

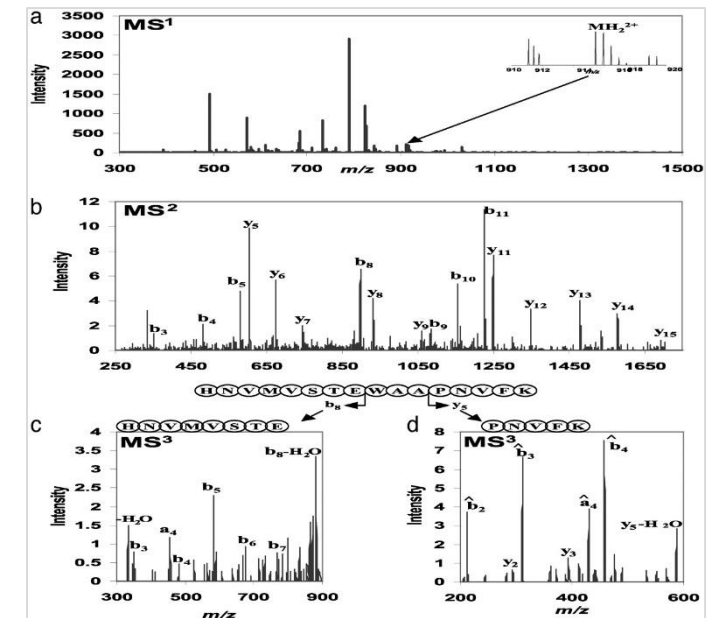
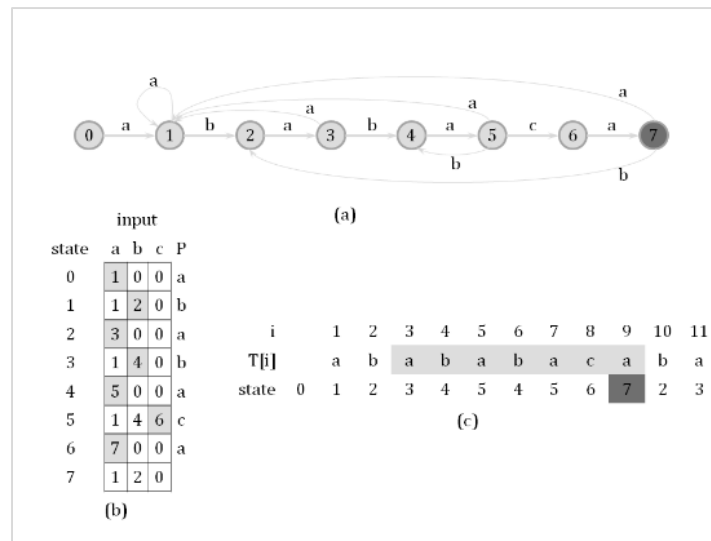
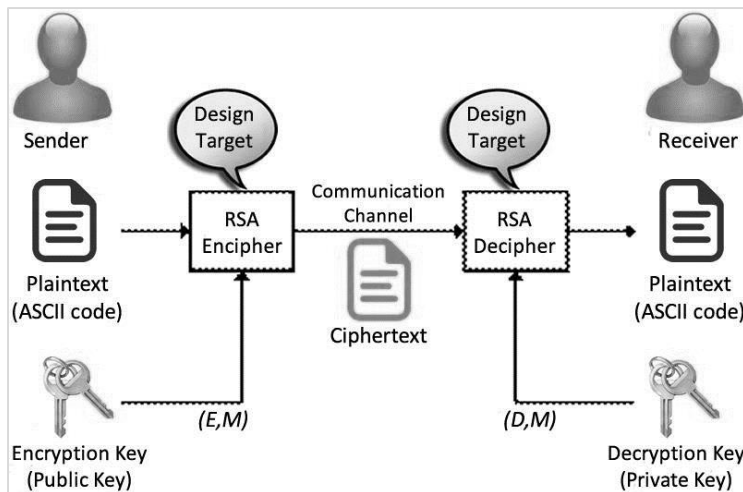


NumPy

Hosung Jo

■ 조호성

- 한양대학교 SW융합원 SW교육전담교수
- 한양대학교 전자컴퓨터통신 박사
 - 알고리즘 (정보보호, 문자열매칭, 생물정보학)



■ NumPy 소개

- Python과 NumPy
- NumPy를 사용하는 이유

■ NumPy의 기초

- Arrays, Shaping, Transposition
- Mathematical Operations
- Indexing and Slicing



