

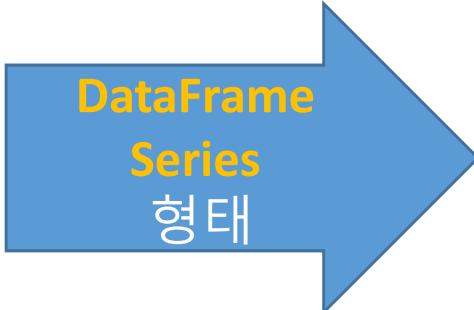
# 데이터 분석을 위한 필수 패키지 NUMPY

2022.07



**NUMPY**





# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

>> C언어로 구현된 파이썬 라이브러리

>> **고성능의 수치계산**을 위해 제작된 라이브러리

>> **loop 프로그래밍 없이** 전체 배열에 대해 표준 **수학 함수** **빠른 속도 수행**

>> 파이썬에 **배열(Array)** 기능 제공 라이브러리

>> **대량의 데이터**에 대한 고급 수학적 및 기타 유형 작업

>> **설치 : pip install numpy / conda install numpy**



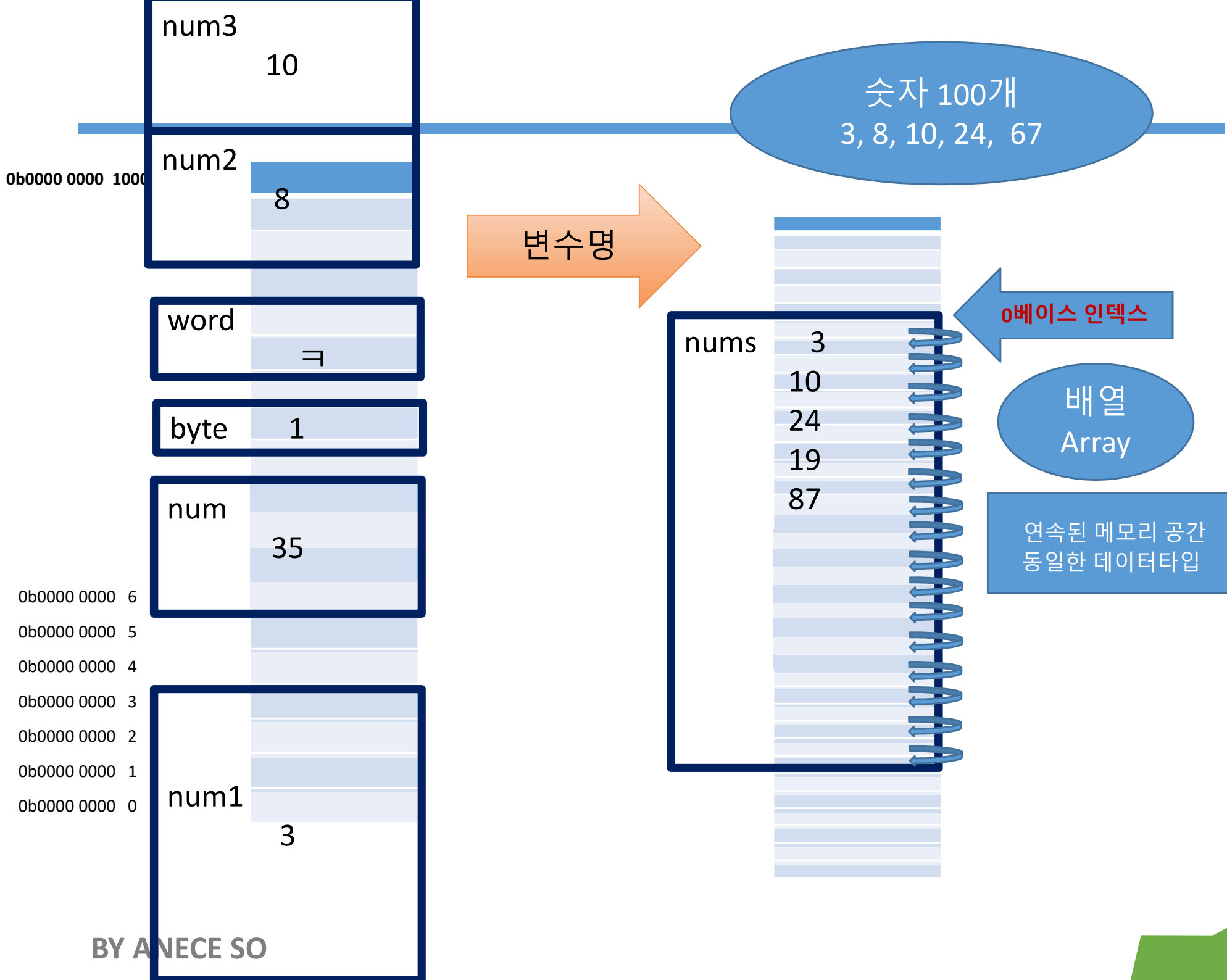
# 필수 라이브러리 NUMPY

---

## ◆ NumPy (Numerical Python)

### 배열(ARRAY)

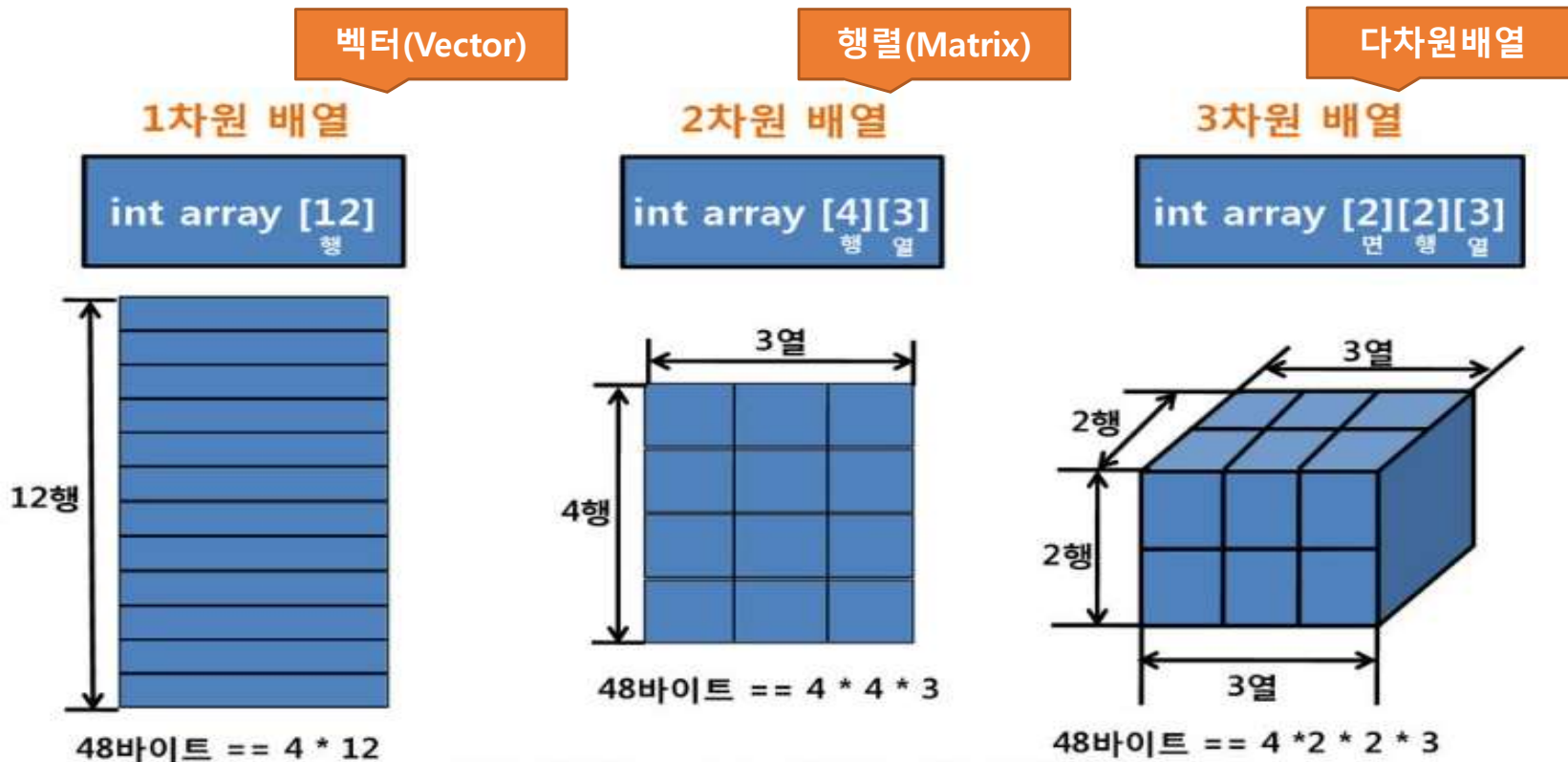
- 순차적으로 데이터를 저장하는 자료형
- 서로 연관있는 데이터를 하나의 변수명으로 저장
- 같은 타입의 변수들로 이루어진 유한 집합으로 정의
- 원소 개수 변경 불가
- 구성하는 각각의 값을 배열 요소/원소(element)
- 위치를 가리키는 숫자는 인덱스(index)
- 같은 종류의 데이터를 많이 다뤄야 하는 경우에 사용



# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### 배열(ARRAY) 종류



# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

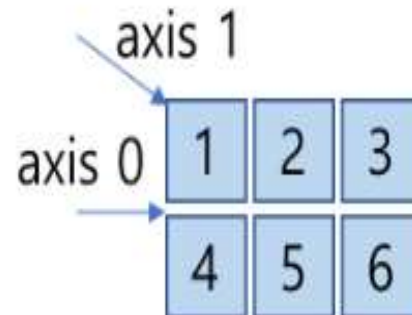
### 배열 (ARRAY) 종류

벡터 (Vector)



