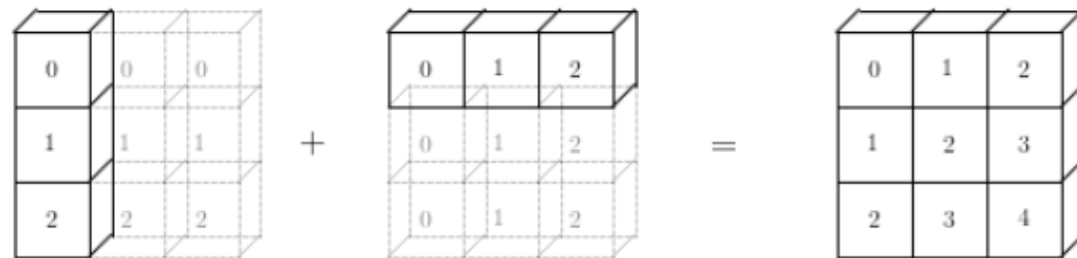
$$\begin{aligned} A + B &= \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} + (0 \quad 1 \quad 2) \\ &= \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{pmatrix} \end{aligned}$$

브로드캐스팅

- 브로드캐스팅: “널리 전하다”
- 예제: **shape(3, 1) + shape(1, 3) = shape(3, 3)**

$$\begin{aligned} A + B &= \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} + (0 \quad 1 \quad 2) \\ &= \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{pmatrix} \end{aligned}$$

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

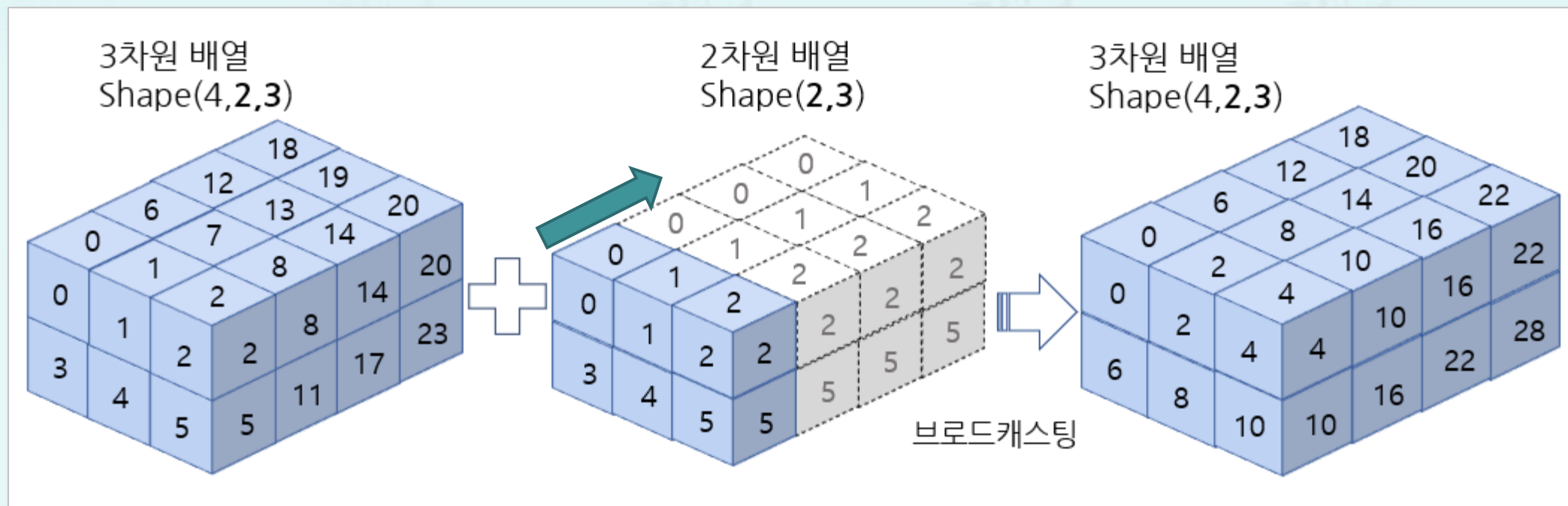


`np.arange(3).reshape(3, 1) + np.arange(3)`

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

브로드캐스팅

- 브로드캐스팅: “널리 전하다”
- 예제: **shape(3,4,2) + shape(4,2) = shape(3, 4, 2)**



배열의 축 Axis 다루기

1차원 배열

axis 0 →



Shape(3,)

2차원 배열

axis 1 →

axis 0 ↓

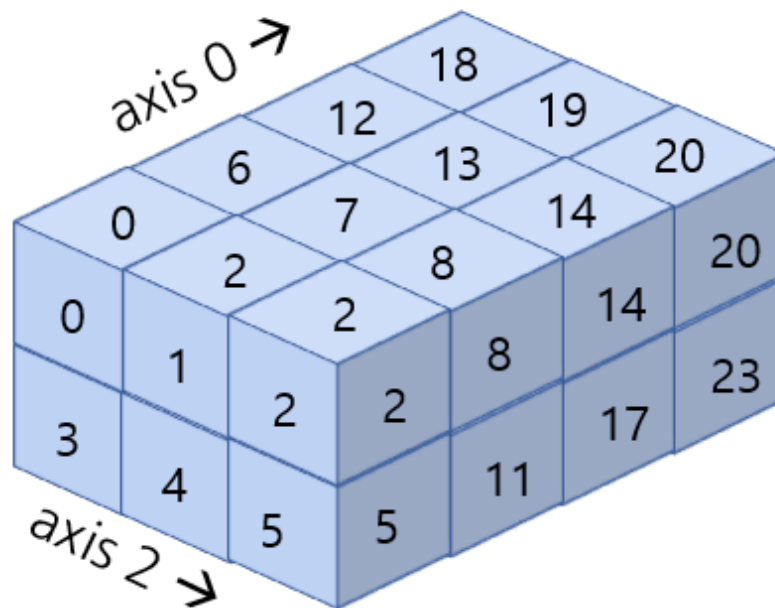


Shape(2, 3)