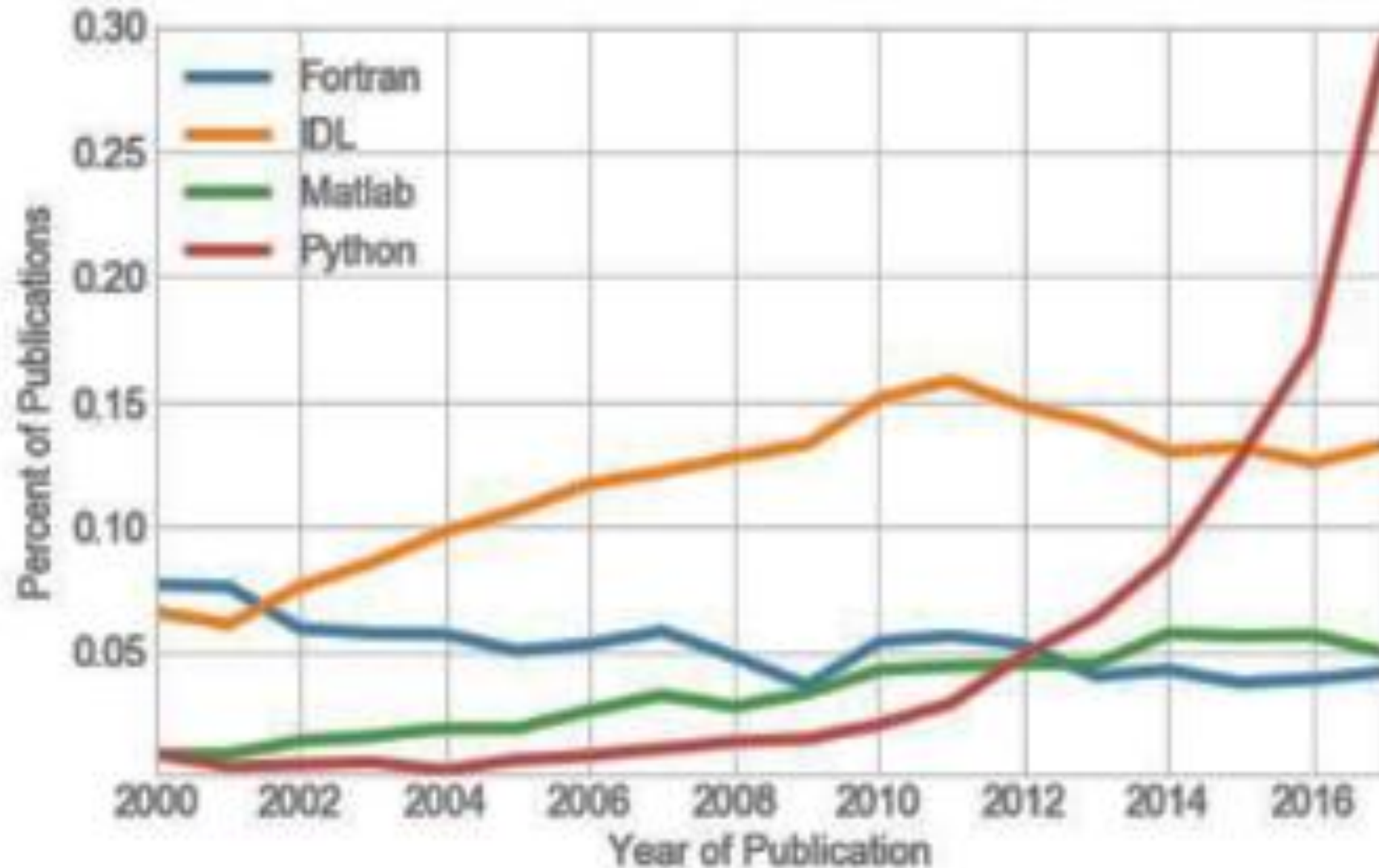


■ Python

- 오픈소스 기반의 하イレ벨 컴퓨터 언어
- 간결하여 이해가 쉽고, 컴퓨터과학 분야의 활용도가 많음



Mentions of Software in Astronomy Publications:



■ Python

- 오픈소스 기반의 하イレ벨 컴퓨터 언어
- 간결하여 이해가 쉽고, 활용 분야가 넓음

■ Python의 장점

- 다른 컴퓨터 언어와의 상호 운용성 (Glue Language)
- 단순성과 동적 특성
- OpenSource 기반의 개발환경

■ Python의 수행속도

- Python은 느리다.
- 굳이 더 복잡한 계산을 Python으로?

Slowest things on earth:



■ Python

- 오픈소스 기반의 하이레벨 컴퓨터 언어
- 간결하여 이해가 쉽고, 활용 분야가 넓음

■ Python의 장점

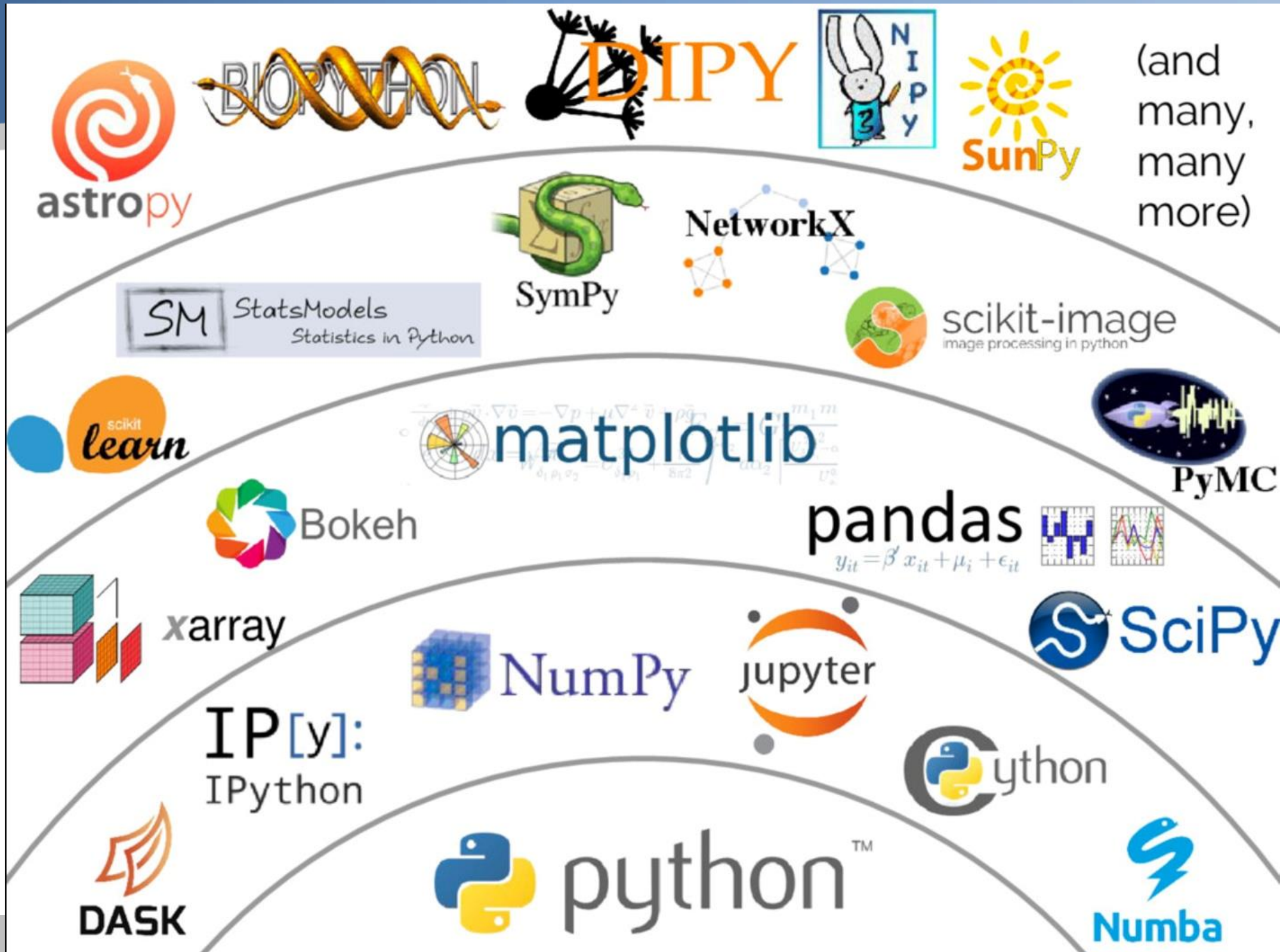
- 다른 컴퓨터 언어와의 상호 운용성 (Glue Language)
- 단순성과 동적 특성
- OpenSource 기반의 개발환경
- 웬만한 건 다 있는 Third-party 라이브러리



Python Library

■ Powerful Third-party Library

- array 컴퓨팅을 지원하는 NumPy
- 과학문서작성과 연구를 정리하는 Ipython 과 Jupyter
- 분산컴퓨팅을 위한 Numba
- 데이터프레임을 사용하는 Pandas
- 데이터 시각화를 위한 matplotlib과 Bokeh
- 기계학습과 이미지 프로세싱을 처리하는 Scikit-learn과 Scikit-Image





NumPy란?

