NumPy/SciPy

http://www.scipy-lectures.org/index.html

http://www.labri.fr/perso/nrougier/teaching/numpy.100/index.html

https://www.labri.fr/perso/nrougier/from-python-to-numpy/

~~Previous on~~ Numerical Computing

for expert (scientist, engineer)
= money

efficiency / convenience is very important!

~~for beginner~~

MATLAB

https://numpy.org/doc/stable/user/numpy-for-matlab-users.html

Matplotlib?

Travis Oliphant BOARD MEMBER AND CO-FOUNDER

I was a fairly proficient MATLAB user, but it was not memory efficient enough.

- Loved the expressive syntax of Python
- Loved the fact that slicing didn't make copies
- Loved the existing multiple data-types
- Loved how much more flexible it was to extend than MATLAB was
- Loved that I could read the source code and extend it

General purpose? 2D?

CONTINUUM ANALYTICS

But....

■ Computing Power

○ GPU

○ Multi Processor

○ Parallel Computing

■ Compiler (Language)

○ Cython—This is the most commonly used tool for compiling to C, covering both numpy and normal Python code (requires some knowledge of C)

○ Shed Skin—An automatic Python-to-C converter for non-numpy code

○ Numba—A new compiler specialized for numpy code

○ Pythran—A new compiler for both numpy and non-numpy code

○ PyPy—A stable just-in-time compiler for non-numpy code that is a replacement for the normal Python executable

■ Glue Language (Library)

■ Algorithm / Data Structure

	Cython	Shed Skin	Numba	Pythran	PyPy
성숙함	Y	Y	–	–	Y
널리 사용 중	Y	–	–	–	–
Numpy 지원	Y	–	Y	Y	–
기존 코드를 깨지 않음	–	Y	Y	Y	Y
C 언어 지식 필요	Y	–	–	–	–
OpenMP 지원	Y	–	Y	Y	Y

