

NUMPY

FOR MACHINE LEARNING

PART I

NUMPY

NUMPY PACKAGE

3

◆ NumPy (Numerical Python)

❖ 수학 및 과학 연산을 위한 파이썬 패키지

- C언어로 구현된 파이썬 라이브러리로 고성능의 수치계산 위해 제작
- **벡터 및 행렬 연산에 있어서 매우 편리한 기능을 제공**
- 데이터 분석 및 시각화 패키지인 pandas, matplotlib에 내부 데이터 처리에 사용
- 머신러닝 및 딥러닝 패키지인 tensorflow, keras, pytorch에 내부 데이터 처리에 사용
- 기본적으로 **array**라는 단위로 데이터를 관리하며 이에 대해 연산 수행

NUMPY PACKAGE

4

◆ NumPy (Numerical Python)

❖ 배열(ARRAY)란 => C, Java, C#

- 순차적으로 데이터를 저장하는 자료형이며 수학의 행렬에서 유래된 개념
- 서로 연관있는 데이터를 하나의 변수명으로 저장
- 같은 타입의 데이터를 연속된 메모리 공간에 저장
- 원소 개수 변경 불가
- 구성하는 각각의 값을 배열 요소/원소(element)
- 위치를 가리키는 숫자는 인덱스(index)
- 같은 종류의 데이터를 많이 다뤄야 하는 경우에 사용

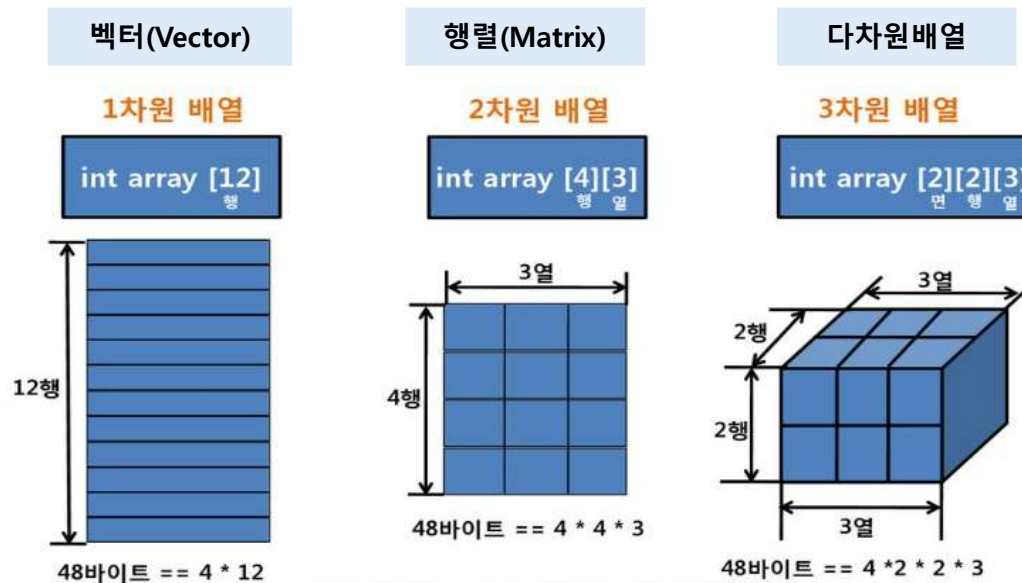
NUMPY PACKAGE

5

◆ NumPy (Numerical Python)

❖ 배열 (ARRAY)란

- C/Java 언어



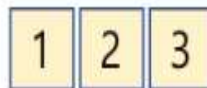
NUMPY PACKAGE

6

◆ NumPy (Numerical Python)

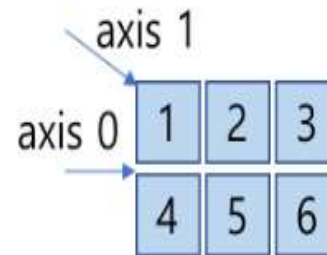
❖ 배열 (ARRAY)란

벡터(Vector)



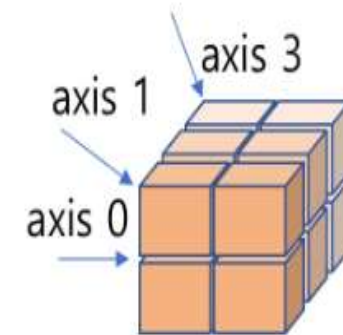
1D array

행렬(Matrix)



2D array

다차원배열



3D array

NUMPY PACKAGE

7

◆ NumPy (Numerical Python)

❖ 배열(ARRAY) vs 리스트(LIST)

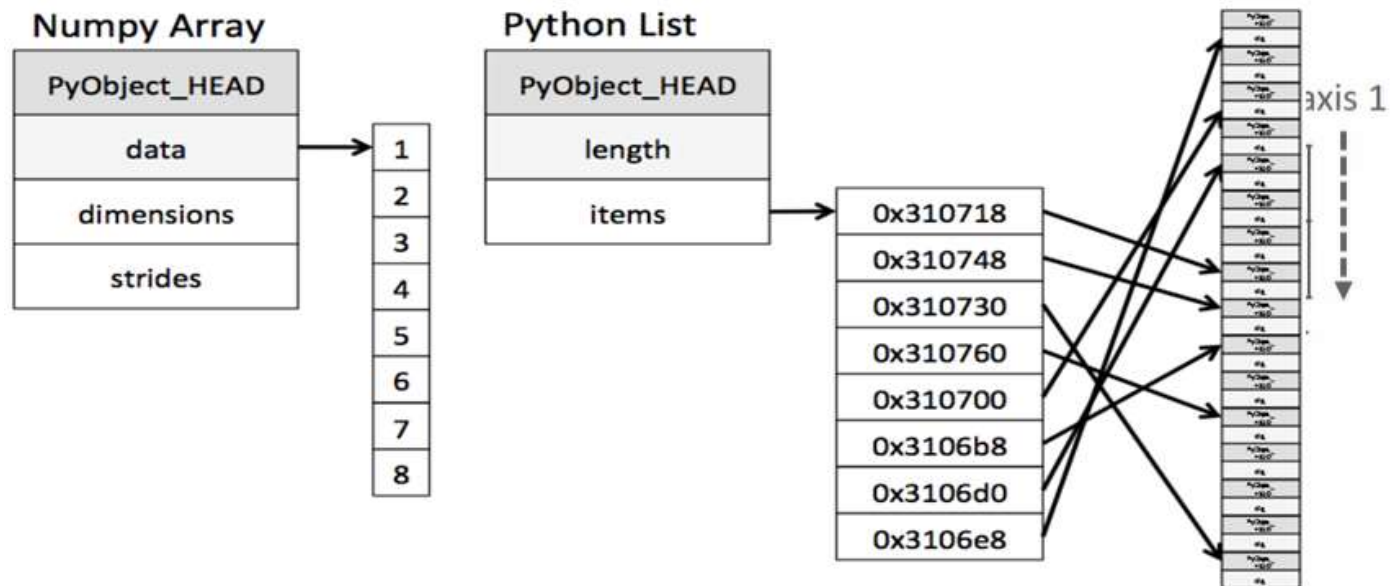
	배열	리스트
공간	고정된 공간	동적 공간
데이터	동일 데이터 타입	다양한 데이터 타입
특징	빠른 처리 & 적은 메모리	느린 처리 & 많은 메모리