NumPy = Number + Python



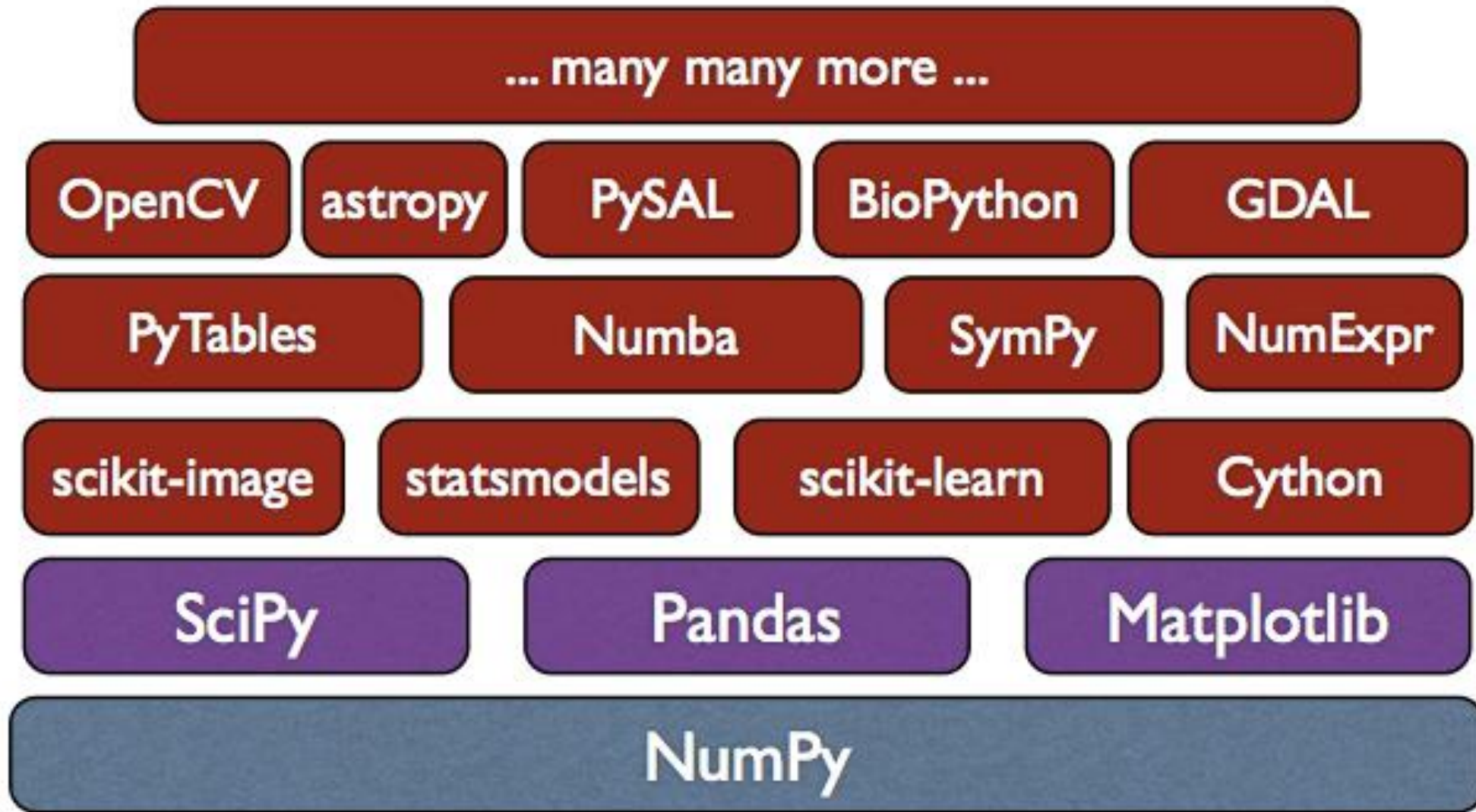
NumPy란?

NumPy = Number + Python

- 파이썬 라이브러리
- 행렬이나 대규모 다차원 배열을 쉽게 처리
- 계산과학 (computational science) 분야의 복잡한 연산을 지원
- Scipy나 Matplotlib, Pandas 등으로 발전, 더 복잡한 연산을 쉽게 처리가능하도록 지원함.



NumPy vs. SciPy vs. Pandas





History of NumPy

■ History of NumPy

- 1995, started a project *Numeric*, the first array packages
- 2001, developed to *Scipy* (based on *Numeric*)
- 2005, named as *NumPy*
- 2006, included in Python Standard Library