NumPy

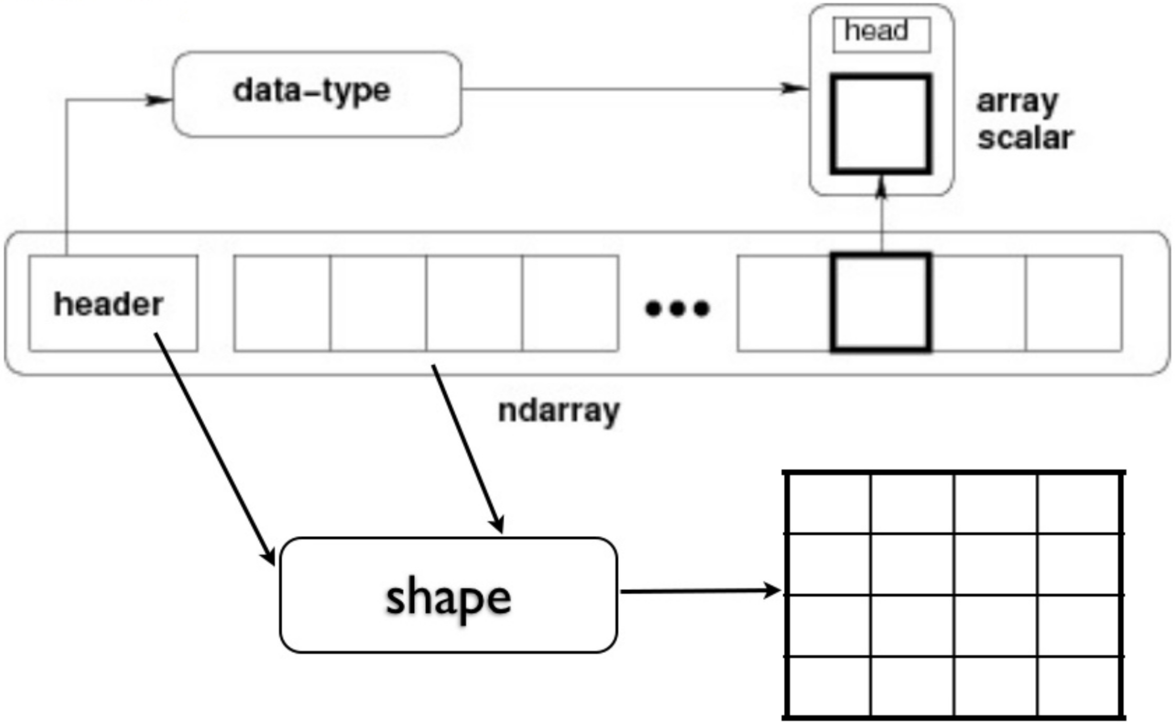


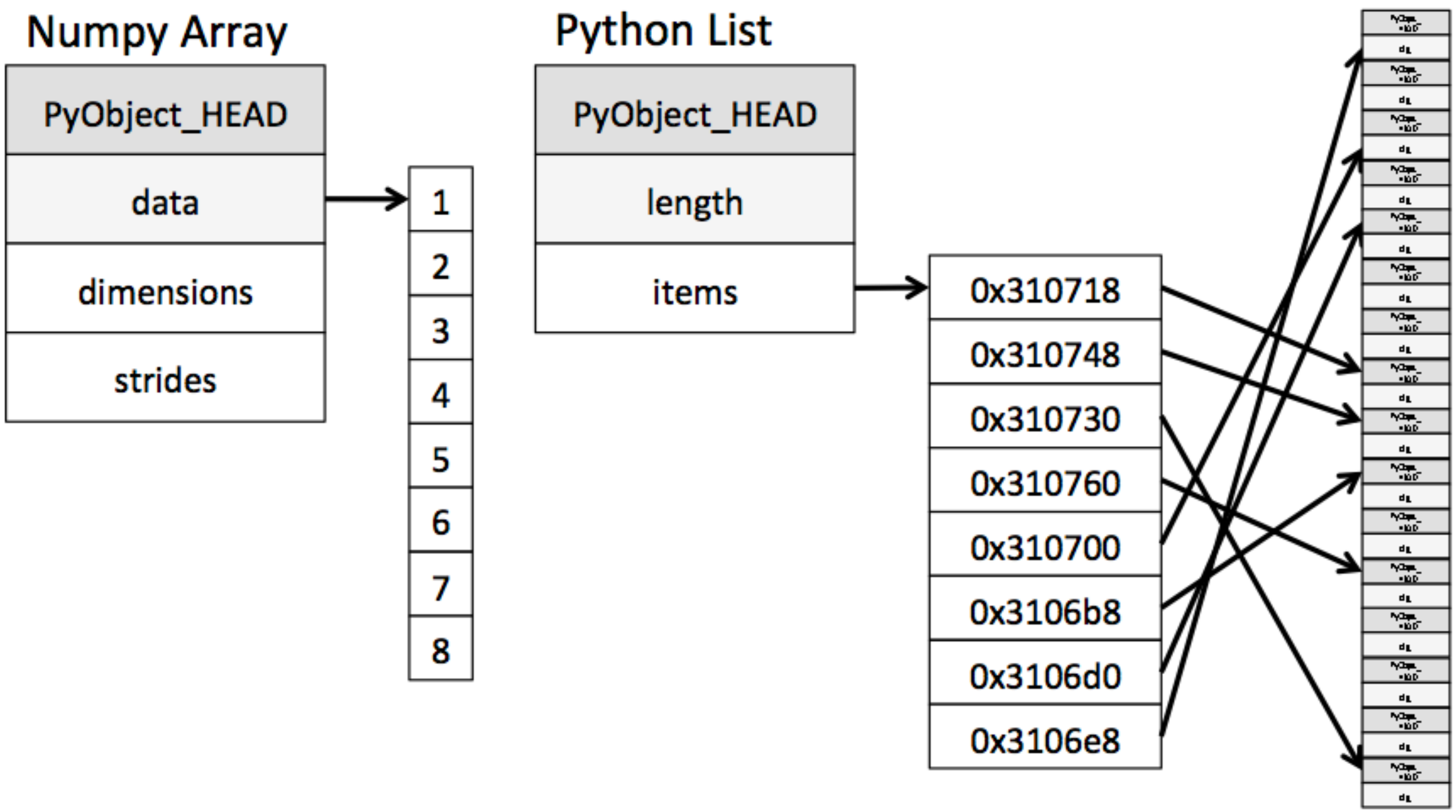

Person	Package	Year
Jim Fulton	Matrix Object in Python	1994
Jim Hugunin	Numeric	1995
Perry Greenfield, Rick White, Todd Miller	Numarray	2001
Travis Oliphant	NumPy	2005