NUMPY PACKAGE

8

◆ NumPy (Numerical Python)

❖ 배열 (ARRAY) vs 리스트 (LIST)



NUMPY PACKAGE

9

◆ NumPy (Numerical Python)

❖ dtype 데이터 타입

- 부호가 있다 → **signed int**
값의 범위 : 음수(-), 0, 양수(+)
- 부호가 없다 → **unsigned int**
값의 범위 : 0, 양수(+)

데이터 개수 동일
저장하는 데이터 범위 다름

접두사	설명		사용 예
'b'	불리언	bool	b (참 혹은 거짓)
'i'	정수	int	i8 (64비트)
'u'	부호 없는 정수	unsigned int	u8 (64비트)
'f'	부동소수점	float	f8 (64비트)
'c'	복소 부동소수점	complex	c16 (128비트)
'O'	객체	Object	0 (객체에 대한 포인터)
'S'	바이트 문자열	Str	S24 (24 글자)
'U'	유니코드 문자열	Unicode Uxxx	U24 (24 유니코드 글자)

NUMPY PACKAGE

10

◆ NumPy (Numerical Python)

❖ ndarray 클래스

- 다차원 배열(n-dimensional array)로 **Numpy의 기본**
- **하나의 자료형으로 만들어진 원소 보관 컨테이너**
- 행과 열 내의 **모든 원소는 동일한 형태**의 데이터
(all of the elements must be the same type)

NUMPY PACKAGE

11

◆ NumPy (Numerical Python)

❖ ndarray 클래스 속성

shape	배열구조 및 모양 (개수,) (행, 열) (깊이, 행, 열)
ndim	차원 수
dtype	데이터 타입
size	요소의 총 개수
itemsize	각 요소의 바이트 크기
data	실제 요소를 갖는 버퍼
strides	메모리 이동 칸수 (dim 간의 간격, 요소 간격)
order	메모리에 저장 순서 (C-style : row / F-style : column)

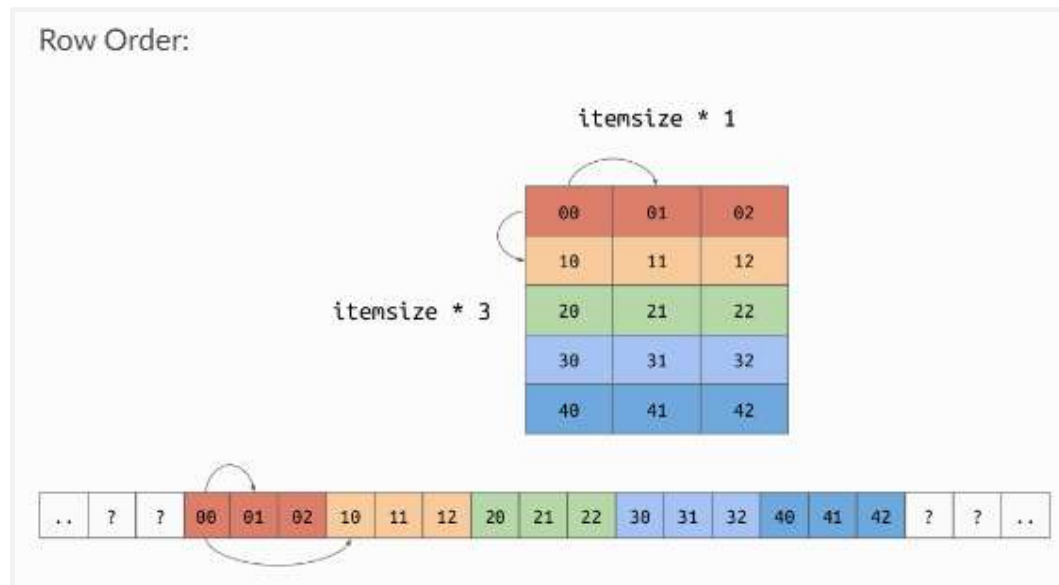
NUMPY PACKAGE

12

◆ NumPy (Numerical Python)

❖ ndarray 클래스 속성 – order

- 데이터 메모리 저장 방식
- C 언어 스타일
- 행우선 저장



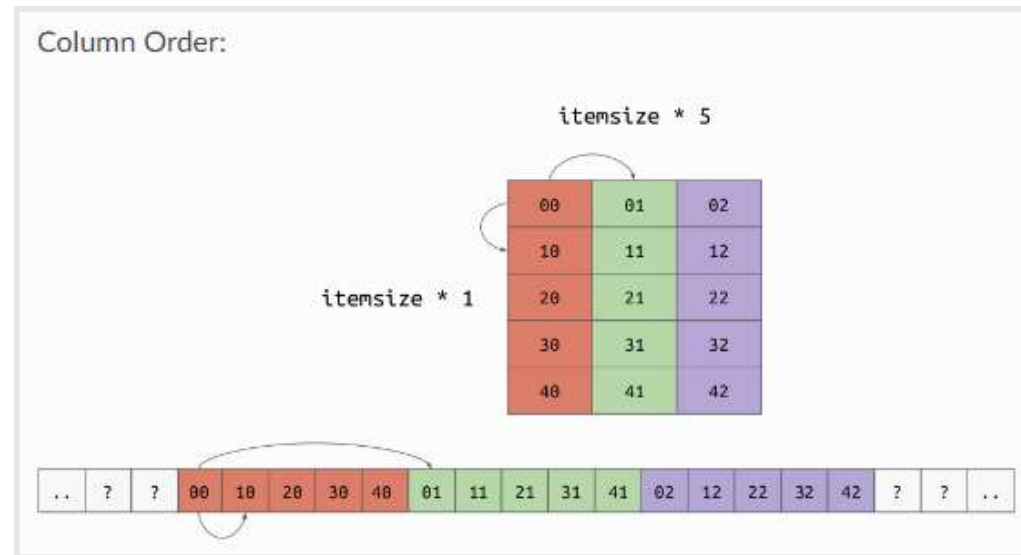
NUMPY PACKAGE

13

◆ NumPy (Numerical Python)

❖ ndarray 클래스 속성 – order

- 데이터 메모리 저장 방식
- Fortran 언어 스타일
- 열 우선 저장

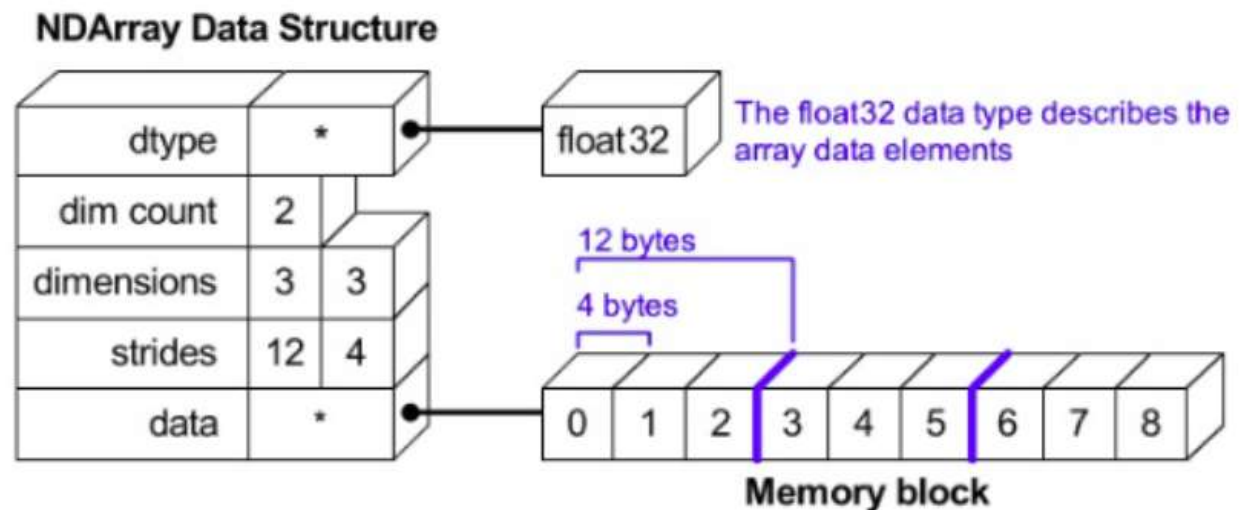


NUMPY PACKAGE

14

◆ NumPy (Numerical Python)