History of NumPy

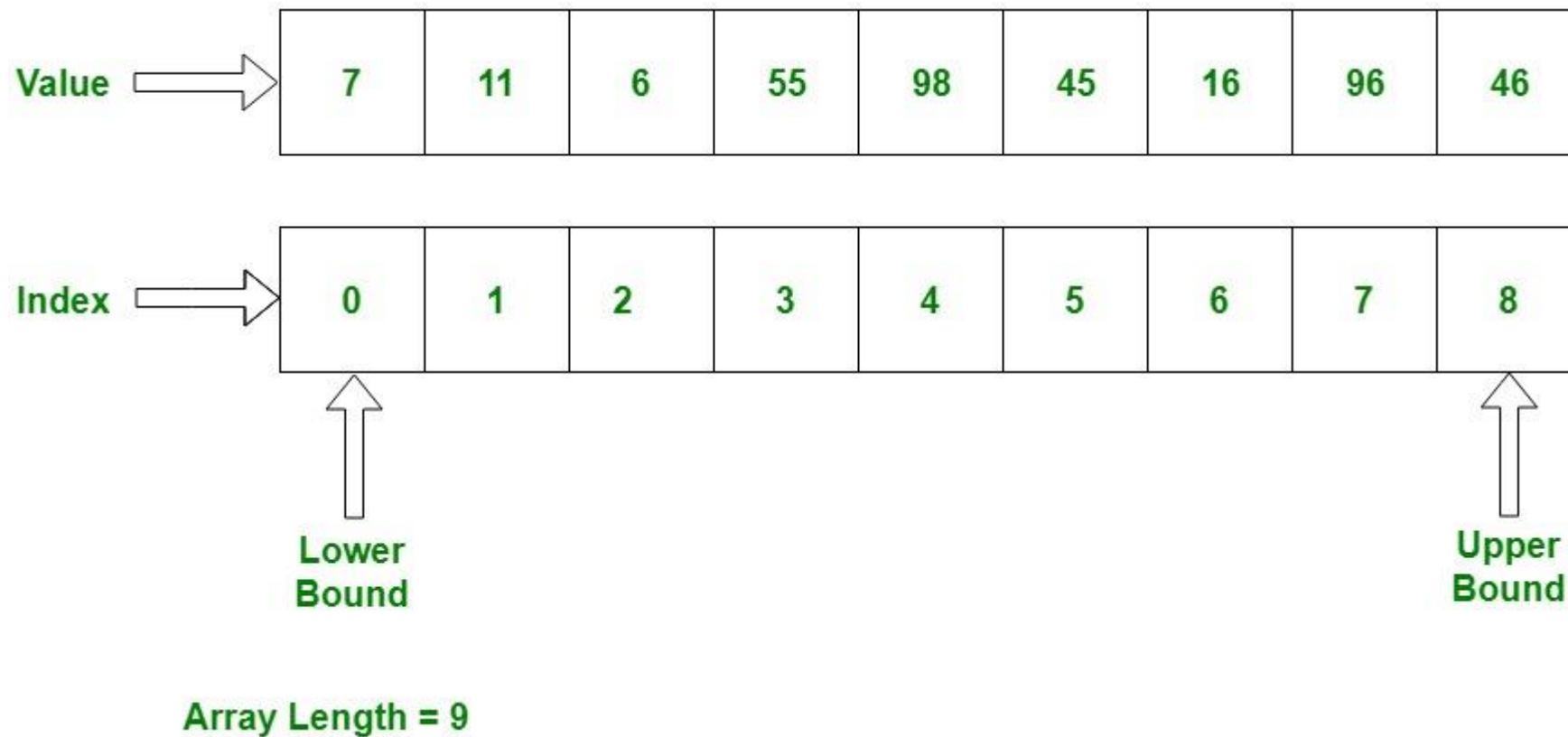
■ History of NumPy

- 1995, started a project *Numeric*, the first array packages
- 2001, developed to *Scipy* (based on Numeric)
- 2005, named as *NumPy*
- 2006, included in Python Standard Library
- **Current, NumPy v1.18. dev0**
 - <http://www.numpy.org>