ndarray

block of memory + indexing scheme + data type descriptor

- raw data
- how to locate an element
- how to interpret an element

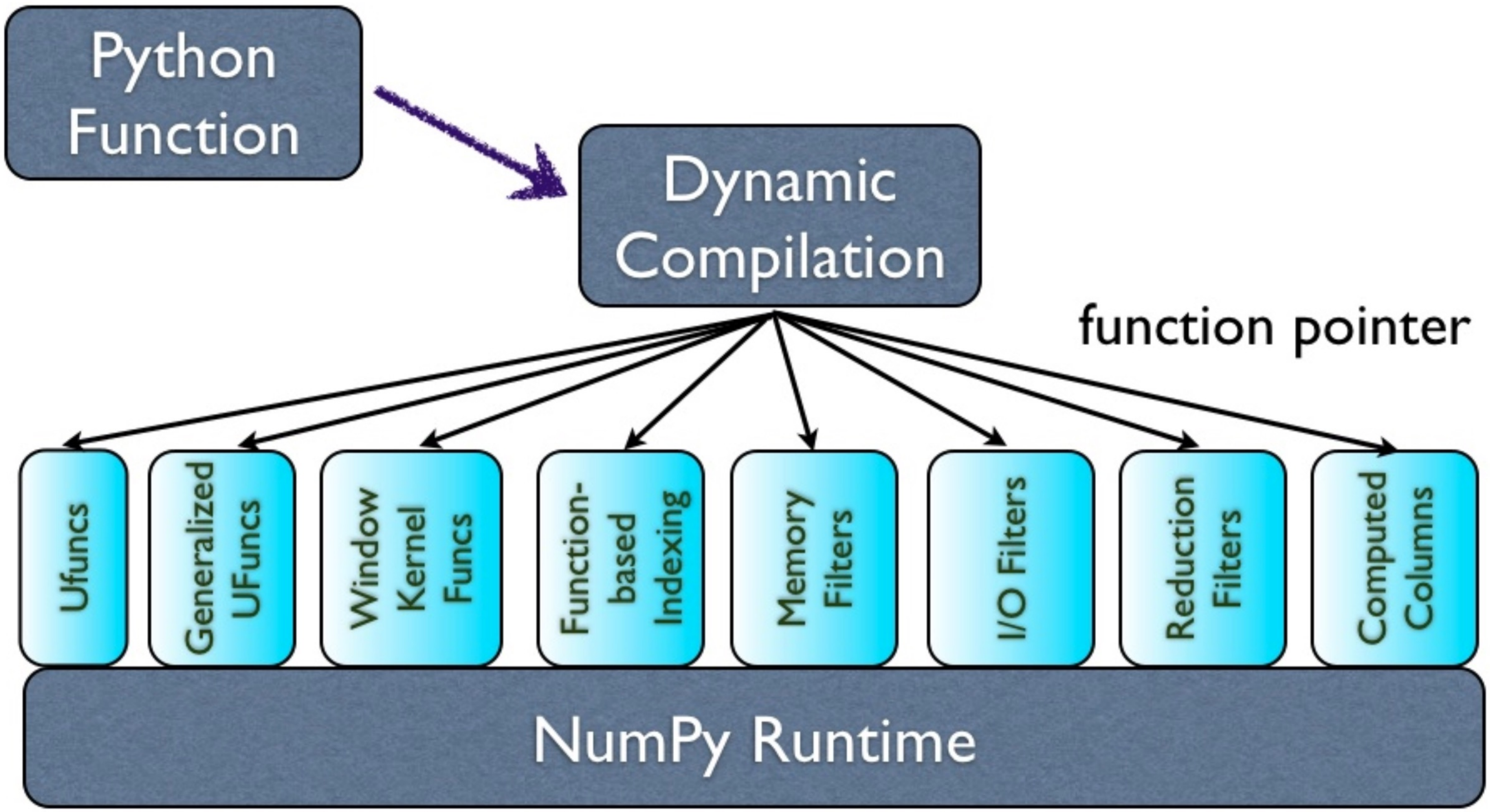




	ndarray	List
메모리 사용 특징	- 한 주소값으로 부터 연속적으로 메모리에 elem값저장	Elem값이 이산적인 주소값에 저장
연산 특징	sequential한 elem값 연산에 유리	Elem 단위 연산에 유리
구현	Array list (C/C++ array)	Linked list
	homogeneous	heterogeneous

list
포인터의배열
각각 객체가 메모리 여기저기 흩어져 있을 수 있음
= 캐시 활용이 어려움

```c
typedef struct PyArrayObject {
    PyObject_HEAD
    /* Block of memory */
    char *data;
    /* Data type descriptor */
    PyArray_Descr *descr;
    /* Indexing scheme */
    int nd;
    npy_intp *dimensions;
    npy_intp *strides;
    /* Other stuff */
    PyObject *base;
    int flags;
    PyObject *weakreflist;
} PyArrayObject;
```


Python Function
Dynamic Compilation
function pointer
Ufuncs
Generalized UFuncs
Window Kernel Funcs
Function-based Indexing
Memory Filters
I/O Filters
Reduction Filters
Computed Columns
NumPy Runtime

■ Vectorization

○ Arrays are important because they enable you to express **batch operations** on data **without writing any for loops**. This is usually called vectorization. Any arithmetic operations between equal-size arrays applies the operation elementwise.