3차원 배열

axis 0 →

axis 1 ↓



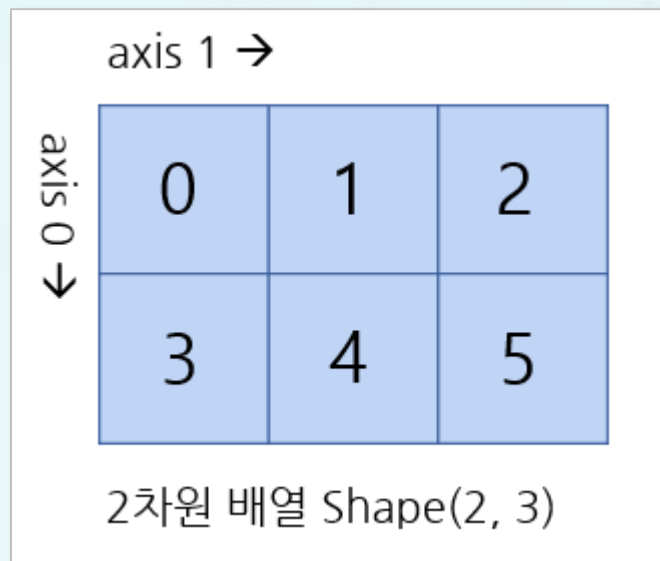
Shape(4, 2, 3)

배열의 축 Axis 다루기

- 2차원 배열 Axis

`np.sum(a) = 15`

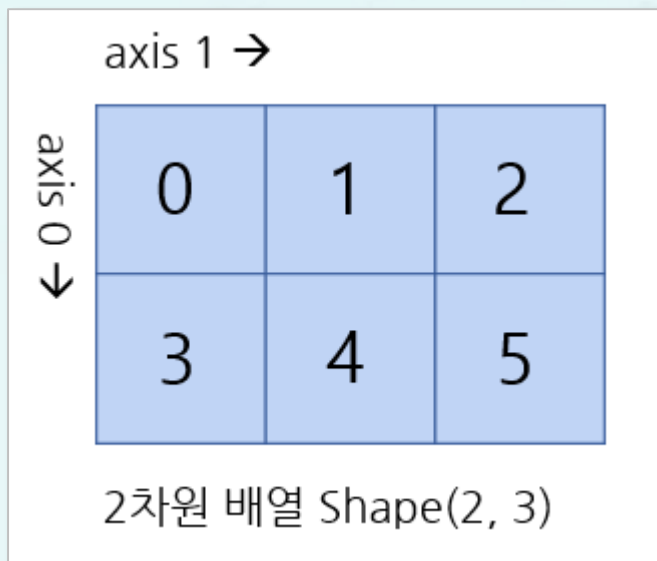
`np.sum(a, axis = 0) = ?`



배열의 축 Axis 다루기

- 2차원 배열 Axis

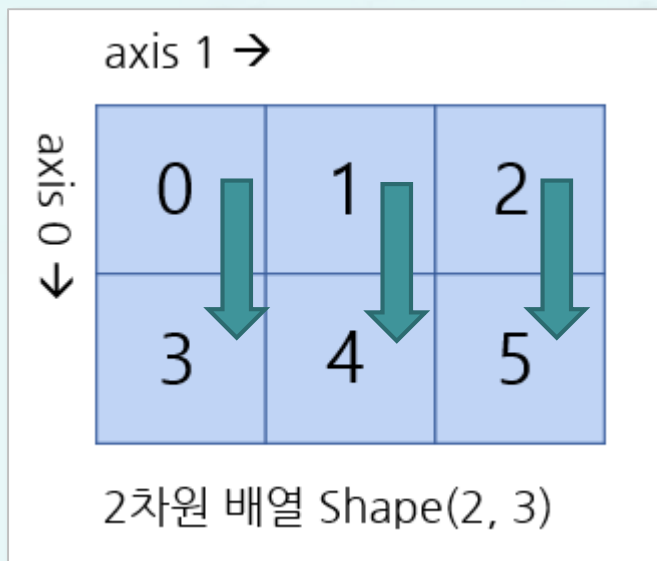
`np.sum(a, axis = 0) = ?`



배열의 축 Axis 다루기

- 2차원 배열 Axis

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



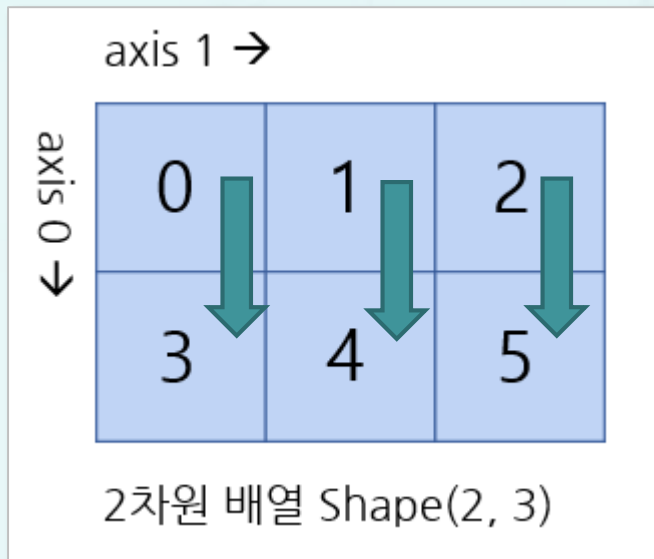
```
print('sum(axis=0):', np.sum(a, axis=0))
```

```
sum(axis=0): [3 5 7]
```


배열의 축 Axis 다루기

- 2차원 배열 Axis

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



```
print('sum(axis=0):', np.sum(a, axis=0))
```