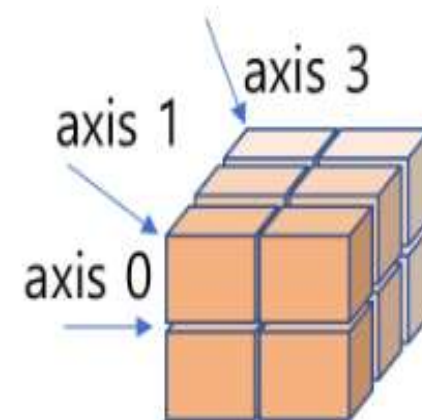
1D array

행렬 (Matrix)



2D array

다차원배열



3D array



# 필수 라이브러리 NUMPY

---

## ◆ NumPy (Numerical Python)

배열(ARRAY) ← NumPy 기본 단위

- 고정된 공간에 동일한 데이터 타입 저장

리스트(LIST) ← Python 데이터 타입

- 동적 공간에 다양한 데이터 타입 저장

많은 숫자 데이터를 하나의 변수에 넣고 관리 할 때

- 리스트 : 속도가 느리고 메모리를 많이 차지
- 배열 : 적은 메모리로 많은 데이터를 빠르게 처리

# 필수 라이브러리 NUMPY

---

## ◆ 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 형태의 데이터(all of the elements must be the same type)

# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### dtype 살펴보기

음수(-) 0 양수(+) ← 부호가 있다 **signed int**  
0, 양수(+) ← 부호가 없다 **unsigned int**

dtype 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)

# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### 속성

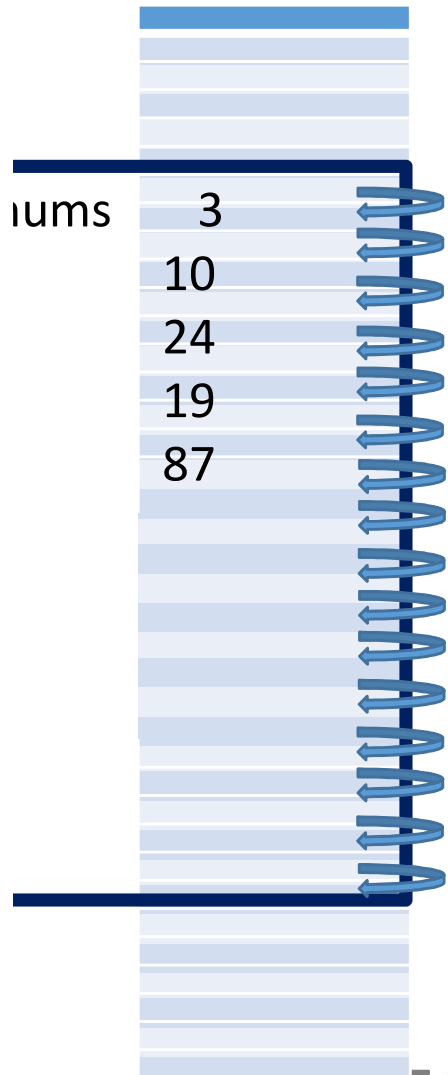
속 성	설명
shape	배열구조 및 모양 (개수, ) (행, 열) (깊이, 행, 열)
ndim	차원 수
dtype	데이터 타입
size	요소의 총 개수
itemsize	각 요소의 바이트 크기
data	실제 요소를 갖는 버퍼