- ➢ 벡터화하여 계산

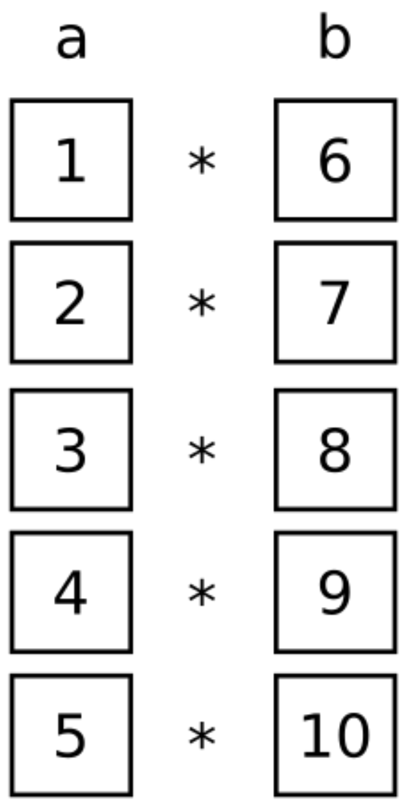
○ 실제 코딩의 양을 줄일뿐만아니라, 벡터 계산은 병렬 계산이 가능하기 때문에, Multi Core 활용 가능

- ➢ CPU 지원 (*vector processor*)
- ➢ https://blogs.msdn.microsoft.com/nativeconcurrency/2012/04/12/what-is-vectorization/
- ➢ **GPU 지원 ?**

○ But NumPy는 대형 배열에 최적화된 라이브러리라는 한계가 존재

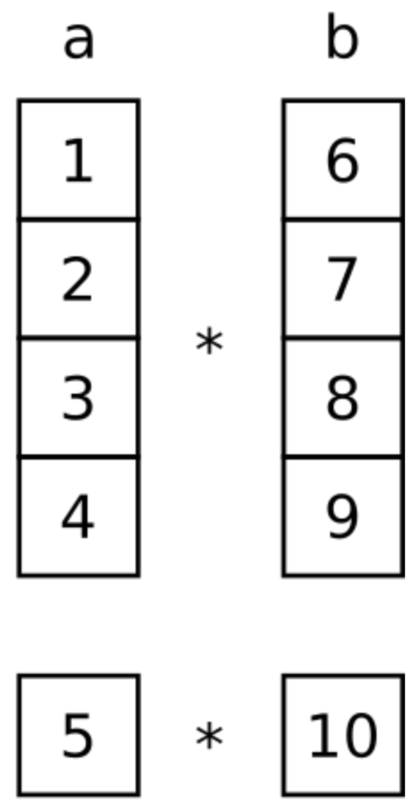
- ➢ 실제로 배열의 크기가 100개 이내인 경우 NumPy는 순수 파이썬 구현 보다도 오히려 낮은 성능을 보일때가 있음