❖ ndarray 클래스 구조 : 메타데이터(데이터 정보) + 데이터 블록



NUMPY PACKAGE

15

◆ NumPy (Numerical Python)

❖ ndarray 객체 생성 함수들

- array(List 또는 Tuple)
- zeros(형태, 0), ones(형태, 1)
- full(형태, 값)
- eye(행, 열)
- zeros_like(), ones_like()
- empty()
- arrange()
- linspace(), logspace()

NUMPY PACKAGE

16

◆ NumPy (Numerical Python)

❖ ndarray 객체 - 생성

```
numpy.array(object, dtype=None, *, copy=True, order='K',  
            subok=False, ndmin=0, like=None) #
```

```
# 모듈 로딩 -----  
import numpy as np  
  
# ndarray 객체 생성 -----  
arr1=np.array( 5)  
arr2=np.array( [12, 3 ] )  
arr3=np.array( [ [1,2,3], [4,5,6,] ] , dtype=int )
```


NUMPY PACKAGE

17

◆ NumPy (Numerical Python)

❖ ndarray 객체 - 속성

```
def printAttribute(obj, strObj):  
    print(f'--- [{strObj} 속성] ---')  
    print(f'차원 : {obj.ndim}')  
    print(f'형태 : {obj.shape}')  
    print(f'타입 : {obj.dtype}')  
    print(f'크기/개수 : {obj.size}')  
    print(f'원소크기 : {obj.itemsize}')  
    print(f'데이터 : {obj.data}')  
    print(obj)
```

NUMPY PACKAGE

18

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 메서드

```
def runArrayMethod(obj):  
    print(f'obj.max()=> {obj.max()}')  
    print(f'obj.min()=> {obj.min()}')  
    print(f'obj.sum()=> {obj.sum()}')  
    print(f'obj.sum()=> {obj.mean()}')  
    if obj.ndim >=2:  
        print(f'obj.max(axis=0)=> {obj.max(axis=0)}')  
        print(f'obj.max(axis=1)=> {obj.max(axis=1)}')  
        print(f'obj.sum(axis=0)=> {obj.sum(axis=0)}')  
        print(f'obj.sum(axis=1)=> {obj.sum(axis=1)}')  
        print(f'obj.mean(axis=0)=> {obj.mean(axis=0)}')  
        print(f'obj.mean(axis=1)=> {obj.mean(axis=1)}')
```

NUMPY PACKAGE

19

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 생성

`numpy.ones(shape, dtype=None, order='C', *, like=None)`

```
arr1=np.ones(5)
arr2=np.ones((5,))
arr3=np.ones((2, 2))
arr4=np.ones((2, 2), dtype=int)
```

```
arr1, arr2, arr3, arr4
```

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

NUMPY PACKAGE

20

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 생성

```
numpy.zeros(shape, dtype=float, order='C', *, like=None) #
```

```
arr1=np.zeros(5)  
arr2=np.zeros((5,))  
arr3=np.zeros((2,2), dtype=int)
```

```
arr1, arr2, arr3
```

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

NUMPY PACKAGE

21

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 생성

`numpy.full(shape, fill_value, dtype=None, order='C', *, like=None)`

```
arr1=np.full(5, 12)
arr2=np.full((5,), 6)
arr3=np.full((2,2), 3, dtype=float)
```

```
arr1, arr2, arr3
```

```
(array([12, 12, 12, 12, 12]),
 array([6, 6, 6, 6, 6]),
 array([[3., 3.],
        [3., 3.]])
```

NUMPY PACKAGE

22

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 생성

`numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C', *, Like=None)`

```
arr1=np.eye(2)
arr2=np.eye(2,3)
arr3=np.eye(2, k=1)
```

```
arr1, arr2, arr3
```

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

NUMPY PACKAGE

23

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 생성

`numpy.empty(shape, dtype=float, order='C', *, like=None)`

```
arr1=np.empty(2)
arr2=np.empty((2,), dtype=np.float16)
arr3=np.empty((2,3), dtype=np.int64)
```

```
arr1, arr2, arr3
```

```
(array([1., 1.]),
 array([0., 0.], dtype=float16),
 array([[ 9198723453795908, -4066468847625715316, -3924605459494550075],
        [-5130507880119614756, -4991395621186251432, -5556119464938449528]],
      dtype=int64))
```

메모리 공간 할당
초기화 되지 않음
쓰레기값 존재

NUMPY PACKAGE

24

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 생성

`numpy.ones_like(a, dtype=None, order='K', subok=True, shape=None)`

```
arr1=np.array([[11,11],[22,22],[33,33]])  
arr2=np.ones_like(arr1)
```

```
arr1, arr2  
  
(array([[11, 11],  
        [22, 22],  
        [33, 33]]),  
 array([[1, 1],  
        [1, 1],  
        [1, 1]]))
```


NUMPY PACKAGE

25

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 생성

`numpy.arange([start,]stop, [step,]dtype=None, *, like=None)`

```
arr1=np.arange(10)
arr2=np.arange(1, 10)
arr3=np.arange(1, 10, 3, dtype=np.float32)
```

```
arr1, arr2, arr3
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([1., 4., 7.], dtype=float32))
```

NUMPY PACKAGE

26

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 생성

`numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)`

start	: 시작 값
stop	: 마지막 값
	endpoint True면 포함, False면 미포함
num	: 배열 멤버의 개수 / 기본 값 50
endpoint	: stop 값 포함 여부 설정
retstep	: 배열과 스텝값 반환 여부 설정
dtype	: 데이터 타입

```
arr1=np.linspace(2, 3, num=3)
arr2=np.linspace(1,5, num=5)
arr3=np.linspace(1,5, num=5, endpoint=False)
arr4, step=np.linspace(1,5, num=5, retstep=True)
```

```
arr1, arr2, arr3, arr4, step
```

```
(array([2. , 2.5, 3. ]),
 array([1., 2., 3., 4., 5.]),
 array([1. , 1.8, 2.6, 3.4, 4.2]),
 array([1., 2., 3., 4., 5.]),
 1.0)
```