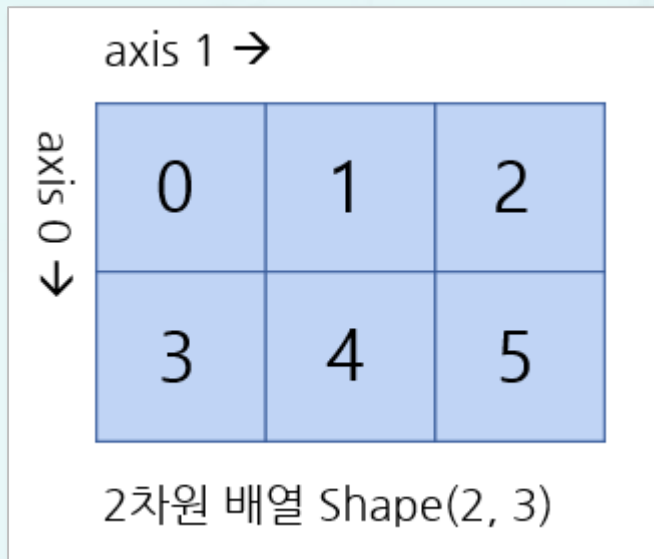
```
sum(axis=0): [3 5 7]
```

형상: (3,)

배열의 축 Axis 다루기

- 2차원 배열 Axis

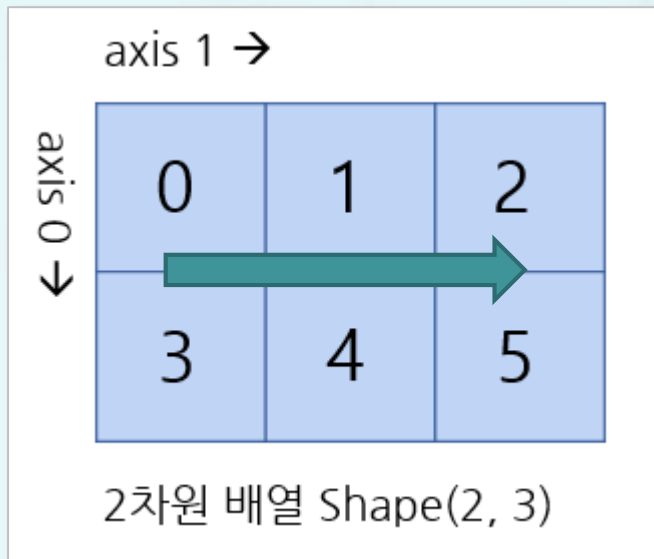
`np.max(a, axis = 1) = ?`



배열의 축 Axis 다루기

- 2차원 배열 Axis

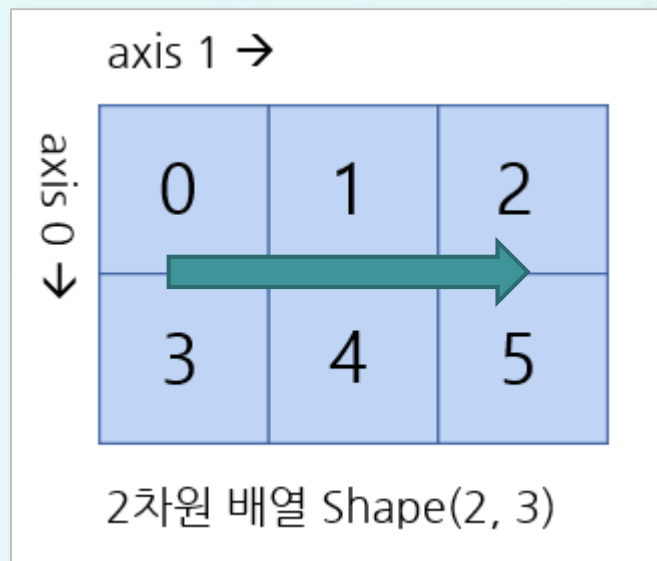
`np.max(a, axis = 1)` = ?



배열의 축 Axis 다루기

- 2차원 배열 Axis

np.max(a, axis = 1) = ?



```
print(a)
print('max(axis=1):', np.max(a, axis=1))
```

```
[[0 1 2]
 [3 4 5]]
max(axis=1): [2 5]
```

배열의 축Axis다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열

2차원 배열

Shape(4, 2)


0	1
2	3
4	5
6	7

배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열

2차원 배열
Shape(4, 2)

0	1
2	3
4	5
6	7



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


```
type:<class 'numpy.ndarray'>, size:8  
shape:(4, 2), ndim/rank:2, dtype:int32  
Array's Data:  
[[0 1]  
 [2 3]  
 [4 5]  
 [6 7]]
```

배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열

2차원 배열
Shape(4, 2)

0	1
2	3
4	5
6	7



```
a = np.arange(8).reshape(4,2)
pprint(a)
```

```
type:<class 'numpy.ndarray'>, size:8
shape:(4, 2), ndim/rank:2, dtype:int32
Array's Data:
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
```

배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열


2차원 배열
Shape(4, 2)

0	1
2	3
4	5
6	7



3차원 배열
Shape(3, 4, 2)

		16	17
	8	9	
0	1		19
		11	
2	3		21
		13	
4	5		23
		15	
6	7		



```
a = np.arange(24).  
pprint(a)
```

```
type:<class 'numpy.ndarray'>, size:24  
shape:(3, 4, 2), ndim/rank:3, dtype:int32
```

Array's Data:

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

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

```
[[16 17]  
 [18 19]  
 [20 21]  
 [22 23]]]
```


배열의 축Axis다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열


2차원 배열
Shape(4, 2)

0	1
2	3
4	5
6	7



3차원 배열
Shape(3, 4, 2)