not vectorized



5 operations

vectorized

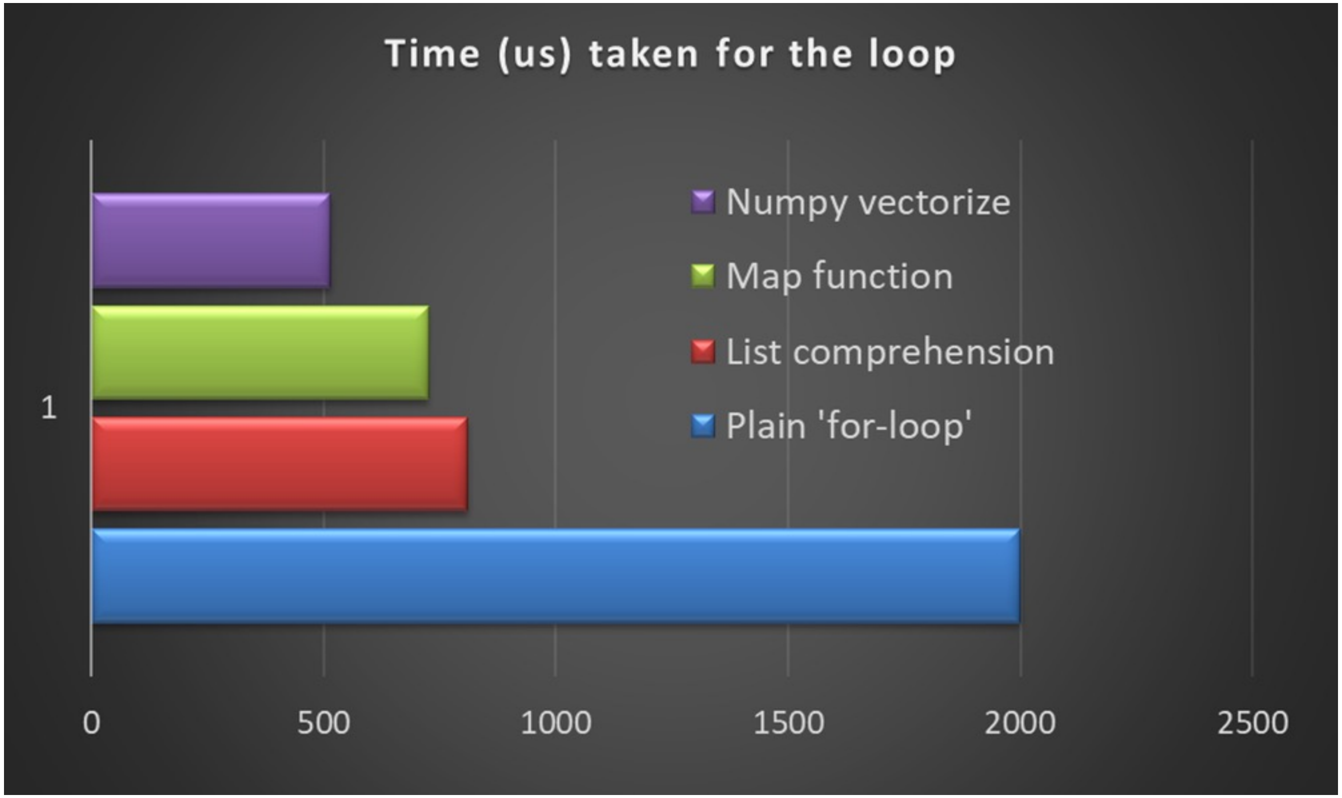


2 operations

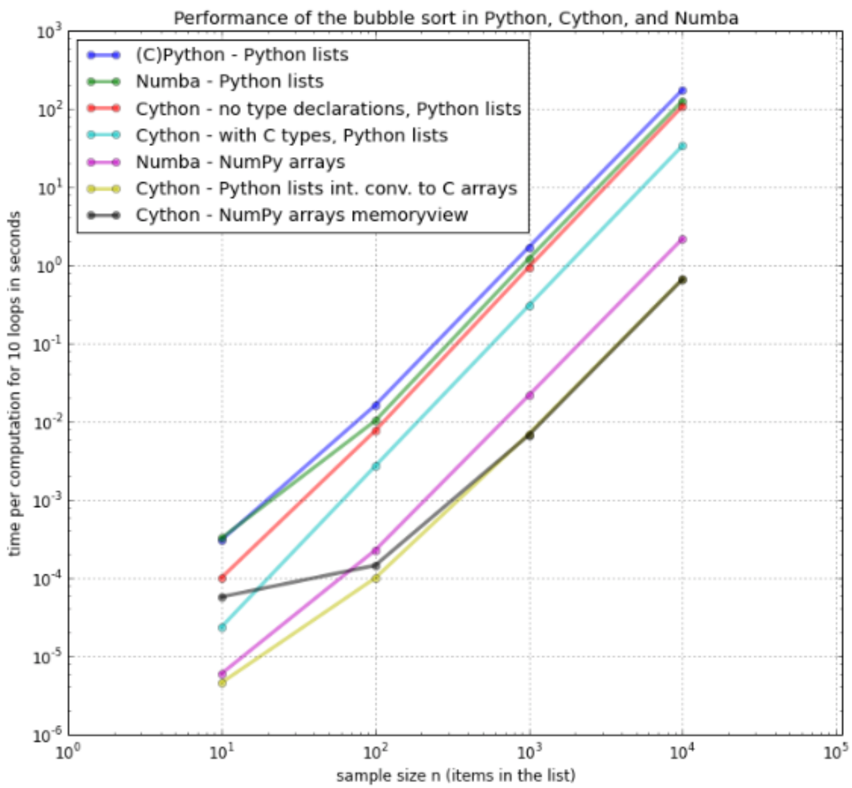
- stride

- broadcasting

- ufunc



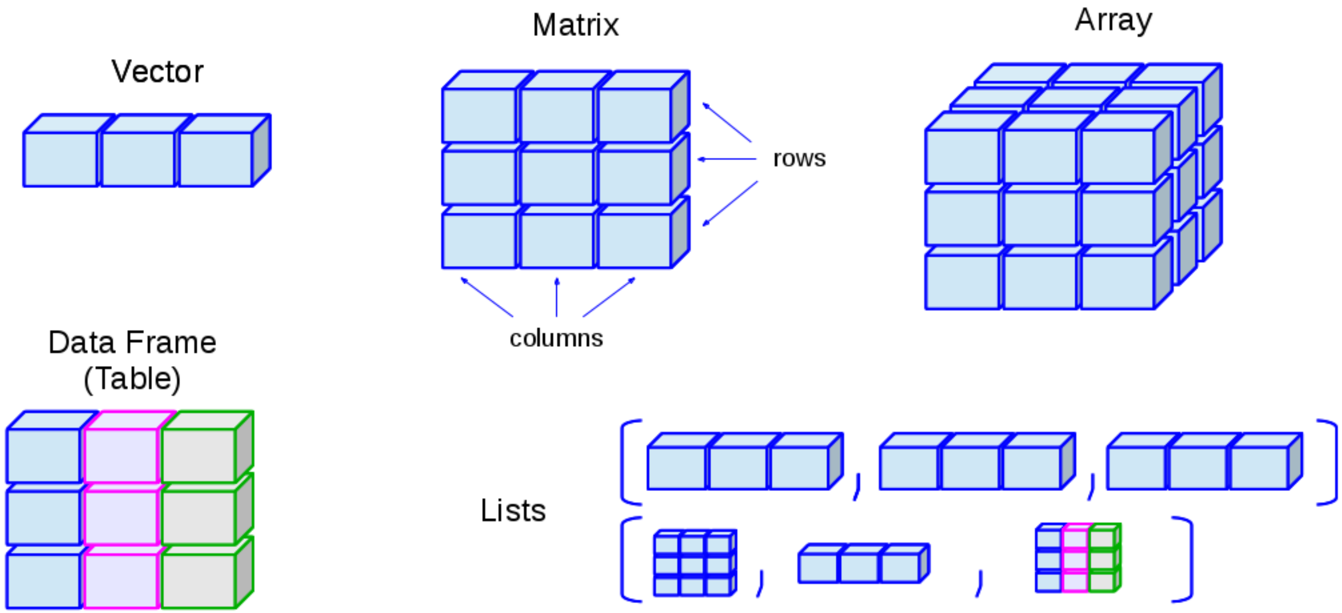
https://www.codementor.io/tirthajyotisarkar/data-science-with-python-turn-your-conditional-loops-to-numpy-vectors-he1yo9265