---

◆ `arr=np.ones(5, dtype='i')`

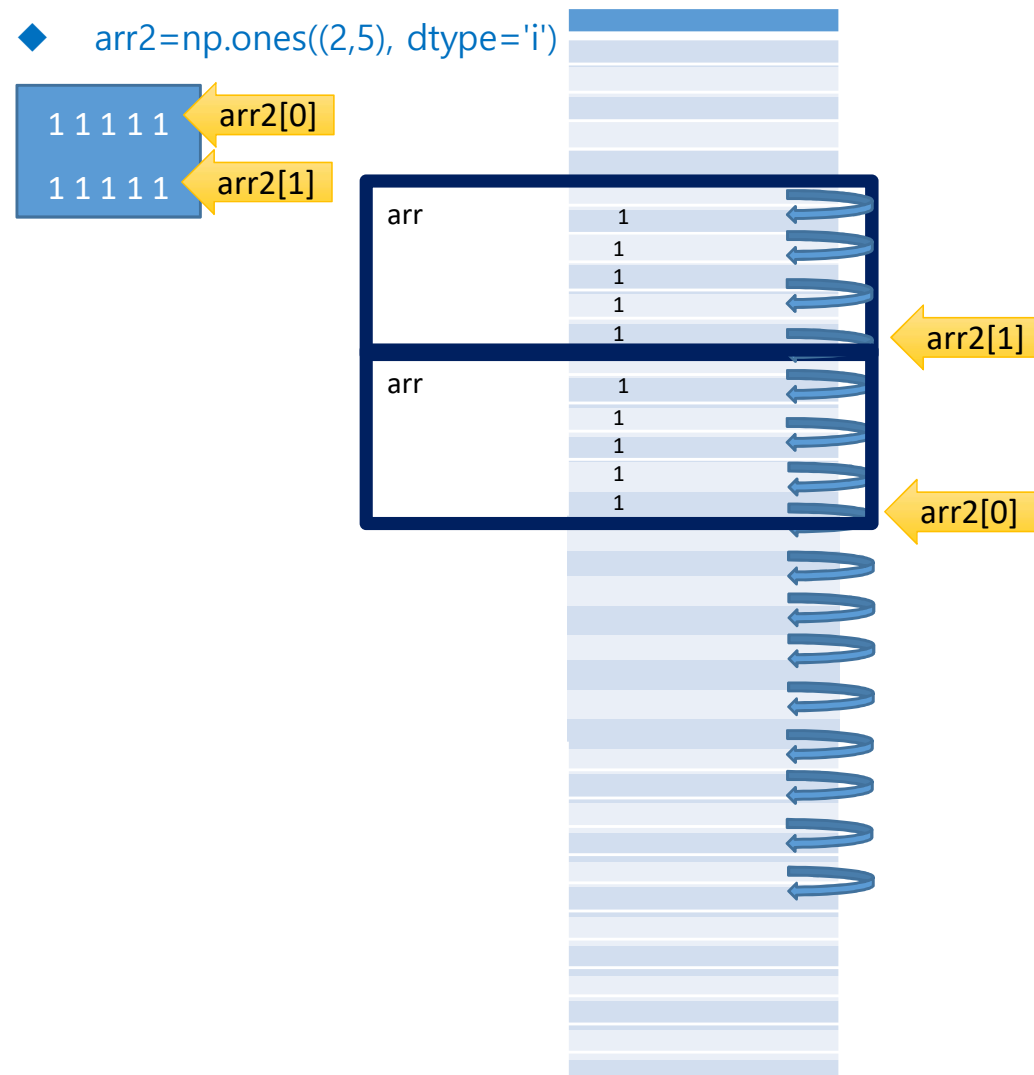
=> 정수 5개를 연속된 메모리 공간에 저장

=> 기존에 메모리에 있던 값 지우고 0으로 설정



◆ `arr=np.ones(5, dtype='i')`

◆ `arr2=np.ones((2,5), dtype='i')`



# 필수 라이브러리 NUMPY

---

## ◆ NumPy (Numerical Python)

### ❖ ndarray 객체 생성 명령

- `array()`
- `zeros(배열 형태, 0)`, `ones(배열 형태, 1)`
- `full(배열 형태, 값)`
- `zeros_like()`, `ones_like()`
- `empty()`
- `arange()`
- `linspace()`, `logspace()`

# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### ❖ 생성 array()

```
import numpy as np
```

```
# numpy ndarray 객체 생성 -----
```

```
nparray01 = np.array([1,2,3,4,5])
```

```
print('type(nparray01) =', type(nparray01))
```

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

```
print('nparray01.size =', nparray01.size)
```

```
print(nparray01)
```