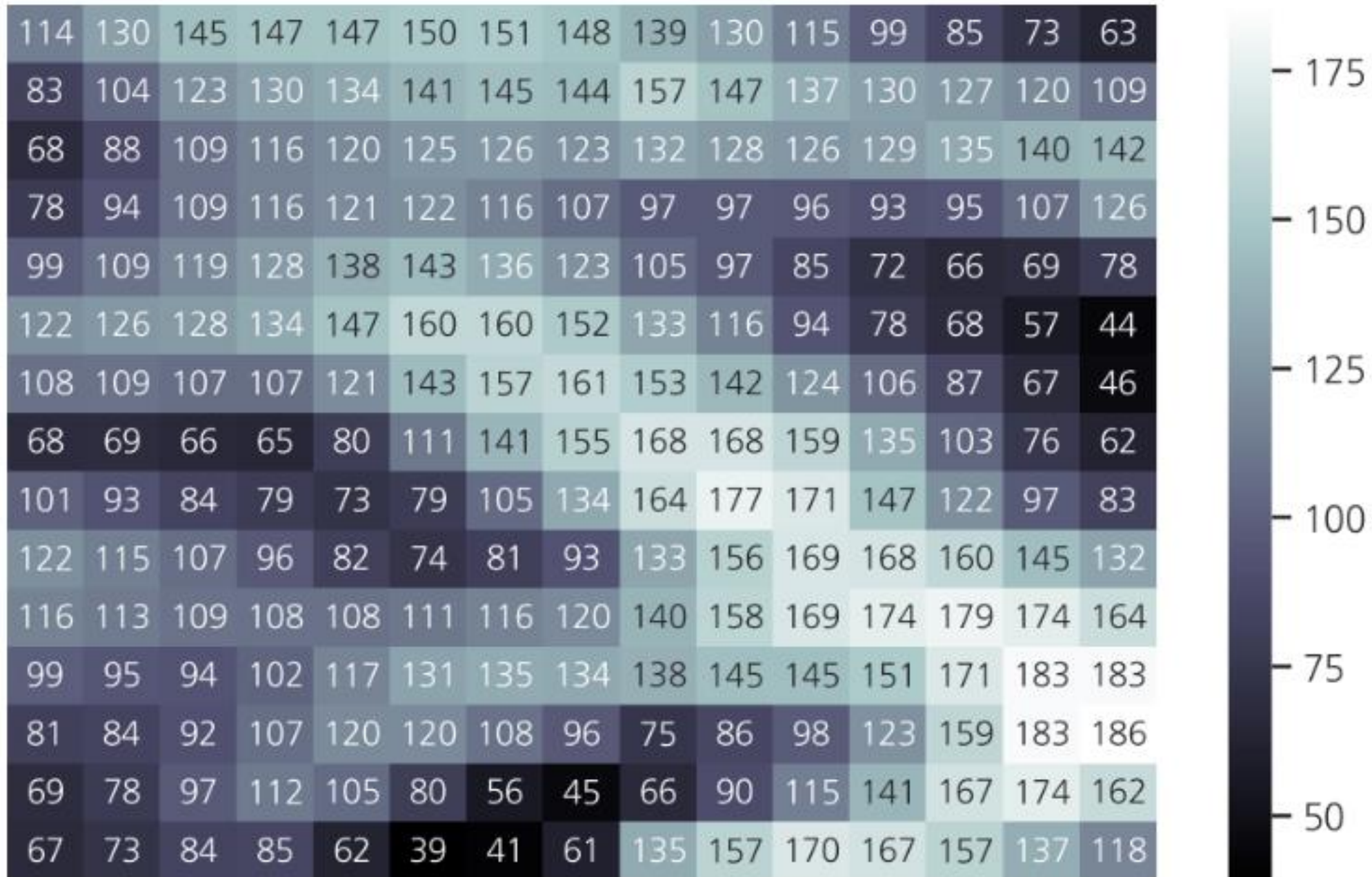
리스트, 벡터, 행렬, 배열

- 문자열 처리에 적합





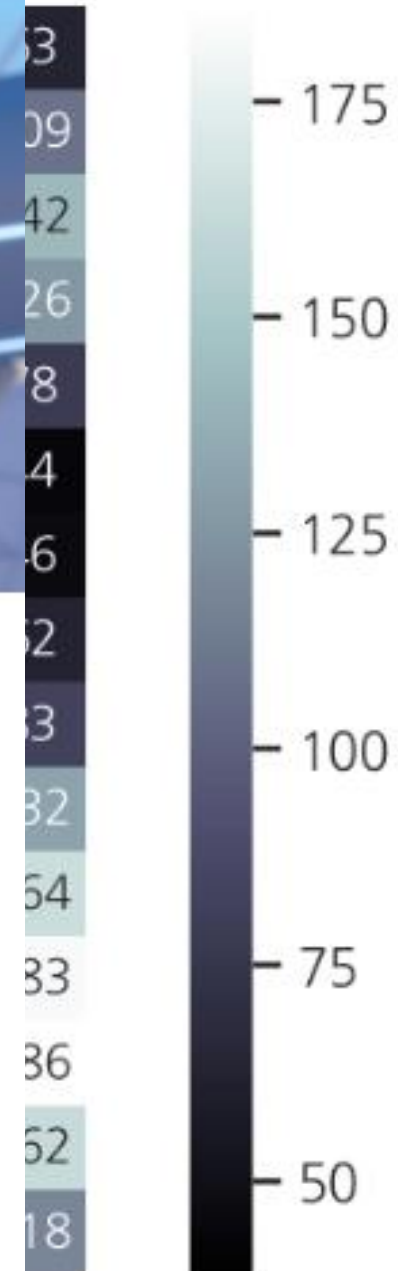
리스트, 벡터, 행렬, 배열





리스트, 벡터

114	130	145
83	104	123
68	88	109
78	94	109
99	109	119
122	126	128
108	109	107
68	69	66
101	93	84
122	115	107
116	113	109
99	95	94
81	84	92
69	78	97
67	73	84





리스트, 벡터



175

150

125

100

75

50

■ Python의 수행속도

- Python은 느리다.
- 굳이 더 복잡한 계산을 Python으로?
- 빨라질까? 그래도 느리지 않을까?

Slowest things on earth:





리스트, 벡터, 행렬, 배열

■ 수행시간 비교

- 정수 리스트를 2개 만들고, 각 원소를 곱하는 연산을 수행
- %time 을 사용하여 수행시간을 확인
- `ResultC = inputA*inputB`
 - `inputA=[1,2,3,4,5,6,7,8,9,10]`
 - `inputB=[1,2,3,4,5,6,7,8,9,10]`



리스트, 벡터, 행렬, 배열

■ 수행시간 비교

- 정수 리스트를 2개 만들고, 각 원소를 곱하는 연산을 수행
- %time 을 사용하여 수행시간을 확인
- $\text{ResultC} = \text{inputA} * \text{inputB}$
 - $\text{inputA} = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
 - $\text{inputB} = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
 - $\text{ResultC} = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]$



리스트, 벡터, 행렬, 배열

■ 수행시간 비교

- 정수 리스트를 2개 만들고, 각 원소를 곱하는 연산을 수행
- %time 을 사용하여 수행시간을 확인
- 아래 두 경우 시간을 각각 측정
 - A) NumPy를 사용하지 않고 곱셈을 하는 경우,
 - B) NumPy를 사용하여 곱셈을 하는 경우,



리스트, 벡터, 행렬, 배열

■ 수행시간 비교

- 아래 두 경우 시간을 각각 측정
 - A) NumPy를 사용하지 않고 곱셈을 하는 경우,
 - B) NumPy를 사용하여 곱셈을 하는 경우,
- 크기를 바꾸면서 다음 표를 완성하시오.

	1,000	10,000	100,000	10^6	10^9
Python					
NumPy					



NumPy란?

NumPy = Number + Python

- 파이썬 라이브러리
- 행렬이나 대규모 다차원 배열을 쉽게 처리
- 계산과학 (computational science) 분야의 복잡한 연산을 지원
- Scipy나 Matplotlib, Pandas 등으로 발전, 더 복잡한 연산을 쉽게 처리가능하도록 지원함.



배열(Arrays)

- Vectors
- Matrices
- Images
- Tensors
- ConvNets

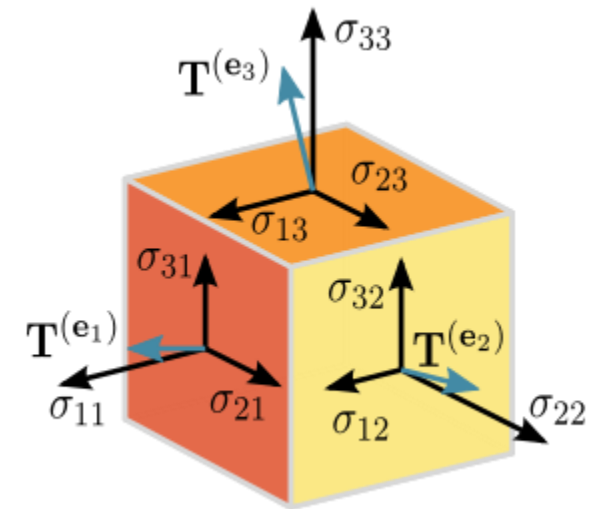
$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$



배열(Arrays)