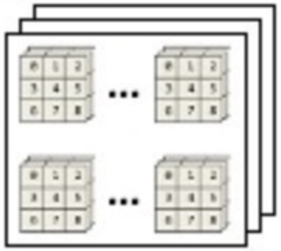
n=10000	bubblesort function	time in sec.	rel. performance gain
	(C)Python - Python lists	173.37	1.0
	Numba - Python lists	125.39	1.38
	Cython - no type declarations, Python lists	106.67	1.63
	Cython - with C types, Python lists	33.39	5.19
	Numba - NumPy arrays	2.16	80.39
	Cython - Python lists int. conv. to C arrays	0.66	262.87
	Cython - NumPy arrays memoryview	0.65	266.19

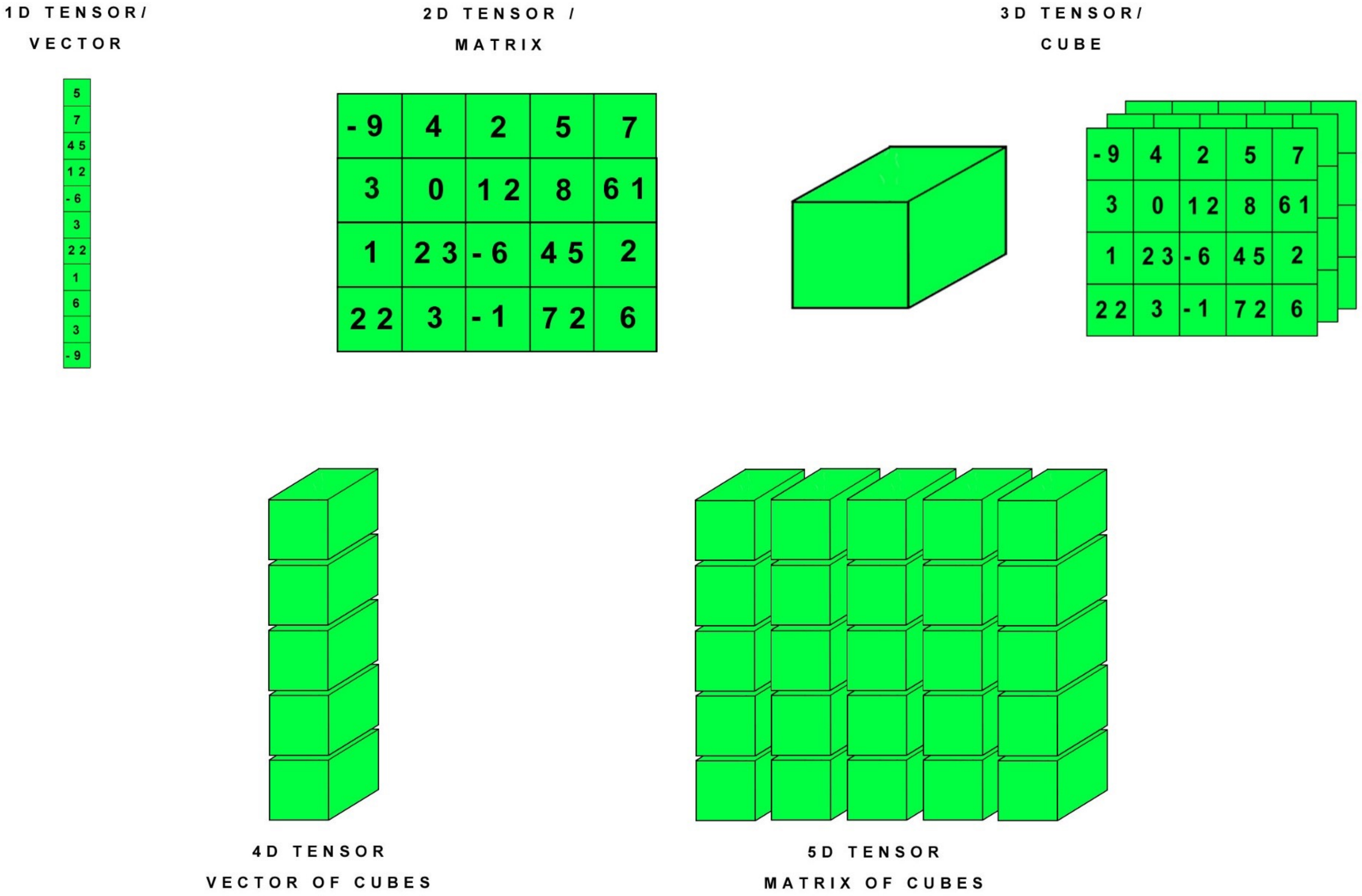
https://stackoverflow.com/questions/23661636/poorer-performance-of-cython-with-numpy-array-memoryview-compared-to-c-arrays