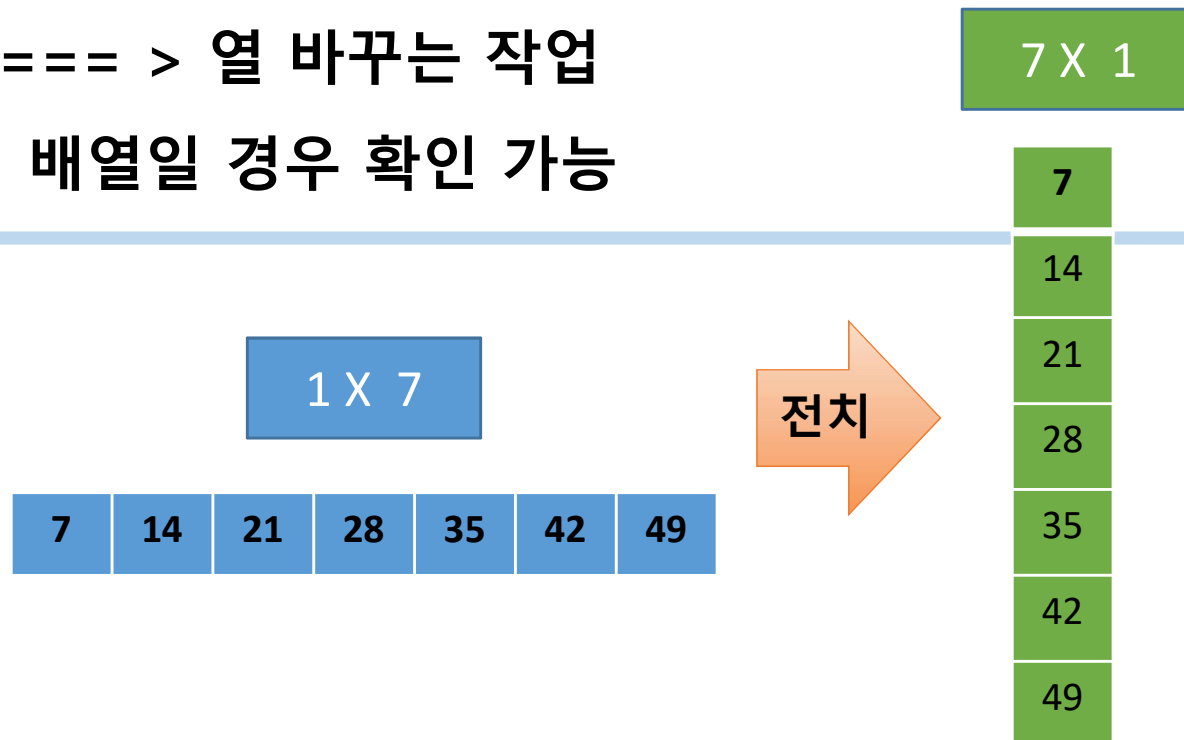


# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### ❖ 전치(Transpose) 연산

- 속성 : `ndarray.T`
- 행 < === > 열 바꾸는 작업
- 2차원 배열일 경우 확인 가능



# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

❖ 전치(Transpose) => 속성.T

```
import numpy as np
```

```
# numpy ndarray 객체 생성 -----
```

```
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)
```

# 필수 라이브러리 NUMPY

---

## ◆ NumPy (Numerical Python)

### ❖ 크기 변경

- `reshape( 행, 열)`
- 다차원 배열 ==> 1차원
  - `flatten ()`
  - `ravel()`

# 49

7	14	21	28	35	42	49	56	63	70	77	84	91	98
---	----	----	----	----	----	----	----	----	----	----	----	----	----

7	14	21	28	35	42	49
56	63	70	77	84	91	98

7	14
21	
35	
49	
63	
77	
84	
91	

1 X 7

7	14	21	28	35	42	49
---	----	----	----	----	----	----



7 X 1

7
14
21
28
35
42
49

	56	63	70	77	84	91	98
7	14	21	28	35	42	49	

# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

❖ Shape 변경 => reshape()

```
import numpy as np
```

```
# numpy ndarray 객체 생성 -----
```

```
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)
```

# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### ❖ 연결 - 2개 이상의 배열연결

- **hstack( )** : 수평 방향 연결
- **vstack( )** : 수직 방향 연결
- **dstack( )** : 깊이(depth)방향 연결
- **stack( )** : 사용자 지정차원으로 배열 연결
- **r\_[ ]** : hstack 명령과 비슷, 좌우로 연결
- **c\_[ ]** : 배열의 차원을 증가시킨 후 좌우로 연결
- **tile( )** : 동일한 배열을 **반복**하여 연결

특별한  
메서드  
인덱서  
(Indexer)

# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### ❖ 연결 - 2개 이상의 배열연결

➤ `hstack()` : 수평 방향 연결

5	6																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
---	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

전제조건  
행(row) 개수동일

7	14	21	28	35	42	49	7	14	21	7	14	21
56	63	70	77	84	91	98	56	63	70	56	63	70


# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### ❖ 연결 - 2개 이상의 배열연결

➤ `vstack( )` : 수직 방향 연결

전제조건  
열(column) 개수동일



28	35	42
77	84	91
7	14	21
56	63	70
7	14	21
56	63	70



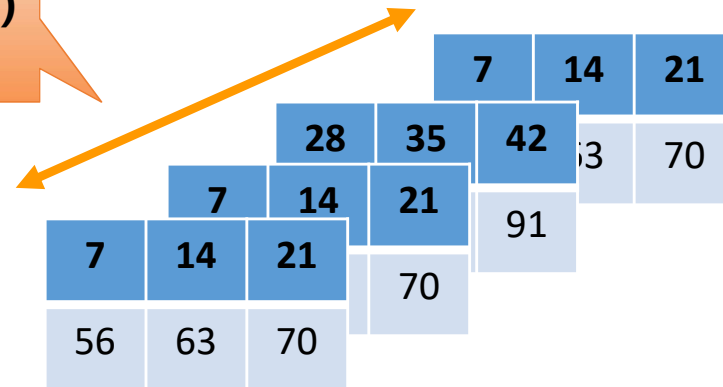
# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### ❖ 연결 - 2개 이상의 배열연결

➤ `dstack( )` : 깊이/채널/차원 방향 연결

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



# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### ❖ 연결 - 2개 이상의 배열연결

➤ `dstack( )` : 깊이/채널/차원 방향 연결

#### 전제조건

깊이(dept) / 채널(channel) / 차원(dimension)  
내부적 `reshape()` 3차원 변형 후 연결

1 X 7    7    14    21    28    35    42    49

(1, 7, 1)

1 X 7    56    63    70    77    84    91    98

(1, 7, 1)