		16	17
	8	9	
0	1		19
		11	21
2	3		
		13	23
4	5		
		15	
6	7		



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

```
type:<class 'numpy.ndarray'>, size:24
shape:(3, 4, 2), ndim/rank:3, dtype:int32
```

Array's Data:

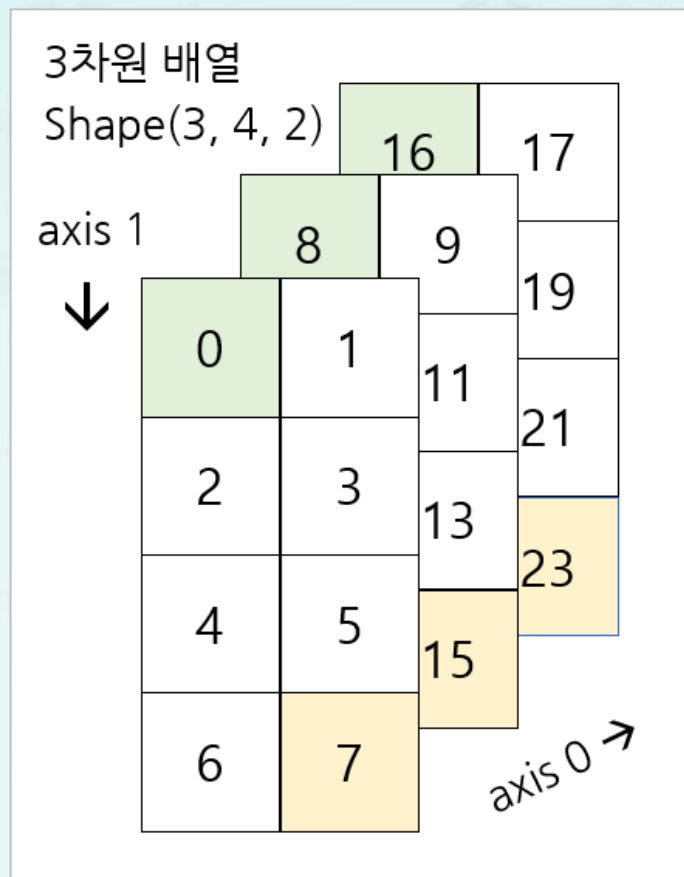
```
[[[ 0  1]
   [ 2  3]
   [ 4  5]
   [ 6  7]]
```

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

```
[[16 17]
 [18 19]
 [20 21]
 [22 23]]]
```

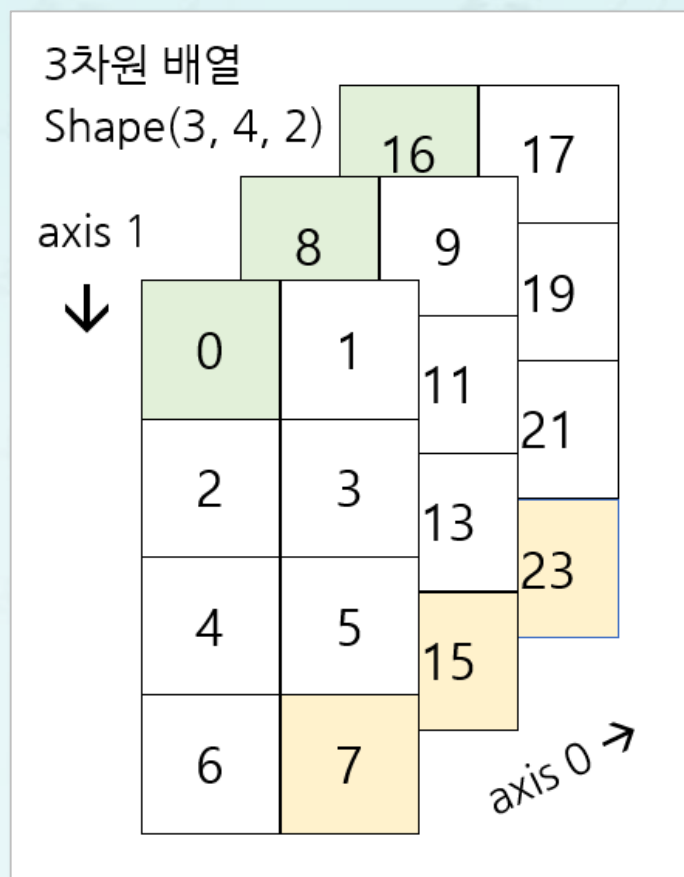
배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열
 - `np.sum(a, axis = 0) = ?`



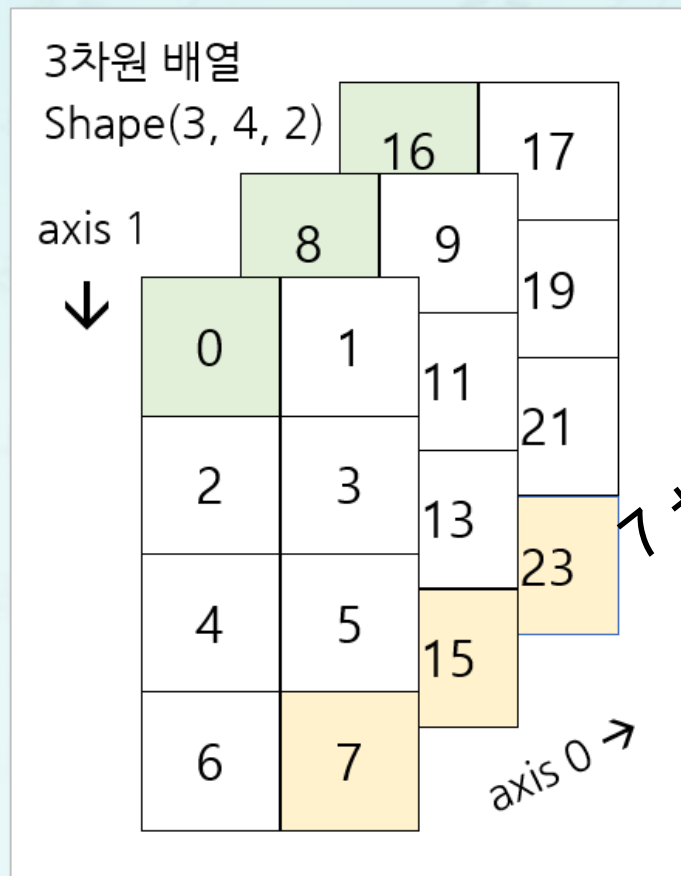
배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열
 - `np.sum(a, axis = 0) = ?`