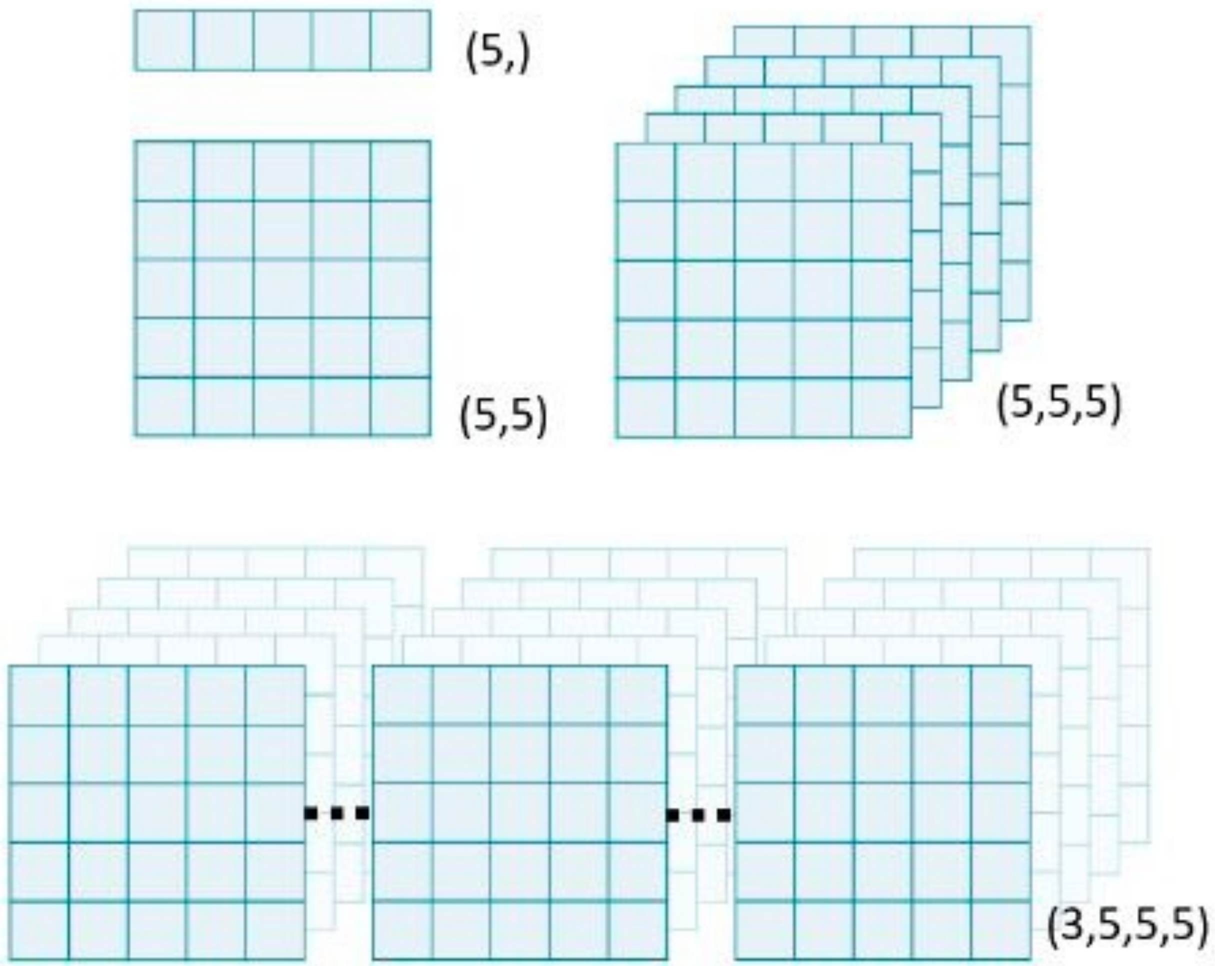
Array



•"Array" is the data structure. It provides O(1) access. Basically, *you store data in arrays*.
•"Vector" is the mathematics concept.