- Vectors
- Matrices
- Images
- Tensors
- ConvNets



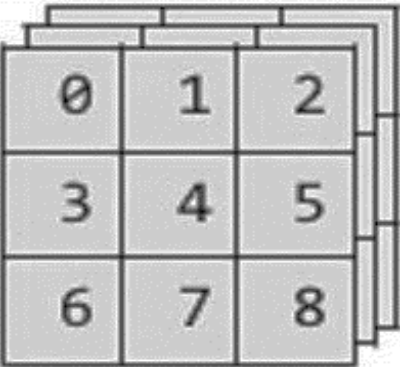
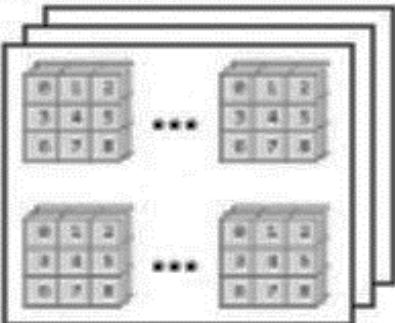


■ ndarray class

- 배열을 이용한 벡터화연산
- 배열에는 모두 동일한 자료형만 저장 가능(★)
- 메모리에 연속적으로 저장(★)
- 일반적으로 리스트보다 빠른 것으로 간주



리스트, 벡터, 행렬, 배열

Dimensions	Example	Terminology
1		Vector
2		Matrix
3		3D Array (3 rd order Tensor)
N		ND Array

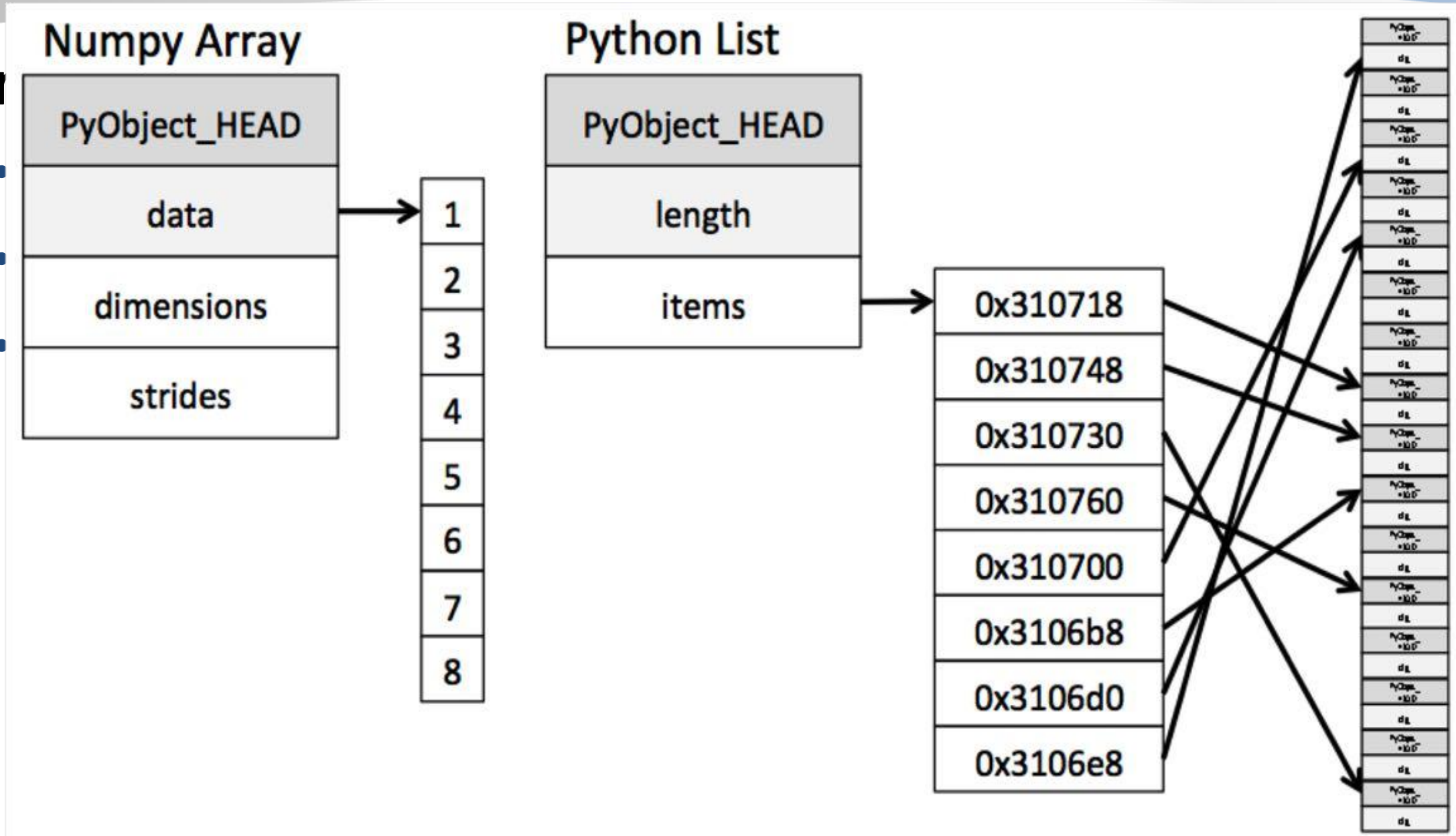
■ ndarray class

- 배열을 이용한 벡터화연산
- 배열에는 모두 동일한 자료형만 저장 가능(★)
- 메모리에 연속적으로 저장(★)
- 일반적으로 리스트보다 빠른 것으로 간주

```
shell
>>> a = [1, 2, 3]
>>> b = a * 2
>>> b
[1, 2, 3, 1, 2, 3]
```

```
shell
>>> a = [1, 2, 3]
>>> b = []
>>> for n in a:
>>>     b.append(n*2)
>>> b
[2, 4, 6]
```

```
shell
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = []
>>> for i in range(3):
>>>     c.append(a[i] + b[i])
>>> c
[5, 7, 9]
```





배열(Arrays)

■ NumPy 패키지 импорт

```
import numpy as np
```

- NumPy의 배열을 사용하기 위해 numpy 패키지를 импорт
- 일반적으로 np라는 이름을 사용



배열(Arrays)

■ 1-D array 생성하기

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

```
ar
```

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

- array 명령어를 사용하여 직접 값을 입력할 수 있음
- 리스트와 같아 보이지만 다른 타입임 (type(ar))



배열(Arrays)

■ 2-D array 생성하기

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

```
ar
```

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

- `len(ar), len(ar[0])`으로 행과 열의 길이를 확인 가능



배열(Arrays)

■ 3-D array 생성하기

```
ar = np.array([[[1,2,3], [4,5,6]],  
               [[7,8,9], [10,11,12]]])
```

ar

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

- shape, ndim을 이용한 행렬 구성과 차수 확인 가능



배열(Arrays)