배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열
 - `np.sum(a, axis = 0) = ?`



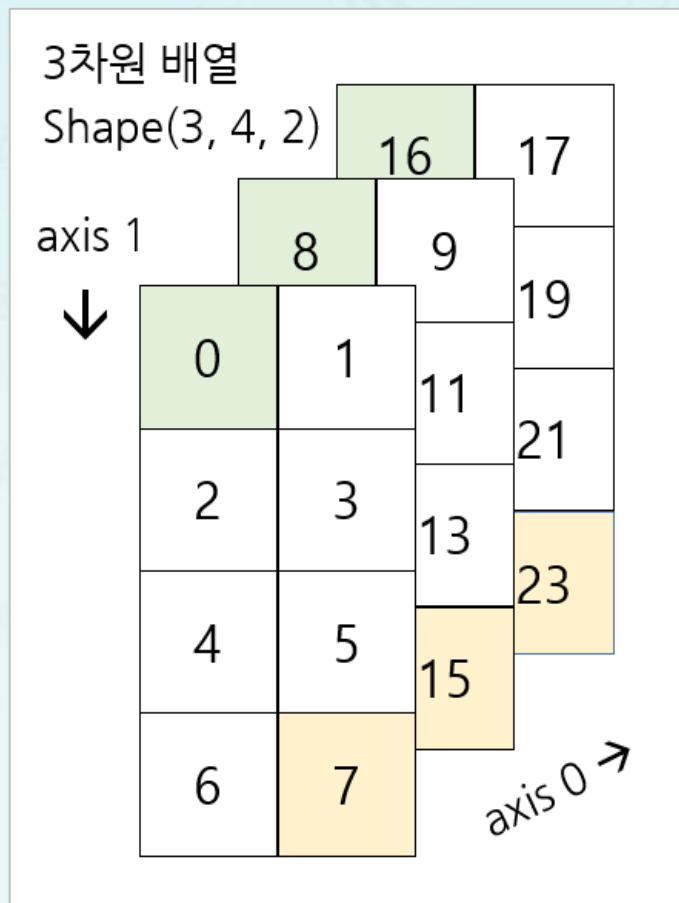
$$7 + 15 + 23 = 45$$

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

```
array([[24, 27],  
       [30, 33],  
       [36, 39],  
       [42, 45]])
```

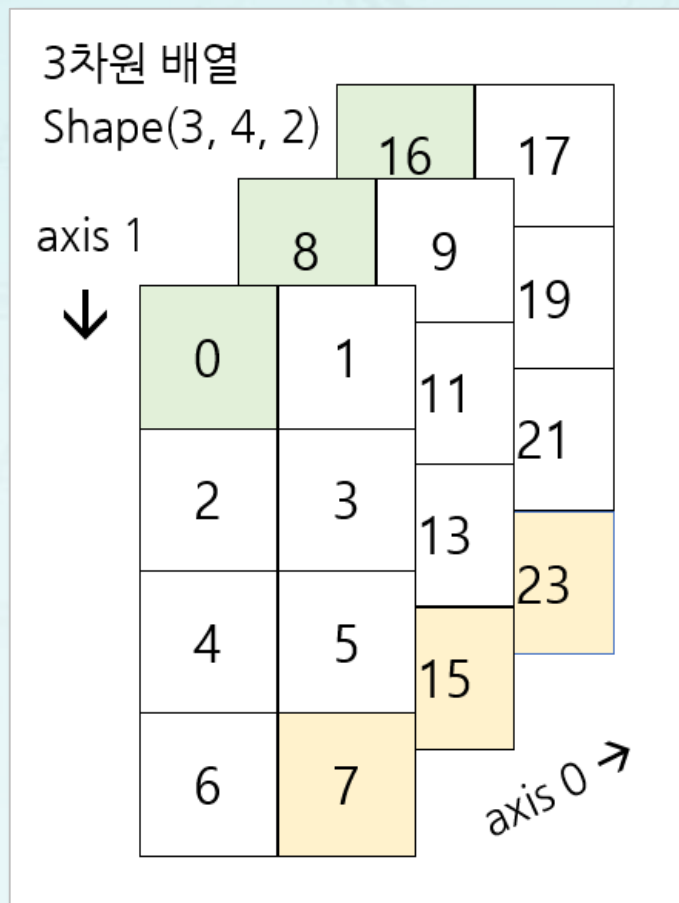
배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열
 - `np.min(a, axis = 1) = ?`