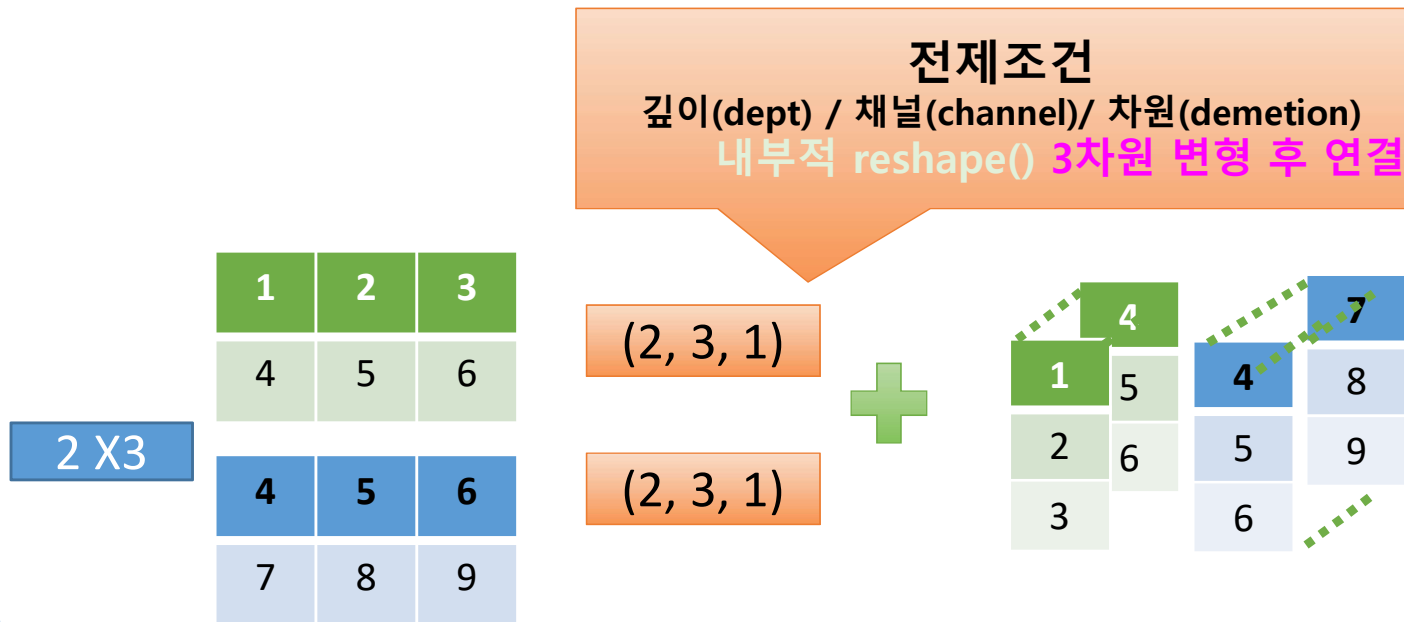
7	56
14	63
49	98

# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### ❖ 연결 - 2개 이상의 배열연결

➤ **dstack( )** : 깊이/채널/차원 방향 연결, 연결 후 3차원



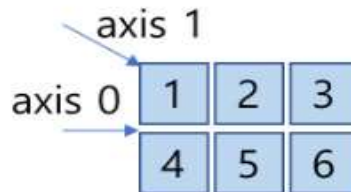
# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

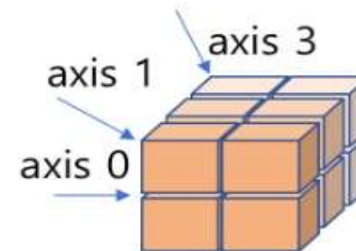
### ❖ 통계 관련 함수와 축(axis)



1D array



2D array



3D array



$1+2+3 \Rightarrow \text{sum}$   
 $(1+2+3)/3 \Rightarrow \text{mean}$   
 $1,2,3 \text{ 비교} \Rightarrow \text{max}$

axis=1



$1+4 \Rightarrow \text{sum}$   
 $(1+4)/2 \Rightarrow \text{mean}$   
 $1,4 \text{ 비교} \Rightarrow \text{max}$

axis=0

# 필수 라이브러리 NUMPY

## ◆ NumPy (Numerical Python)

### ❖ 통계 관련 함수

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

축(Axis) 주의 필요!

행(row)기준 계산 : axis=1

열(column)기준 계산 : axis=0

**차원 축소 연산**

(Dimension Reduction)

# 필수 라이브러리 NUMPY

## ◆ 기본 연산

벡터의 같은 인덱스에 위치한 원소(Element-wise)들끼리 연산  
수행하는 것 의미 → 벡터화 연산

- 명시적 반복문 사용하지 않고 배열 모든 원소에 대해 반복연산
- 기본적으로 두 배열의 shape가 정확히 같은 경우 가능
- 하나의 배열의 차원이 1인 경우 가능
- 각 차원을 비교했을 때 차원의 요소수가 동등 또는  
둘 중 하나의 차원의 요소 갯수가 1이면 가능

# 필수 라이브러리 NUMPY

## ◆ 기본 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  
answer= [ 2*di for di in data]
```

### NUMPY

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

```
2 * x
```



# 필수 라이브러리 NUMPY

## ◆ 기본 연산

행(row)벡터 1 x M 행렬

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

기본

열(column)벡터 M x 1 행렬

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



# 필수 라이브러리 NUMPY

## ◆ 기본 연산

행(row)벡터 1 x M 행렬

전치

열(column)벡터 M x 1 행렬

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

# 필수 라이브러리 NUMPY

## ◆ 기본 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 }')
```

# 필수 라이브러리 NUMPY

## ◆ 기본 연산



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

# 필수 라이브러리 NUMPY

## ◆ 기본 연산



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

# 필수 라이브러리 NUMPY

## ◆ 기본 연산

열(column)벡터  $M \times 1$  행렬

같은 인덱스에 위치한 원소(Element-wise)들끼리

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

# 필수 라이브러리 NUMPY

## ◆ 기본 연산

```
# 1차원 배열 ndarray 더하기 => By Index -----
```

```
x = np.arange(1, 10001)
```

```
y = np.arange(10001, 20001)
```

```
z = np.zeros_like(x)
```

```
print(f'---- BEFORE ----')
```

```
print(f'x => size : { x.size }, shape : {x.shape}, x[:10] : { x[:10] }')
```

```
print(f'y => size : { y.size }, shape : {y.shape}, y[:10] : { y[:10] }')
```

```
print(f'z => size : { z.size }, shape : {z.shape}, z[:10] : { z[:10] }')
```

```
for i in range(10000): z[i] = x[i] + y[i]
```

```
print(f'\n---- BEFORE ----')
```

```
print(f'z => size : { z.size }, shape : {z.shape}, z[:10] : { z[:10] }')
```