■ 다음과 같은 배열을 생성하시오.

```
array([10, 8, 15, 7, 19])
```

```
array([[10, 20, 30],  
       [40, 50, 60],  
       [70, 80, 90]])
```

```
array([[[100, 90, 80],  
        [ 70, 60, 50]],
```

```
       [[ 15, 25, 35],  
        [ 45, 55, 65]],
```

```
       [[ 11, 22, 33],  
        [ 44, 55, 66]])])
```

— 생성한 배열에 대해 type, len, ndim, shape 으로 확인해보시오



배열(Arrays)

■ 생성하기

- `np.ones`
- `np.zeros`
- `np.arange`
 - `np.reshape`
- `np.random.random`



배열(Arrays)

■ 생성하기

- np.ones
- np.zeros
- np.arange
 - np.reshape
- np.random.random

```
In [5]: import numpy as np  
arrayA=np.zeros((3,6))
```

```
In [6]: arrayA
```

```
Out[6]: array([[0., 0., 0., 0., 0., 0.],  
               [0., 0., 0., 0., 0., 0.],  
               [0., 0., 0., 0., 0., 0.]])
```

```
In [2]: import numpy as np  
arrayA=np.ones((3,6))
```

```
In [3]: arrayA
```

```
Out[3]: array([[1., 1., 1., 1., 1., 1.],  
               [1., 1., 1., 1., 1., 1.],  
               [1., 1., 1., 1., 1., 1.]])
```



배열(Arrays)

■ 생성하기

- np.ones
- np.zeros

```
In [7]: np.arange(100, 110)
```

```
Out[7]: array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109])
```

- np.arange
 - np.reshape

```
In [10]: np.arange(100, 110).reshape(2, 5)
```

```
Out[10]: array([[100, 101, 102, 103, 104],  
                [105, 106, 107, 108, 109]])
```

- np.random.random



배열(Arrays)

■ 생성하기

- np.ones
- np.zeros
- np.arange
 - np.reshape
- np.random.random

```
np.random.random((2,5,3))
```

```
array([[ [0.50304424, 0.38336065, 0.13551835],  
        [0.65835036, 0.6299836 , 0.06980386],  
        [0.33456302, 0.12452275, 0.27404641],  
        [0.81429317, 0.97796819, 0.17747256],  
        [0.11925552, 0.38933514, 0.70769086]],  
      [[0.94521757, 0.39078975, 0.61647959],  
        [0.08273138, 0.38627341, 0.26111908],  
        [0.47311267, 0.37812702, 0.09469965],  
        [0.38228631, 0.54241897, 0.1197545 ],  
        [0.41486631, 0.2775834 , 0.23600334]]])
```




배열(Arrays)

■ 다음과 같은 결과가 나오도록 배열을 생성하시오.

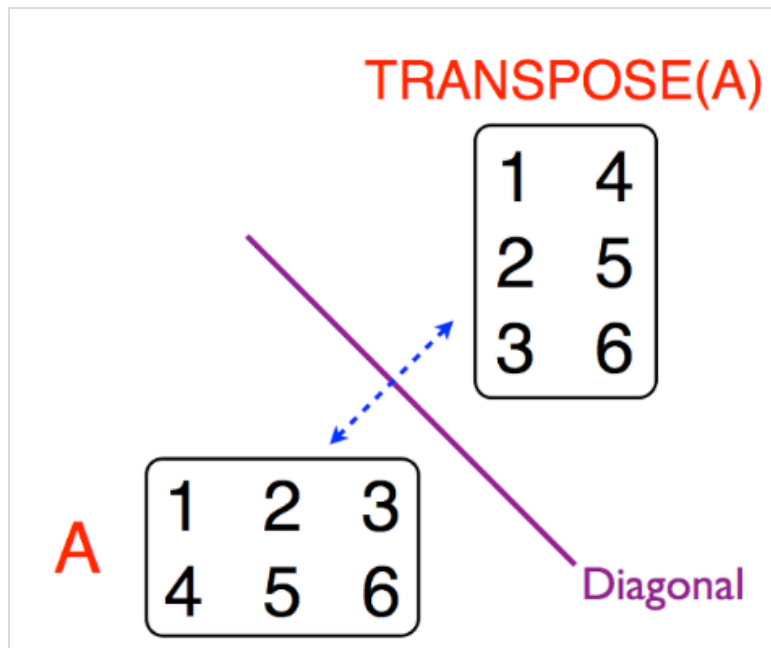
- (1) `array([[10, 11, 12, 13, 14],
 [15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24]])`
- (2) `array([[11, 12, 13, 14, 15],
 [16, 17, 18, 19, 20],
 [21, 22, 23, 24, 25],
 [26, 27, 28, 29, 30],
 [31, 32, 33, 34, 35]],
 [[36, 37, 38, 39, 40],
 [41, 42, 43, 44, 45],
 [46, 47, 48, 49, 50],
 [51, 52, 53, 54, 55],
 [56, 57, 58, 59, 60]])`
- (3) `array([[[74, 79, 84, 89],
 [94, 99, 104, 109],
 [114, 119, 124, 129]],
 [[134, 139, 144, 149],
 [154, 159, 164, 169],
 [174, 179, 184, 189]],
 [[194, 199, 204, 209],
 [214, 219, 224, 229],
 [234, 239, 244, 249]])`



배열(Arrays)

■ Transpose

- 행과 열을 교체
- `ar.T`

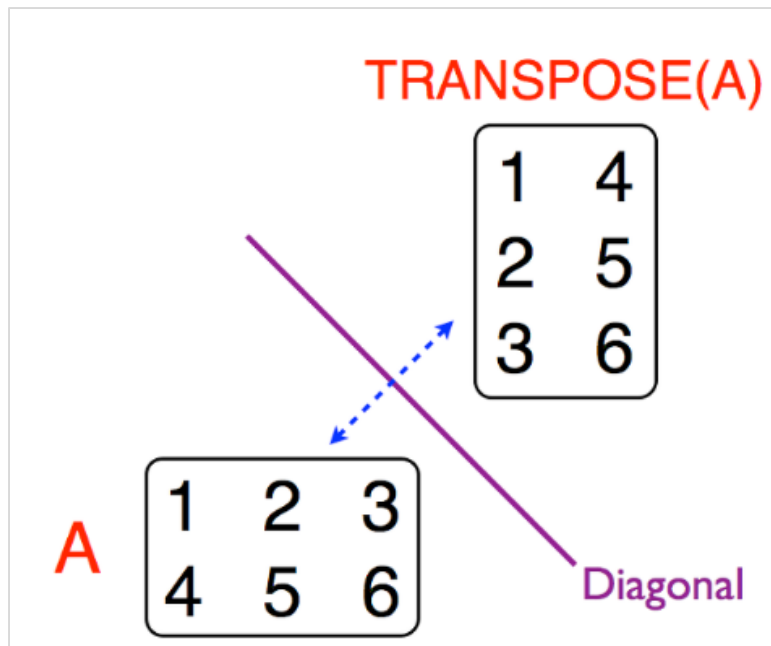




배열(Arrays)