NUMPY PACKAGE

27

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 생성

`numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)`

start	: 시작 값
stop	: 마지막 값 endpoint True면 포함, False면 미포함
num	: 배열 멤버의 개수 / 기본 값 50
endpoint	: stop 값 포함 여부 설정
base	: Log 베이스 값 / 기본값 10
dtype	: 데이터 타입

```
np.set_printoptions(precision=6, suppress=True)
```

```
arr1=np.logspace(2, 3, num=2)  
arr2=np.logspace(1,5, num=5, base=2)  
arr3=np.logspace(1,3, num=3, endpoint=False)  
arr4=np.logspace(1,3, num=3, endpoint=True)
```

```
arr1,arr2,arr3, arr4
```

```
(array([ 100., 1000.]),  
 array([ 2.,  4.,  8., 16., 32.]),  
 array([ 10.,      46.415888, 215.443469]),  
 array([ 10.,  100., 1000.]))
```

NUMPY PACKAGE

28

◆ NumPy (Numerical Python)

❖ ndarray 객체 – shape 변환

➤ 1차원 ----> 1 ~ N 차원

- ndarray.**reshape**(행, 열) : view ndarray

➤ N차원 ----> 1차원

- ndarray.**flatten**() : copy ndarray
- ndarray.**ravel**() : view ndarray

NUMPY PACKAGE

29

◆ NumPy (Numerical Python)

❖ ndarray 객체 – shape 변환

```
import numpy as np
```

```
# ndarray 객체 생성 -----
```

```
arr01=np.array( [ [1,2,3,4,5], [6,7, 8, 9, 10] ])
```

```
print('arr01.shape =', arr01.shape)
```

```
print(arr01)
```

```
arr01=arr01.T
```

```
print('arr01.shape =', arr01.shape)
```

```
print(arr01)
```

NUMPY PACKAGE

30

◆ NumPy (Numerical Python)

❖ ndarray 객체 – shape & size(원소 수) 변환

➤ `numpy.resize(array, new_shape)`

: 복사 본 반환

➤ `ndarray.resize(new_shape, refcheck= True)`

: 원본 ndarray에 변경 적용

NUMPY PACKAGE

31

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 요소 인덱싱(Indexing)

- 객체변수명[인덱스] - Zero Base Indexing

7	14	21	28	35	42	49
[0]	[1]	[2]	[3]	[4]	[5]	[6]
[-7]					[-2]	[-1]

NUMPY PACKAGE

32

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 요소 인덱싱(Indexing)

▪ 1차원

7	14	21	28	35	42	49
[0]	[1]	[2]	[3]	[4]	[5]	[6]
[-7]	[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

➔ `arr[-1] == arr[6]` , `arr[3] == arr[-4]`

NUMPY PACKAGE

33

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 요소 인덱싱(Indexing)

- 2차원 : array[행인덱스][요소인덱스]

→ `arr[-1][-1] == arr[1][2]`

→ `arr[0][-1] == arr[-2][2]`

row 0 [0]

row 1 [1]

11 [0][0]	22 [0][1]	33 [0][2]
44 [1][0]	55 [1][1]	66 [1][2]

[0]

[1]

[2]

[-3]

[-2]

[-1]

NUMPY PACKAGE

34

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 요소 범위 지정 슬라이싱(Slicing)

- 2차원 : array[시작인덱스:끝인덱스]

→ 범위 : 시작인덱스 <= 요소 < 끝인덱스

→ 결과 : View

→ arr[0][1:], arr[:-1][:]

[0] [-3]

[1] [-2]

[2] [-1]

11	22	33	44
55	66	77	88
99	100	110	120

[0] [1] [2] [3]

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

NUMPY PACKAGE

35

◆ NumPy (Numerical Python)

❖ ndarray 객체 – 요소 범위 지정 슬라이싱(Slicing)

[0]	[-3]	11	22	33	44
[1]	[-2]	55	66	77	88
[2]	[-1]	99	100	110	120
		[0]	[1]	[2]	[3]
		[-4]	[-3]	[-2]	[-1]

```
arr[0][:]=-1
```

```
arr
```

```
array([[ -1,  -1,  -1,  -1],  
       [ 55,  66,  77,  88],  
       [ 99, 100, 110, 120]])
```

```
arr2=arr[0][:].copy()
```

```
arr2[:] = -1
```

```
arr2
```

NUMPY PACKAGE

36

◆ NumPy (Numerical Python)

❖ 팬시 인덱싱 - Fancy Indexing

- 단일 스칼라 대신 **배열로 배열을 인덱싱할 수 있는 기능**
- 반드시 **정수 타입만 가능**, view가 아니라 **copy로 생성**

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

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

Fancy Indexing

arr[**[1, 2]**]

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

NUMPY PACKAGE

37

◆ NumPy (Numerical Python)

❖ 불린 인덱싱 - Boolean Indexing

- 불린 연산(>, >=, <, <=, ==, !=) 이용한 불린 벡터를 이용한 인덱싱
- 원하는 행 또는 열의 값만 추출, 마스크(Mask) 또는 필터(Filter)라고도 함

비교 연산자	의미	
<, <=	$x < y$, $x \leq y$	x가 y보다 작으면 True x가 y보다 작거나 같으면 True
>, >=	$x > y$, $x \geq y$	x가 y보다 크면 True x가 y보다 크거나 같으면 True
==, !=	$x == y$, $x != y$	X와 y가 같으면 True X와 y가 같지 않으면 True

논리 연산자	기호	의미
and	&	x and y
or		x or y
not	~	not x

NUMPY PACKAGE

38

◆ NumPy (Numerical Python)

❖ 불린 인덱싱 - Boolean Indexing

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

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

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

```
~(names == 'Bob')
```

```
( names == 'Bob' ) & ( names == 'Joe' ) ← and 연산자
```

```
( names == 'Bob' ) | ( names == 'Joe' ) ← or 연산자
```

NUMPY PACKAGE

39

◆ NumPy (Numerical Python)

❖ 불린 인덱싱 - Boolean Indexing

```
array1 = np.array(['A','B','C','A','E'])  
bool_array1 = array1 == 'A'  
print('bool_array1 =>', bool_array1)  
  
bool_array1 = array1 >= 'B'  
print('bool_array1 =>', bool_array1)  
  
array2 = np.array([1,2,3,4,5])  
bool_array2 = array2 >= 3  
print('bool_array2 =>', bool_array2)
```

NUMPY PACKAGE

40

◆ NumPy (Numerical Python)

❖ 불린 인덱싱 - Boolean Indexing

```
# ndarray 객체 생성 -----  
data=np.array( [[0,1,2],[3,4,5],[6,7,8],[9,10,11]] )  
simpleInfo('data',data)  
  
# 3,4,5 데이터 추출을 위한 마스크 생성 및 적용  
mask1=(data>2)&(data<6)  
print('mask1 =>\n ', mask1)  
print('data[mask1] =>\n ', data[mask1])
```


NUMPY PACKAGE

41

◆ NumPy (Numerical Python)

❖ 불린 인덱싱 - Boolean Indexing

```
# 3,4,5 데이터 제외한 나머지 데이터 추출 위한 마스크 생성 및 적용  
mask2=~((data>2)&(data<6))  
  
print('mask2 =>\n ', mask2)  
  
print('data[mask2] =>\n ', data[mask2])
```

NUMPY PACKAGE

42

◆ NumPy (Numerical Python)

❖ 행, 열, 요소 삭제

➤ 요소 삭제 → 반환 1D

- `numpy.delete(ndarray, index)`
- `numpy.delete(ndarray, [index, ..., index])`

➤ 행 삭제

- `numpy.delete(ndarray, index, axis=0)`

➤ 열 삭제

- `numpy.delete(ndarray, index, axis=1)`

NUMPY PACKAGE

43

◆ NumPy (Numerical Python)