# 필수 라이브러리 NUMPY

## ◆ 기본 연산

```
# 1차원 배열 ndarray 더하기 => 덧셈 연산 -----
```

```
print(f'Wn----- x + y -----')
```

```
xy=x+y
```

```
print(f'z => size : { xy.size }, shape : {xy.shape}, xy[:10] : { xy[:10] }')
```

# 필수 라이브러리 NUMPY

## ◆ 기본 연산

```
# 1차원 배열 ndarray 같다 => Left == Right -----  
print(f'Wn---- x == y ----')  
xy= x==y  
print(f'x => size : { x.size }, shape : {x.shape}, x[:10] : { x[:10] }')  
print(f'y => size : { y.size }, shape : {y.shape}, y[:10] : { y[:10] }')  
print(f'z => size : { xy.size }, shape : {xy.shape}Wnxy[:10] : { xy[:10] }')
```

```
# 1차원 배열 ndarray 크기가 같다 => Left >= Right -----  
print(f'Wn---- x >= y ----')  
xy= x>=y  
print(f'x => size : { x.size }, shape : {x.shape}, x[:10] : { x[:10] }')  
print(f'y => size : { y.size }, shape : {y.shape}, y[:10] : { y[:10] }')  
print(f'z => size : { xy.size }, shape : {xy.shape}Wnxy[:10] : { xy[:10] }')
```



# 필수 라이브러리 NUMPY

## ◆ 기본 연산

```
x = np.linspace(5, 25, 5, dtype='i')
y = np.linspace(3, 15, 5, dtype='i')
print(f'x => size : { x.size }, shape : {x.shape}, x : { x }')
print(f'y => size : { y.size }, shape : {y.shape}, y : { y }')
```

*# 인덱스 매칭 요소 비교 결과 모두 True인 경우*

```
xy=np.all(x==y)
xy=np.all(x>=y)
xy=np.all(x<=y)
xy=np.all(x!=y)
```

*# 인덱스 매칭 요소 비교 결과 1개 이상 True*

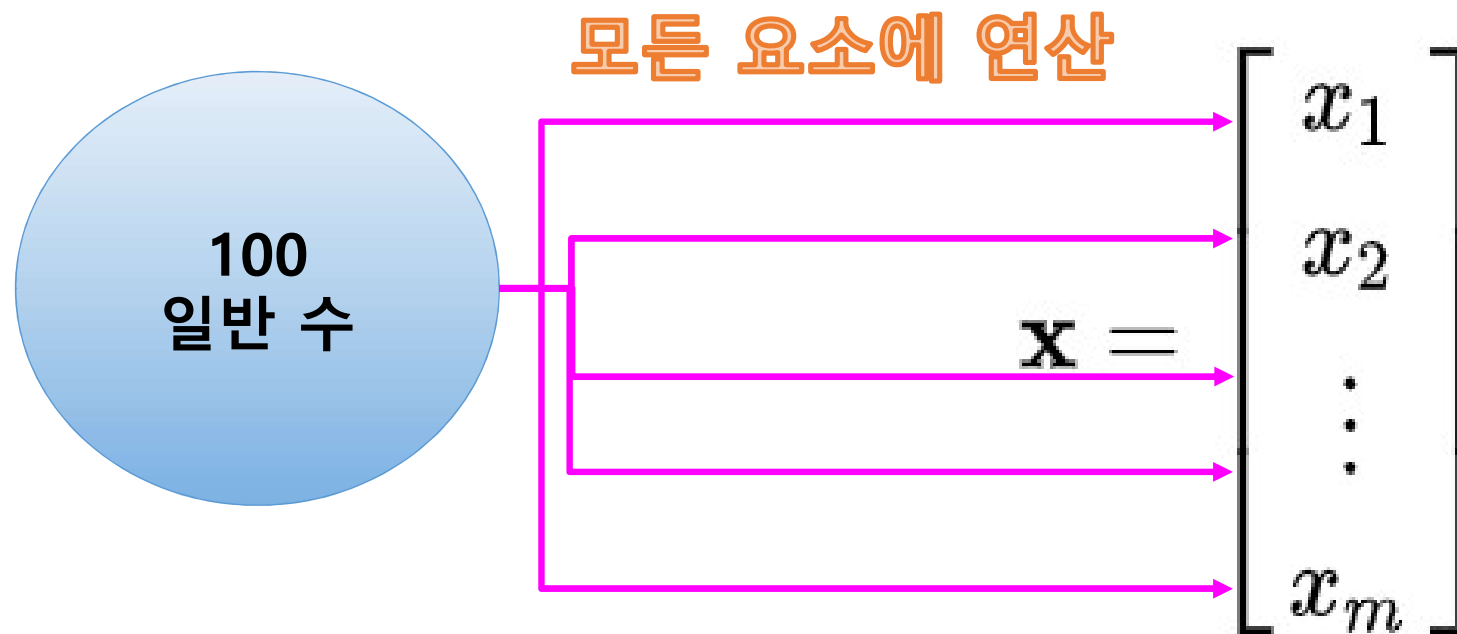
```
xy=np.any(x==y)
xy=np.any(x>=y)
xy=np.any(x<=y)
xy=np.any(x!=y)
```

# 필수 라이브러리 NUMPY

## ◆ 기본 연산

스칼라(Scalar)

열(column)벡터  $M \times 1$  행렬



# 필수 라이브러리 NUMPY

## ◆ 기본 연산

```
x = np.arange(1, 11)
print(f'x => size : { x.size }, shape : {x.shape}, x : { x }')

# 스칼라 즉 일반 수와 ndarray 객체 연산
x2 = 5 + x
print(f'5 + x => size : { x2.size }, shape : {x2.shape}, x2 : { x2 }')

x2 = 5 - x
print(f'5 - x => size : { x2.size }, shape : {x2.shape}, x2 : { x2 }')

x2 = 5 * x
print(f'5 * x => size : { x2.size }, shape : {x2.shape}, x2 : { x2 }')

x2 = 5 / x
print(f'5 / x => size : { x2.size }, shape : {x2.shape}, x2 : { x2 }')
```