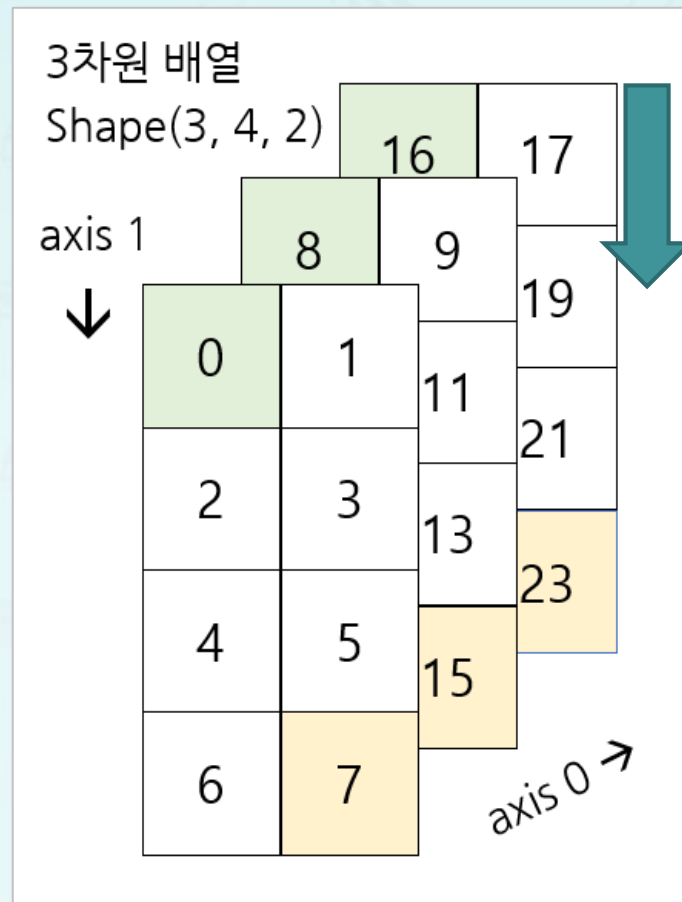
배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열
 - `np.min(a, axis = 1) = ?`



배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열
 - `np.min(a, axis = 1) = ?`



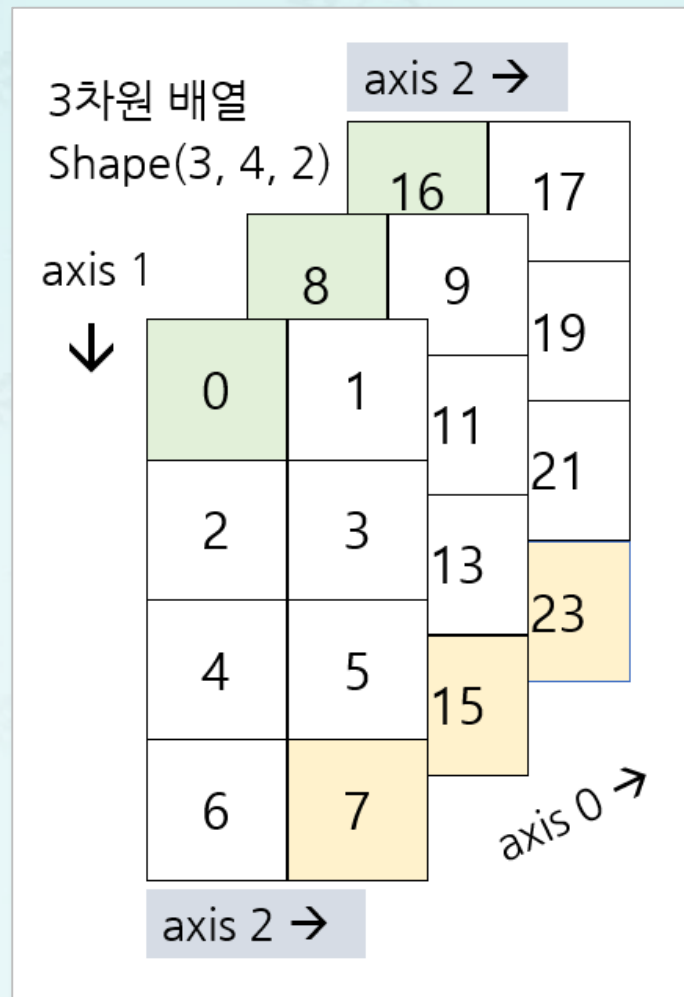
```
np.min(a, axis = 1)
```

```
array([[ 0,  1],  
       [ 8,  9],  
       [16, 17]])
```