❖ 행, 열, 요소 추가 – append()

**** 추가되는 새로운 ndarray는 행 크기/열 크기 동일해야 함**

➤ 요소 추가

- `numpy.append(ndarray, value)`

➤ 행 추가

- `numpy.append(ndarray, new_array , axis=0)`

➤ 열 추가

- `numpy.append (ndarray, new_array , axis=1)`

NUMPY PACKAGE

44

◆ NumPy (Numerical Python)

❖ 행, 열, 요소 삽입 – insert()

**** 추가되는 새로운 ndarray는 행 크기/열 크기 동일해야 함**

➤ 요소 추가

- `numpy.insert(ndarray, index, value)`

➤ 행 추가

- `numpy.insert(ndarray, index, new_array , axis=0)`

➤ 열 추가

- `numpy.insert (ndarray, index, new_array , axis=1)`

NUMPY PACKAGE

45

◆ NumPy (Numerical Python)

❖ 형태 변경 전치(Transpose)

- 속 성 : `ndarray.T`
- 메서드 : `ndarray.transpose(축0, 축1, 축2)`
- 행 < === > 열 바꾸는 작업
- 2차원 이상 배열인 경우 가능

1 x 3

7	14	21
---	----	----

전치

7	3 X 1
14	
21	

NUMPY PACKAGE

46

◆ NumPy (Numerical Python)

❖ 형태 변경 전치(Transpose)

```
import numpy as np

nparray01=np.array( [ [1,2,3,4,5], [6,7, 8, 9, 10] ] )

print('nparray01.shape =', nparray01.shape)
print(nparray01)

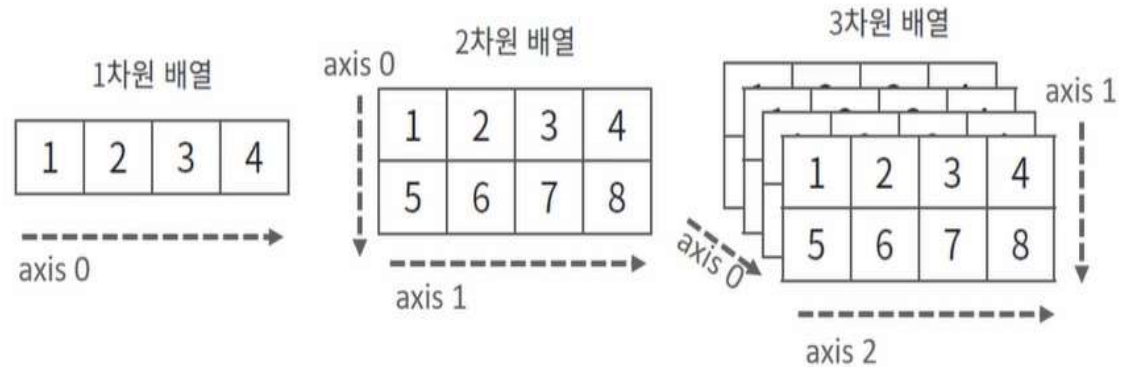
nparray01=nparray01.T
print('nparray01.shape =', nparray01.shape)
print(nparray01)
```

NUMPY PACKAGE

47

◆ NumPy (Numerical Python)

❖ 축(axis)



shape(수,)

shape(행, 열)

axis = 0 행 row

axis = 1 열 column

shape(차원, 행, 열)

axis = 0 차원

axis = 1 행 row

axis = 2 열 column

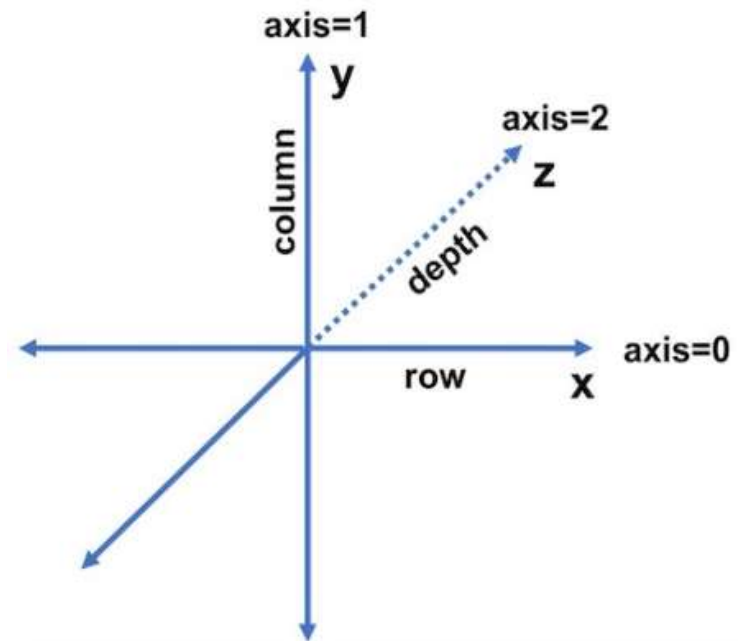
NUMPY PACKAGE

48

◆ NumPy (Numerical Python)

❖ 축(axis)

다차원 ndarray



NUMPY PACKAGE

49

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

	<code>concatenate((array, array), axis)</code>	배열 합치기
	<code>hstack()</code>	수평 방향 연결
	<code>vstack()</code>	수직 방향 연결
	<code>dstack()</code>	깊이 방향 연결
	<code>stack()</code>	사용자 지정차원으로 연결
특별한 메서드 인덱서 (Indexer)	<code>r_[]</code>	<code>hstack()</code> 명령과 비슷, 좌우로 연결
	<code>c_[]</code>	배열의 차원 증가시킨 후 좌우 연결
	<code>tile()</code>	동일한 배열 반복하여 연결

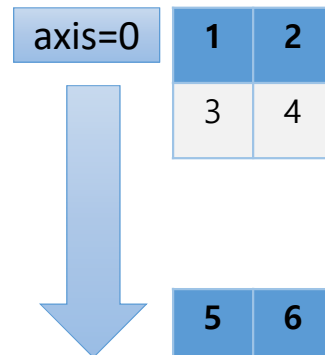
NUMPY PACKAGE

50

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

- `numpy.concatenate((arr1, arr2), axis)` : 축 기준 결합
- 조건 : 기준 축 제외한 나머지의 shape 동일해야 함!



행방향 연결

a.shape = (2, 2)

b.shape = (1, 2)

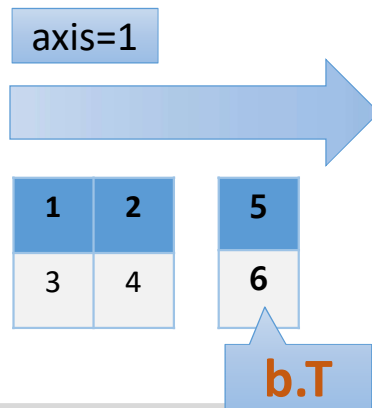
NUMPY PACKAGE

51

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

- `numpy.concatenate((arr1, arr2), axis)` : 축 기준 결합
- 조건 : 기준 축 제외한 나머지의 shape 동일해야 함!