# 필수 라이브러리 NUMPY

## ◆ 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

# 필수 라이브러리 NUMPY

## ◆ Broadcasting

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

자동확장

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

# 필수 라이브러리 NUMPY

## ◆ Broadcasting

자동 확장  
행(row) 갯수일치

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

# 필수 라이브러리 NUMPY

## ◆ Broadcasting

자동확장  
단! 요소수==컬럼수

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

# 필수 라이브러리 NUMPY

## ◆ Boolean Indexing Array

boolean인덱싱을 통해 array에서 원하는 행 또는 열의 값만 추출  
즉, 가리고 싶은 부분은 가리고, 원하는 요소만 꺼내는 방법

- Boolean Array 또는 Mask 라고도 함

### ■ 생성 방법

ndarray에 True, False 값을 가지는 연산 수행

→ boolean 배열 생성



# 필수 라이브러리 NUMPY

## ◆ Boolean Indexing Array

### ■ boolean 관련 연산자

논리연산자	기호		의미
and	&	x and y	x, y 둘 다 True이면 True
or		x or y	x, y 둘 중 하나라도 True이면 True
not	~	not x	x가 True면 False, False면 True

비교연산자		의미
<, <=	x < y , x <= y	x가 y보다 작으면 True x가 y보다 작거나 같으면 True
>, >=	x > y , x >= y	x가 y보다 크면 True x가 y보다 크거나 같으면 True
==, !=	x == y , x != y	x와 y가 같으면 True x와 y가 같지 않으면 True
is, is not	obj1 is obj2 obj1 is not obj2	객체 id가 동일하면 True 객체 id가 동일하지 않으면 True

# 필수 라이브러리 NUMPY

## ◆ Boolean Indexing Array

### ■ 생성 예시

```
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)
```

# 필수 라이브러리 NUMPY

## ◆ Boolean Indexing Array

- 사용 예시

```
# 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])
```

```
# 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])
```

# 필수 라이브러리 NUMPY

## ◆ Fancy Indexing Array

ndarray를 index value로 사용하는 것 즉, 추출하고 싶은  
데이터의 인덱스를 배열로가지고 있는 것

단! 반드시 정수타입이어야함

### ■ 생성 방법

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

➔ 반드시 integer로 선언

# 필수 라이브러리 NUMPY

## ◆ Fancy Indexing Array

- 사용 예시

```
# ndarray 객체 생성 -----  
a=np.array([2,4,6,8,9, float])  
b=np.array([0,0,3,1,2,4], int)  
  
# b배열의 값을 index로 하여 a의 배열 값 출력 -----  
a[b]  
a.take(b)
```

# 필수 라이브러리 NUMPY

## ◆ 통계 함수

함수	기능
sum()	전체 요소 합을 계산
mean()	전체 요소의 평균 계산
std(), var()	전체 요소의 표준편차, 분산 계산
min(), max()	전체 요소의 최솟값, 최댓값 계산
argmin(), argmax()	전체 요소의 최솟값, 최댓값의 위치 인덱스 반환
cumsum()	첫 번째 성분부터 누적합 계산
cumprod()	첫 번째 성분부터 누적곱 계산

# 필수 라이브러리 NUMPY

## ◆ 유용한 함수

함수	기능
<code>np.sort( ndarray )</code>	오름차순으로 정렬
<code>np.sort( ndarray )[::-1]</code>	내림차순으로 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.sort( ndarray, axis=0 )</code>	각 열을 오름차순으로 정렬
<code>np.sort( ndarray, axis=1 )</code>	각 행을 오름차순으로 정렬
<code>np.unique( ndarray )</code>	중복 성분 제거한 array 반환

# 필수 라이브러리 NUMPY

## ◆ Numpy

```
# 모듈로딩 -----  
import numpy as np  
  
# numpy 객체 생성 -----  
nparray01=np.array( [1,2,3,4,5] )  
print('type(nparray01) =', type(nparray01))  
print(nparray01)  
  
nparray02=np.array( [[1,2,3,4,5], [11,22,33]] )  
print('nparray02[0] =>', nparray02[0])  
print('nparray02[1] =>', nparray02[1])  
  
# nparray 연산  
nparray03=np.array([10,20,30,40,50])  
nparray04=nparray01+nparray03  
print(nparray04)
```



# 필수 라이브러리 NUMPY

## ◆ Numpy

```
# 모듈로딩 -----  
import numpy as np  
  
# numpy 객체 생성 -----  
nparr=np.array( [[1,2], [11,22], [111,222]] )  
print('type(nparr) =', type(nparr))  
print(nparr)  
  
print("=== Numpy Array 속성 ===")  
print('nparr.ndim =>', nparr.ndim, '차원')  
print('nparr.shape =>', nparr.shape)           # shape(col, row)  
print('nparr.size =>', nparr.size)  
print('nparr.dtype =>', nparr.dtype)  
print('nparr.itemsize =>', nparr.itemsize)  
print('nparr.data =>', nparr.data)
```

# 필수 라이브러리 NUMPY

## ◆ Numpy

```
print("Wn=== Numpy Array 메소드 ===")

print('nparr.max() =>', nparr.max())

print('nparr.min() =>', nparr.min())

print('nparr.sum() =>', nparr.sum()) # 원소 합계

print('nparr.sum(axis=0) =>', nparr.sum( axis=0 )) # 열(col)방향 합

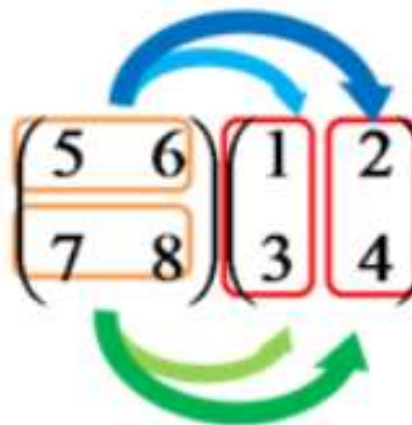
print('nparr.sum(axis=1) =>', nparr.sum( axis=1 )) # 행(row)방향 합

print('nparr.mean() =>', nparr.mean() ) # 원소값 평균
```