배열의 축 Axis 다루기

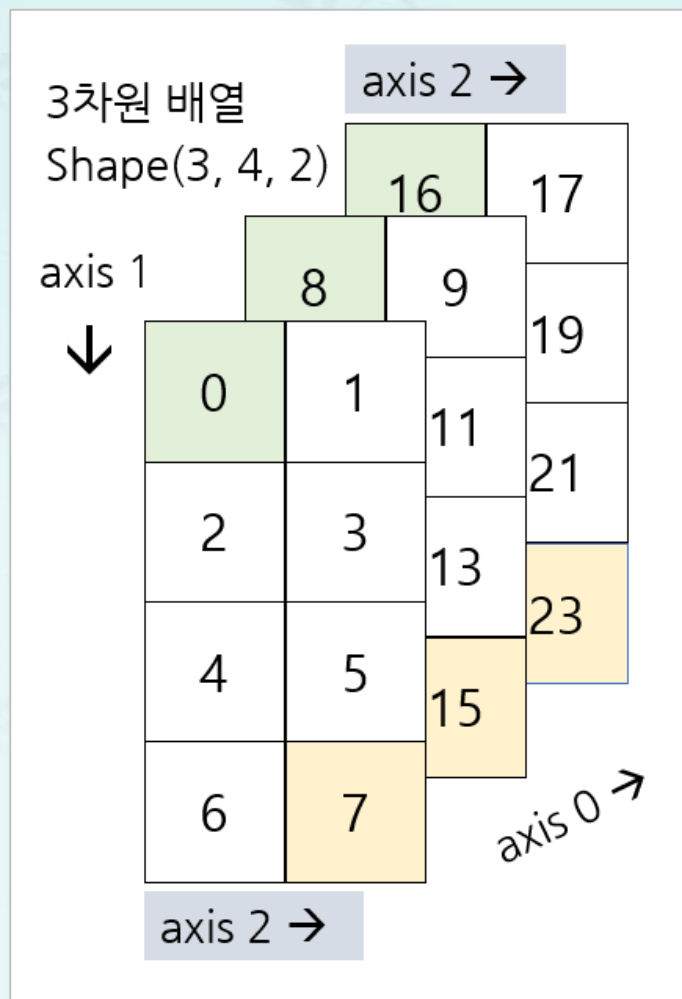
■ 3차원 배열

- 2차원 배열을 두 개 이상 모아 둔 배열
- `np.max(a, axis = 2) = ?`



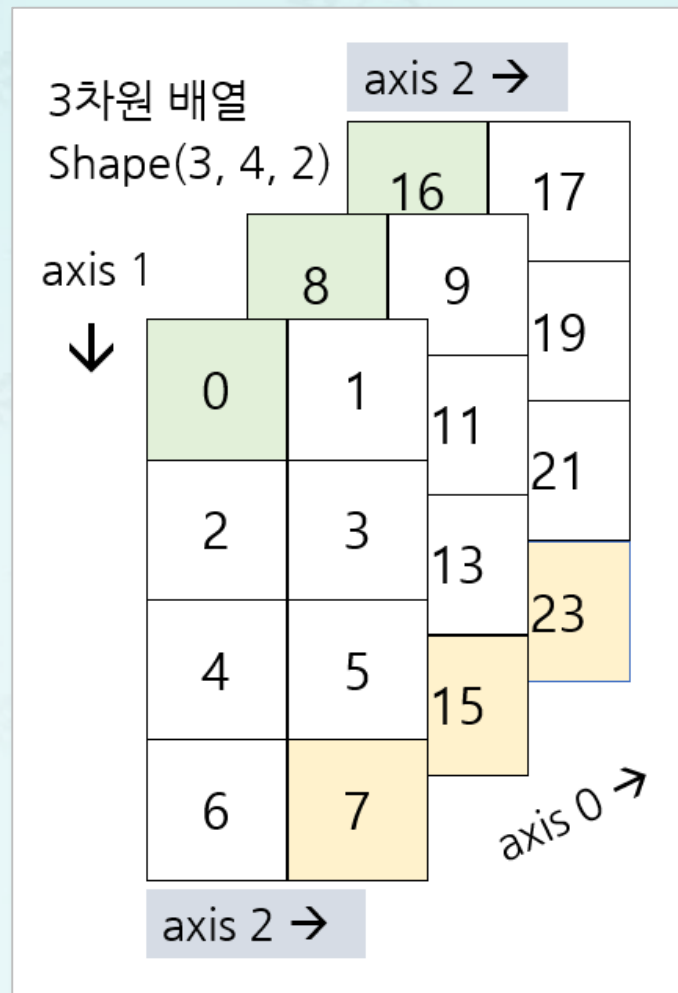
배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열
 - `np.max(a, axis = 2) = ?`



배열의 축 Axis 다루기

- 3차원 배열
 - 2차원 배열을 두 개 이상 모아 둔 배열
 - `np.max(a, axis = 2) = ?`



```
np.max(a, axis = 2)
```

```
array([[ 1,  3,  5,  7],  
       [ 9, 11, 13, 15],  
       [17, 19, 21, 23]])
```

난수 배열의 생성

- **numpy.random** 모듈
 - **randint(low, high=None, size=None, dtype='l')**
 - **normal(loc=0.0, scale=1.0, size=None)**
 - **random(size=None)**

난수 배열의 생성

- **numpy.random** 모듈
 - **randint(low, high=None, size=None, dtype='i')**
정수 표본을 추출하여 배열을 반환

- 예시

```
np.random.randint(-5, 5, size=(2, 4))
```

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

난수 배열의 생성

- **numpy.random** 모듈
 - **randint(low, high=None, size=None, dtype='i')**
정수 표본을 추출하여 배열을 반환

- 예시

```
np.random.randint(-5, 5, size=(2, 4))  
  
array([[ 4, -4,  2, -3],  
       [-3,  0,  1,  0]])
```



```
np.random.randint(2, size=(8))  
  
array([1, 0, 0, 1, 1, 0, 0, 0])
```

난수 배열의 생성

- **numpy.random** 모듈
 - **normal(loc=0.0, scale=1.0, size=None)**
정규 분포 확률 밀도에서 표본을 추출하여 배열로 반환
정규 분포의 평균(**loc**), 표준편차(**scale**)을 지정할 수 있음.

난수 배열의 생성

- **numpy.random** 모듈

- **normal(loc=0.0, scale=1.0, size=None)**

정규 분포 확률 밀도에서 표본을 추출하여 배열로 반환
정규 분포의 평균(**loc**), 표준편차(**scale**)을 지정할 수 있음.

```
mean = 0
std = 1
np.random.normal(mean, std, (2, 3))

array([[ 0.18330292, -1.41009505,  0.29135612],
       [ 1.1569196 ,  1.00073134,  0.79086678]])
```