열방향 연결

`a.shape` = (2, 2)

`b.T.shape` = (2, 1)

NUMPY PACKAGE

52

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

```
import numpy as np

# 임의의 데이터
data1=np.random.randint(1, 20, (5,5))
data1.shape, data1.ndim, data1
```

NUMPY PACKAGE

53

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

[기본] 행 방향

```
data3=np.concatenate((data1, data2))
```

[옵션] 열 방향

```
data4=np.concatenate((data1, data2), axis=1)
```

[옵션] 1D

```
data5=np.concatenate((data1, data2), axis=None)
```

NUMPY PACKAGE

54

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

- **numpy.stack()** : 수평 방향 연결
- 조건 : 행(row)/열(column) 개수 동일해야 함!

```
data2=np.stack((data1, data2))
```

```
data2.shape, data2.ndim
```

```
((2, 5, 5), 3)
```

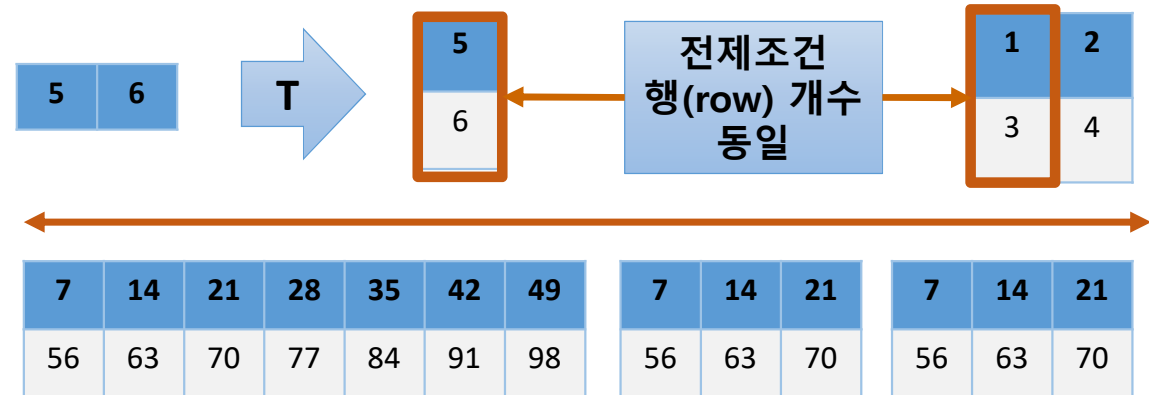
NUMPY PACKAGE

55

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

- `numpy.hstack()` : 수평 방향 연결
- 조건 : 행(row) 개수 동일해야 함!



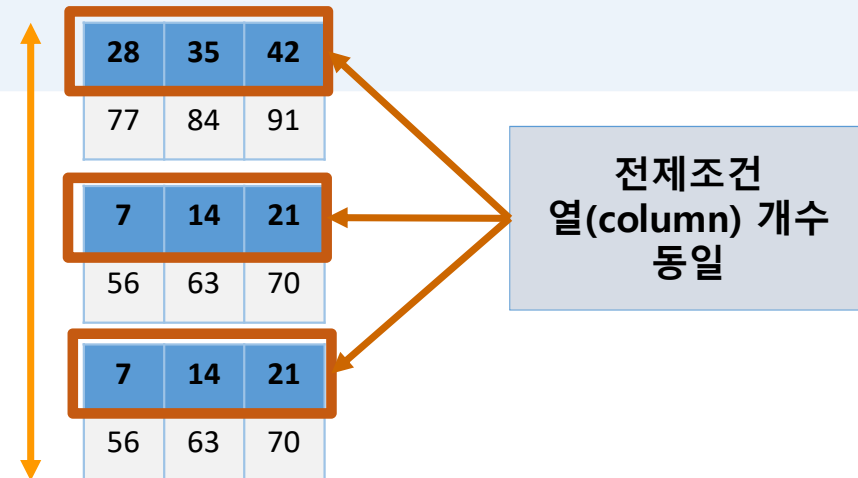
NUMPY PACKAGE

56

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

- **numpy.vstack()** : 수직 방향 연결
- 조건 : 열(column) 개수 동일해야 함!



NUMPY PACKAGE

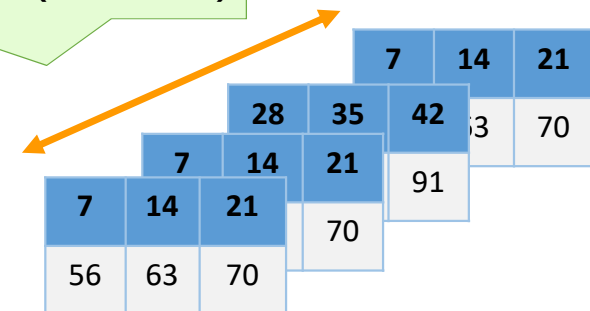
57

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

- **numpy.dstack()** : 깊이/채널/차원 방향연결
- 조건 : 행(row) / 열(column) 개수 동일해야 함!

전제조건
깊이(dept) / 채널(channel) / 차원(dimension)



58

98

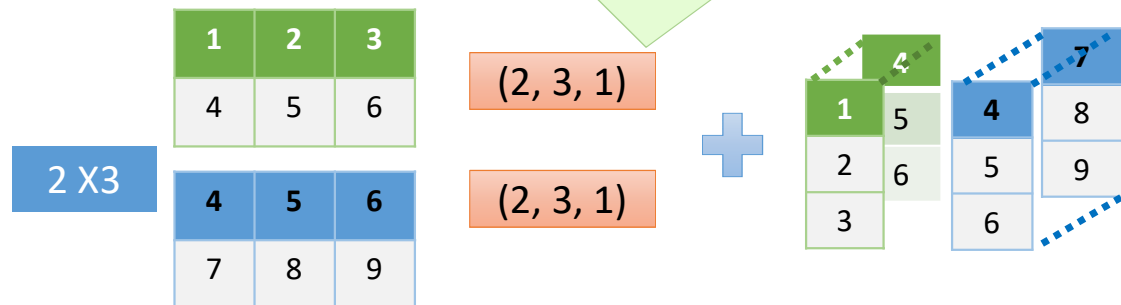
NUMPY PACKAGE

59

◆ NumPy (Numerical Python)

❖ 합치기/연결 – 2개 이상 배열

- **numpy.dstack()** : 깊이/채널/차원 방향연결
- 조건 : 행(row) / 열(column) 개수 동일해야 함!