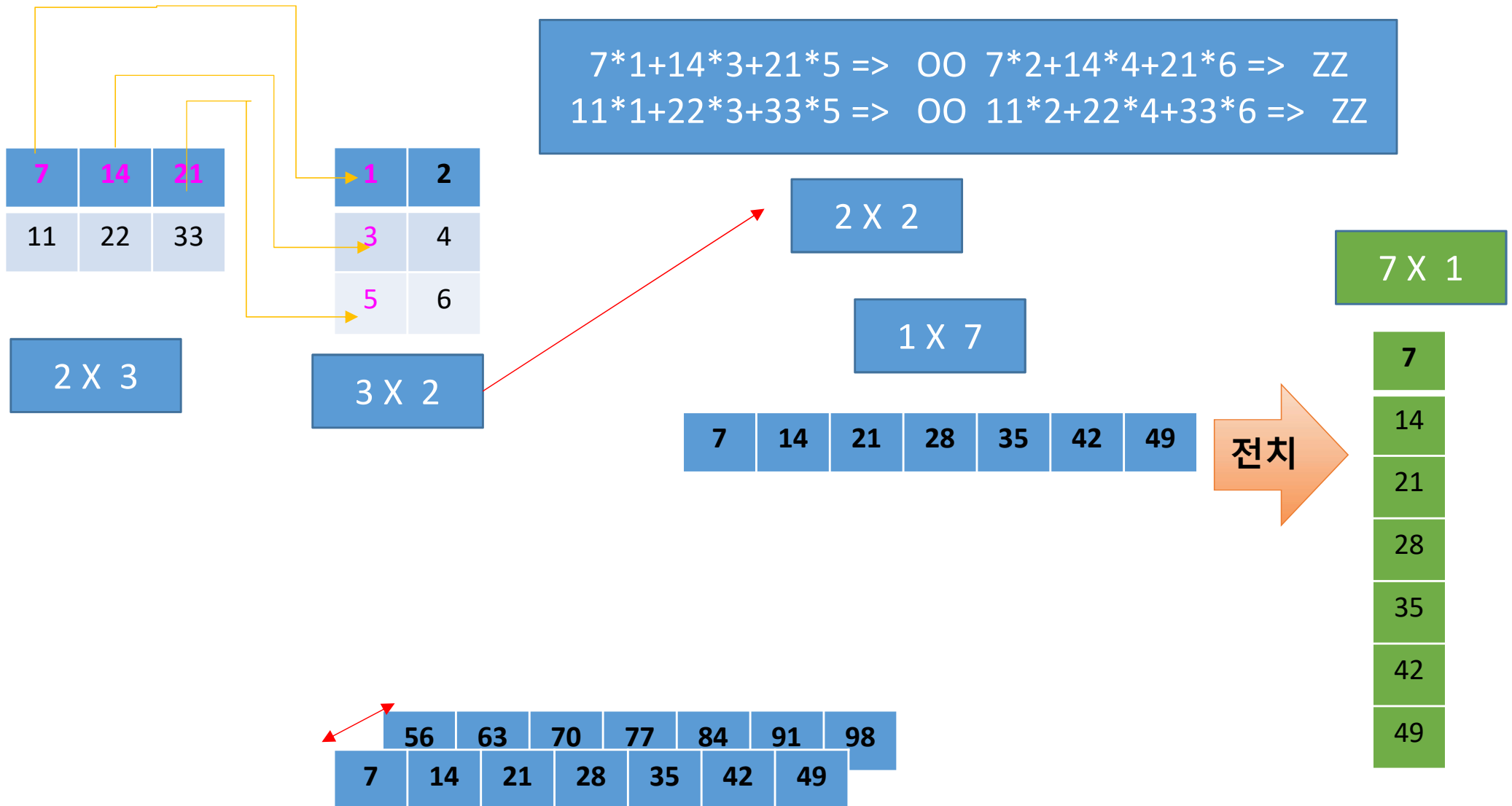
# 필수 라이브러리 NUMPY

## ◆ 행렬(Matrix) 곱

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

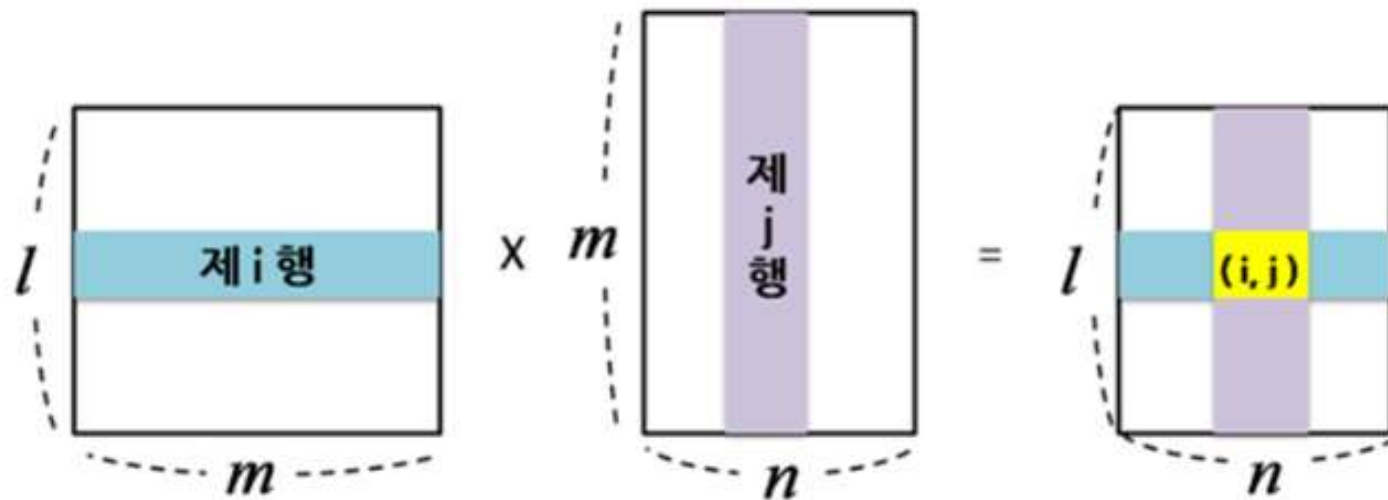

$$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 5 \times 1 + 6 \times 3 & 5 \times 2 + 6 \times 4 \\ 7 \times 1 + 8 \times 3 & 7 \times 2 + 8 \times 4 \end{pmatrix}$$

# 필수 라이브러리 NUMPY



# 필수 라이브러리 NUMPY

## ◆ 행렬(Matrix) 곱



A행렬

$\times$

B행렬

$=$

AB행렬

$l \times m$

$m \times n$

$l \times n$

# 필수 라이브러리 NUMPY

## ◆ 행렬(Matrix) 곱

함수	기능
np.dot( arr1, arr2 )	array1의 열 갯수 == array2의 갯수
np.matmul(arr1, arr2 )	
arr1@arr2	