■ Transpose

- 행과 열을 교체
- `ar.T`



```
np.arange(10).reshape(2,5)
```

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

```
np.arange(10).reshape(2,5).T
```

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



배열(Arrays)

■ Slicing(슬라이싱)

- 1차원 배열: 리스트와 동일
- N차원 배열: ‘,’ 로 구별하여 연결

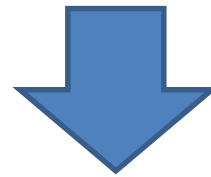


List Explanation

■ List

- 여러 개의 값을 효과적으로 저장하고 사용하는 구조

```
>>> students1 = '김한양'
>>> students2 = '김두양'
>>> students3 = '홍길동'
>>> students4 = '이도령'
```



```
>>> students = ['김한양', '김두양', '홍길동', '이도령']
```



List Explanation

■ Lists

- 리스트의 값들을 item이라고 부른다.

```
>>> students = ['김한양', '김두양', '홍길동', '이도령']
```



item



List Explanation

■ Lists

- 리스트의 값들을 item이라고 부른다.
- Item을 구분하기 위해 번호를 붙이고 index 라고 부른다.

0	1	2	3
'김한양'	'김두양'	'홍길동'	'이도령'

```
>>> students = ['김한양', '김두양', '홍길동', '이도령']
```

item

index

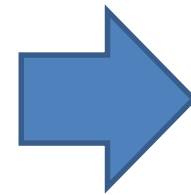


배열(Arrays)

■ Slicing(슬라이싱)

- 1차원 배열: 리스트와 동일
- N차원 배열: ‘,’ 로 구별하여 연결

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



```
ar[0:, 1:]
```

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

```
ar[1:, 1:]
```

```
array([[ 5,  6,  7],  
       [ 9, 10, 11]])
```

```
ar[1:2, 1:2]
```

```
array([[5]])
```




배열(Arrays)

■ 다음 결과를 확인하시오.

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

```
ar[1:, 1:]
```

```
ar[:2, :1]
```

```
ar[1:2, 3:]
```

```
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]])])
```

```
ar[1:, 1:, :1]
```

```
ar[1:, 0:, :3]
```

```
ar[1:3, 1:4, 1:5]
```



■ Vectorization operation

- 배열화 계산
- 원소 하나 하나씩 계산하는 것이 아니라 배열 단위로 계산
- Loops 없이 연산하여 코딩의 양이 줄어듦
- 병렬처리가 가능 (multi-core)

■ Vectorization operation

- 배열화 계산
- 원소 하나 하나씩 계산하는 것이 아니라 배열 단위로 계산
- Loops 없이 연산하여 코딩의 양이 줄어듦
- 병렬처리가 가능 (multi-core)

shell

```
>>> a = [1, 2, 3]
>>> b = a * 2
>>> b
[1, 2, 3, 1, 2, 3]
```

shell

```
>>> a = [1, 2, 3]
>>> b = []
>>> for n in a:
>>>     b.append(n*2)
>>> b
[2, 4, 6]
```

shell

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = []
>>> for i in range(3):
>>>     c.append(a[i] + b[i])
>>> c
[5, 7, 9]
```

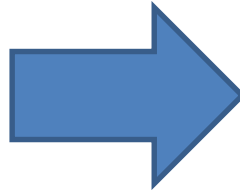


배열(Arrays)

■ Vectorization operation

shell

```
>>> a = [1, 2, 3]
>>> b = a * 2
>>> b
[1, 2, 3, 1, 2, 3]
```



```
ar = np.array([1,2,3,4,5])
ar*2
array([ 2,  4,  6,  8, 10])
```

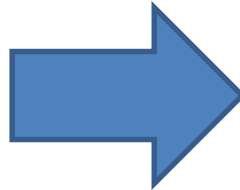


배열(Arrays)

■ Vectorization operation

shell

```
>>> a = [1, 2, 3]
>>> b = a * 2
>>> b
[1, 2, 3, 1, 2, 3]
```



```
ar = np.array([1,2,3,4,5])
ar*2
array([ 2,  4,  6,  8, 10])
```

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

```
array([ 9, 12, 15])
```

```
c > 10
```

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

■ Vectorization operation

- 사칙연산 모두 가능
 - 덧셈, 뺄셈, 곱셈, 나눗셈
- 논리연산과 비교연산도 가능
 - $>$, $<$, $==$, $!=$
- 지수함수, 로그함수 지원
 - Exp , $**$, $\log()$



$$x = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

$$x + y = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 + 0 \\ 11 + 1 \\ 12 + 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 12 \\ 14 \end{bmatrix}$$

$$x - y = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 - 0 \\ 11 - 1 \\ 12 - 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}$$



$$x \cdot y = \langle x, y \rangle = x^T y$$

$$x^T y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$



배열(Arrays)

■ 다음을 수행하시오.

- [7, 5, 9, 10]과 [4, 10, 6, 2] 인 2개의 배열을 생성
 - 1) 각 배열의 값을 3배로 만든 다음 더한 결과를 출력하시오.
- [[7, 5, 9, 10], [4, 10, 6, 2]]과 [[1, 3, 4, 9], [14, 7, 6, 4]]인 2개의 배열을 생성
 - 2) 앞의 배열에서 뒤의 배열을 뺀 뒤 각 값이 양수인지 아닌지를 True, False로 출력하시오.

■ 브로드캐스팅(broadcasting)

- 행렬끼리 연산할 때 크기가 다른 경우 이를 알아서 확대 해주는 기능
 - 특별히 작업해줄 것은 없음

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

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

■ 유용한 기능들

- min, max, argmin, argmax, sum, mean, median
 - axis, sort
 - var, std
 - exp, sqrt, sin, cos
 - dot
-
- 3x4x4의 임의의 정수 배열을 생성한 후 각 기능을 수행해보시오.