NUMPY PACKAGE

60

◆ NumPy (Numerical Python)

❖ 통계 관련 메서드

mean()	평균
std()	표준편차
var()	분산
sum()	합계
min()	최소값
max()	최대값
median()	중앙값

축(Axis) 주의 필요

행(row)기준 계산 : axis=1

열(column)기준 계산 : axis=0

차원 축소 연산

(Dimension Reduction)

NUMPY PACKAGE

61

◆ NumPy (Numerical Python)

❖ 통계 관련 메서드

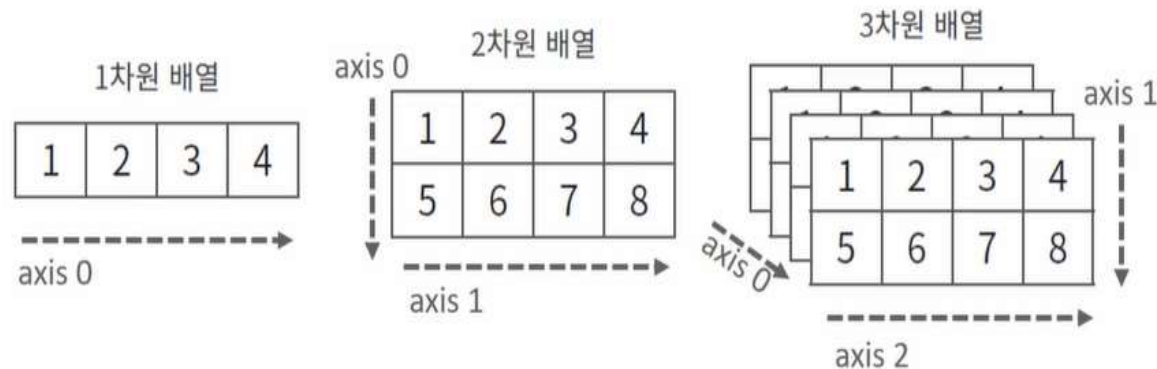
<code>argmin()</code> , <code>argmax()</code>	전체 요소의 최솟값, 최댓값의 위치 인덱스 반환
<code>cumsum()</code> , <code>cumprod()</code>	첫 번째 성분부터 누적합/누적곱 계산
<code>np.sort(ndarray, axis=0)</code>	각 열을 오름차순으로 정렬
<code>np.sort(ndarray, axis=1)</code>	각 행을 오름차순으로 정렬
<code>np.argsort(ndarray)</code>	인덱스를 오름차순으로 정렬 후 인덱스 반환
<code>np.argsort(-ndarray)</code>	인덱스를 내림차순으로 정렬 후 인덱스 반환
<code>np.unique(ndarray)</code>	중복 성분 제거한 array 반환

NUMPY PACKAGE

62

◆ NumPy (Numerical Python)

❖ 통계 관련 함수와 축



1 X 3	1+2+3	=> sum	axis=0	2 X 1	1+4	=> sum	axis=0
1 2 3	(1+2+3)/3	=> mean		1	(1+4)/2	=> mean	
	1,2,3 비교	=> max		4	1,4비교	=> max	

NUMPY PACKAGE

63

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

- 벡터의 **같은 인덱스에 위치한 원소(Element-wise)**들끼리 연산 수행 의미
- 명시적 반복문 사용하지 않고 배열 모든 원소에 대해 반복연산
- 적용 기준
 - 두 배열의 shape가 정확히 같은 경우 가능
 - 하나의 배열의 차원이 1인 경우 가능

NUMPY PACKAGE

64

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

열(column)벡터 M x 1 행렬

같은 인덱스에 위치한 원소(Element-wise)들끼리

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ 10000 \end{bmatrix} + \begin{bmatrix} 10001 \\ 10002 \\ 10003 \\ \vdots \\ 20000 \end{bmatrix} = \begin{bmatrix} 1 + 10001 \\ 2 + 10002 \\ 3 + 10003 \\ \vdots \\ 10000 + 20000 \end{bmatrix} = \begin{bmatrix} 10002 \\ 10004 \\ 10006 \\ \vdots \\ 30000 \end{bmatrix}$$

NUMPY PACKAGE

65

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

- data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- 데이터를 모두 2배

PYTHON

```
answer = [ ]  
for di in data:  
    answer.append(2 * di)  
  
answer= [ 2*di for in data]
```

NUMPY

```
x = np.array(data)  
  
2 * x
```

NUMPY PACKAGE

66

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

Broadcasting : 서로 다른 shape 가진 array의 산술 연산이 가능하도록 하는 것

SHAPE
맞 추 기

SIZE가 작은 쪽의 배열을 큰 쪽의 배열 크기로 확장시켜 연산
자동 reshape 및 반복된 값으로 자동 할당한 후 연산

[조건] 하나의 배열이 1차원인 경우

[조건] 배열의 열(column) 개수가 동일한 경우

NUMPY PACKAGE

67

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

Broadcasting : 서로 다른 shape 가진 array의 산술 연산이 가능하도록 하는 것

$$x = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad x + 1 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 1 = ?$$

자동확장

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 1 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

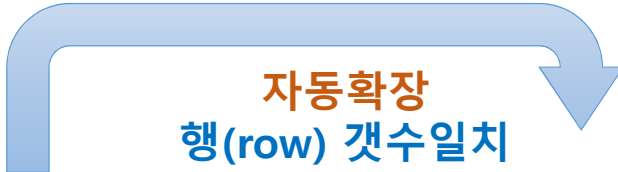
NUMPY PACKAGE

68

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

Broadcasting : 서로 다른 shape 가진 array의 산술 연산이 가능하도록 하는 것


$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{bmatrix}$$

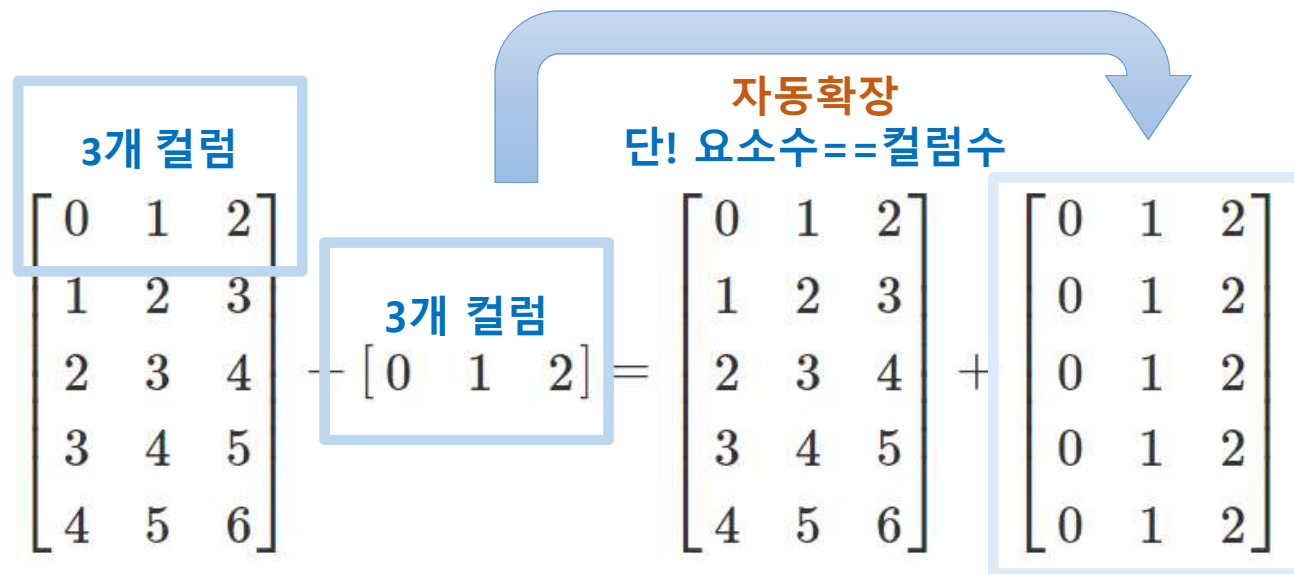
NUMPY PACKAGE

69

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

Broadcasting : 서로 다른 shape 가진 array의 산술 연산이 가능하도록 하는 것



NUMPY PACKAGE

70

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

행(row)벡터 $1 \times M$ 행렬

열(column)벡터 $M \times 1$ 행렬

기본

$$\mathbf{x} = [x_1 \quad x_2 \quad \dots \quad x_m]$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