난수 배열의 생성

- **numpy.random** 모듈

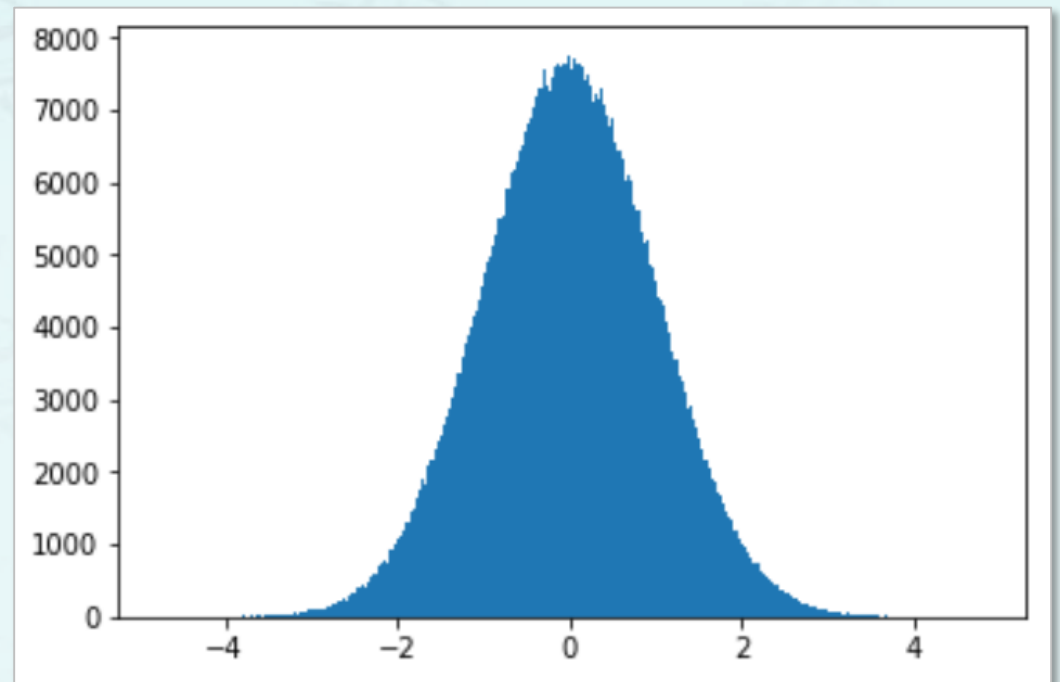
- **normal(loc=0.0, scale=1.0, size=None)**

정규 분포 확률 밀도에서 표본을 추출하여 배열로 반환
정규 분포의 평균(**loc**), 표준편차(**scale**)을 지정할 수 있음.

```
mean = 0
std = 1
np.random.normal(mean, std, (2, 3))

array([[ 0.18330292, -1.41009505,  0.29135612],
       [ 1.1569196 ,  1.00073134,  0.79086678]])
```

```
data = np.random.normal(mean, std, 1000000)
plt.hist(data, bins=500)
plt.show()
```



난수 배열의 생성

- **numpy.random** 모듈
 - **random(size=None)**
[0., 1.) 범위의 난수를 균등 분포에서 표본 추출하여 배열로 반환

난수 배열의 생성

- **numpy.random** 모듈
 - **random(size=None)**
[0., 1.) 범위의 난수를 균등 분포에서 표본 추출하여 배열로 반환

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

```
array([[0.75939176, 0.3229062 , 0.98665638],  
       [0.56294242, 0.82301076, 0.90965698]])
```

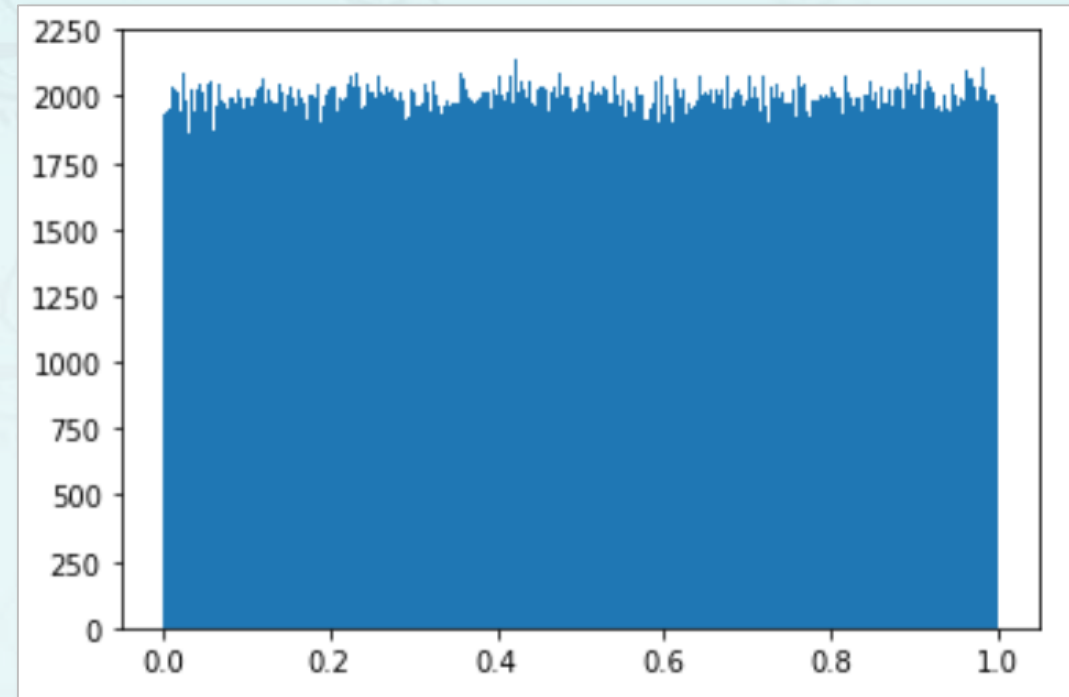
난수 배열의 생성

- **numpy.random** 모듈
 - **random(size=None)**
[0., 1.) 범위의 난수를 균등 분포에서 표본 추출하여 배열로 반환

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

```
array([[0.75939176, 0.3229062 , 0.98665638],  
       [0.56294242, 0.82301076, 0.90965698]])
```

```
data = np.random.random(1000000)  
plt.hist(data, bins=500)  
plt.show()
```



넴파이 튜토리얼 2

- 학습 정리
 - 넴파이를 사용하는 이유
 - 넴파이 개념과 사용법
 - 브로드캐스팅
 - 배열의 축 다루기
 - 난수 배열
- 차시 예고
 - **3-1** 인공뉴론의 동작 원리

2주차(3/3)

넴파이 튜토리얼 2

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

여러분 곁에 항상 열려 있는 K-MOOC 강의실에서 만나 뵙기를 바랍니다.