NUMPY PACKAGE

71

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

행(row)벡터 1 x M 행렬

전치

열(column)벡터 M x 1 행렬

$$\begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

NUMPY PACKAGE

72

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

행(row)벡터 1 x M 행렬

전치

열(column)벡터 M x 1 행렬

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}^T = [x_1 \ x_2 \ \dots \ x_m]$$

NUMPY PACKAGE

73

◆ NumPy (Numerical Python)

❖ 기본 연산 – 벡터화 연산

행(row)벡터 1 x M 행렬

전치

열(column)벡터 M x 1 행렬

```
x = np.arange(5)
```

```
xc=x.reshape(-1, 1)
```

←컬럼 1개, 행은 알아서

```
print(f' x=> size : { x.size }, shape : {x.shape}, x : { x }')
```

```
print(f' xc=> size : { xc.size }, shape : {xc.shape}, xc : { xc }')
```

NUMPY PACKAGE

74

◆ NumPy (Numerical Python)

❖ 행렬(Matrix) 곱 / 내적 (Dot/Inner Product)

$$\begin{array}{l} \text{1열} \quad \text{2열} \\ \begin{array}{l} \text{1행} \\ \text{2행} \end{array} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 5 \times 1 + 6 \times 3 & 5 \times 2 + 6 \times 4 \\ 7 \times 1 + 8 \times 3 & 7 \times 2 + 8 \times 4 \end{pmatrix} \end{array}$$

$$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 5 \times 1 + 6 \times 3 & 5 \times 2 + 6 \times 4 \\ 7 \times 1 + 8 \times 3 & 7 \times 2 + 8 \times 4 \end{pmatrix}$$

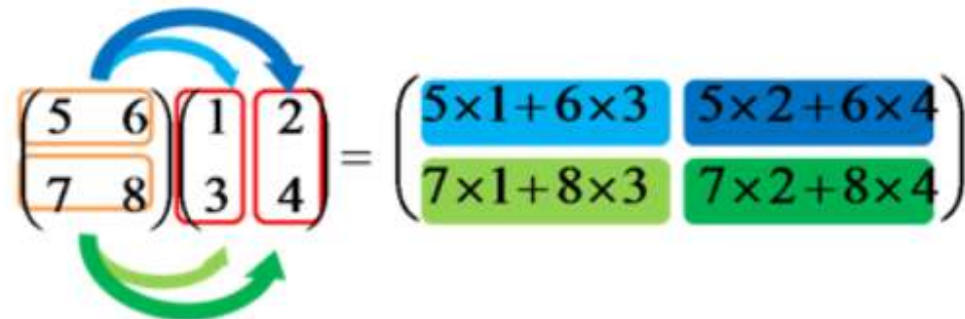
NUMPY PACKAGE

75

◆ NumPy (Numerical Python)

❖ 행렬(Matrix) 곱 / 내적 (Dot/Inner Product)

$$\begin{array}{c} \text{1열} \quad \text{2열} \\ \begin{array}{c} \text{1행} \\ \text{2행} \end{array} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 5 \times 1 + 6 \times 3 & 5 \times 2 + 6 \times 4 \\ 7 \times 1 + 8 \times 3 & 7 \times 2 + 8 \times 4 \end{pmatrix} \end{array}$$

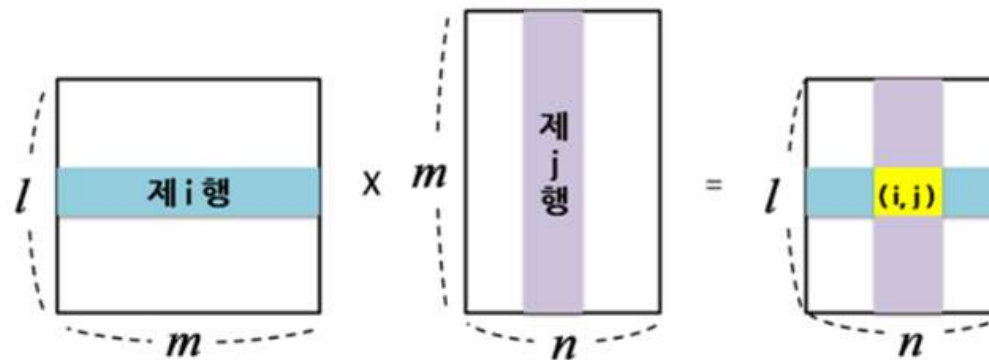

$$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 5 \times 1 + 6 \times 3 & 5 \times 2 + 6 \times 4 \\ 7 \times 1 + 8 \times 3 & 7 \times 2 + 8 \times 4 \end{pmatrix}$$

NUMPY PACKAGE

76

◆ NumPy (Numerical Python)

❖ 행렬(Matrix) 곱 / 내적 (Dot/Inner Product)



A행렬
 $l \times m$

x

B행렬
 $m \times n$

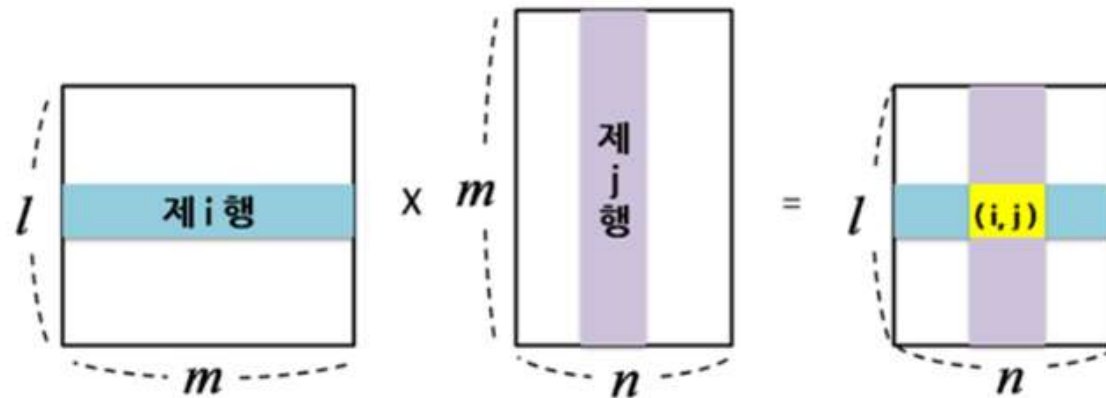
=

AB행렬
 $l \times n$

NUMPY PACKAGE

77

◆ 행렬(Matrix) 곱 / 내적



A행렬

\times

B행렬

$=$

AB행렬

$l \times m$

$m \times n$

$l \times n$

NUMPY PACKAGE

78

◆ 행렬(Matrix) 곱 / 내적

함수	기능
<code>np.dot(arr1, arr2)</code>	1D 경우 : 내적 2D 경우 : 행렬 곱 (<code>np.matmul</code> 사용을 권장) nD 경우 : 첫번째 행렬 마지막 axis, 두번째 행렬 뒤에서 2번째 axis와 내적
<code>np.matmul(arr1, arr2)</code>	2D 경우 : 행렬 곱 스칼라 값으로 배열의 곱셈을 수행
<code>arr1@arr2</code>	<code>np.matmul(arr1, arr2)</code> 함수 호출