Dimensions	Example	Terminology
1	$\begin{bmatrix} 0 & 1 & 2 \end{bmatrix}$	Vector
2	$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$	Matrix
3	$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$	3D Array (3rd order Tensor)
N		ND Array


1D TENSOR/
VECTOR
5
7
45
12
-6
3
22
1
6
3
-9
2D TENSOR /
MATRIX
-9 4 2 5 7
3 0 12 8 61
1 23 -6 45 2
22 3 -1 72 6
3D TENSOR/
CUBE
-9 4 2 5 7
3 0 12 8 61
1 23 -6 45 2
22 3 -1 72 6
4D TENSOR
VECTOR OF CUBES
5D TENSOR
MATRIX OF CUBES


(5,)
(5,5)
(5,5,5)
(3,5,5,5)

■ *Vector data*

○2D tensors of shape(samples,features)

■ *Timeseries data or sequence data*

○3D tensors of shape (samples, timesteps, features)

➢panel data (longitudinal data)

■ *Images*

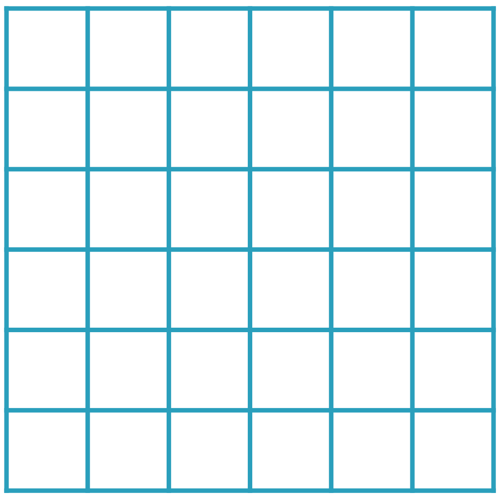
○4D tensors of shape (samples,height,width,channels) or (samples, channels, height, width)

■ *Video*

○5D tensors of shape (samples, frames, height, width, channels) or (samples, frames, channels, height, width)

Images as Matrices

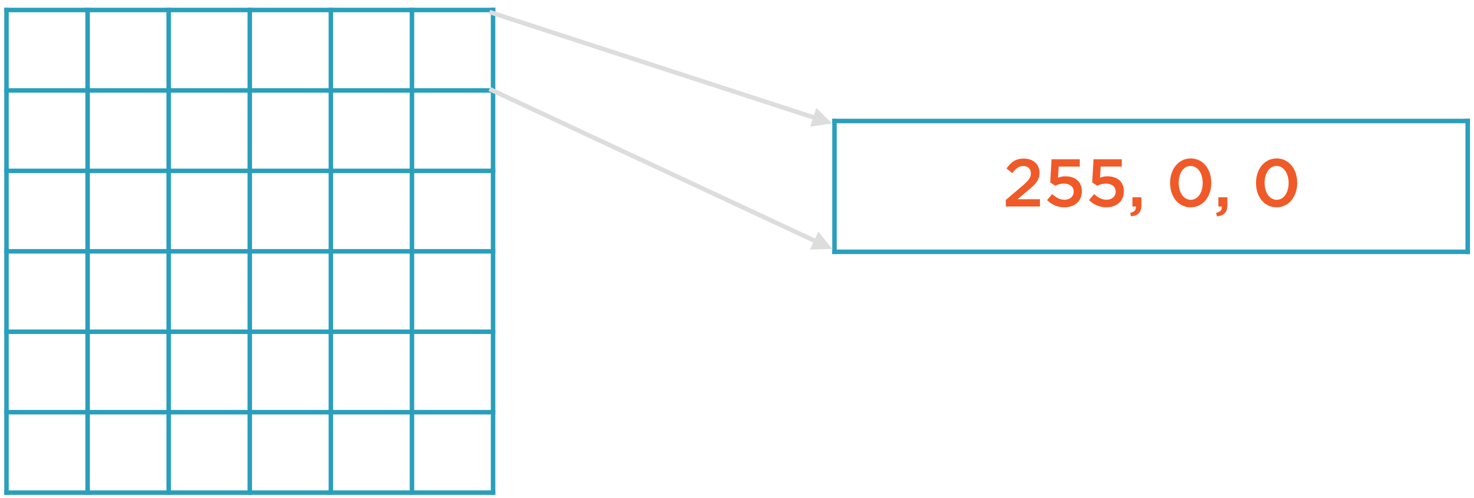
RGB Images



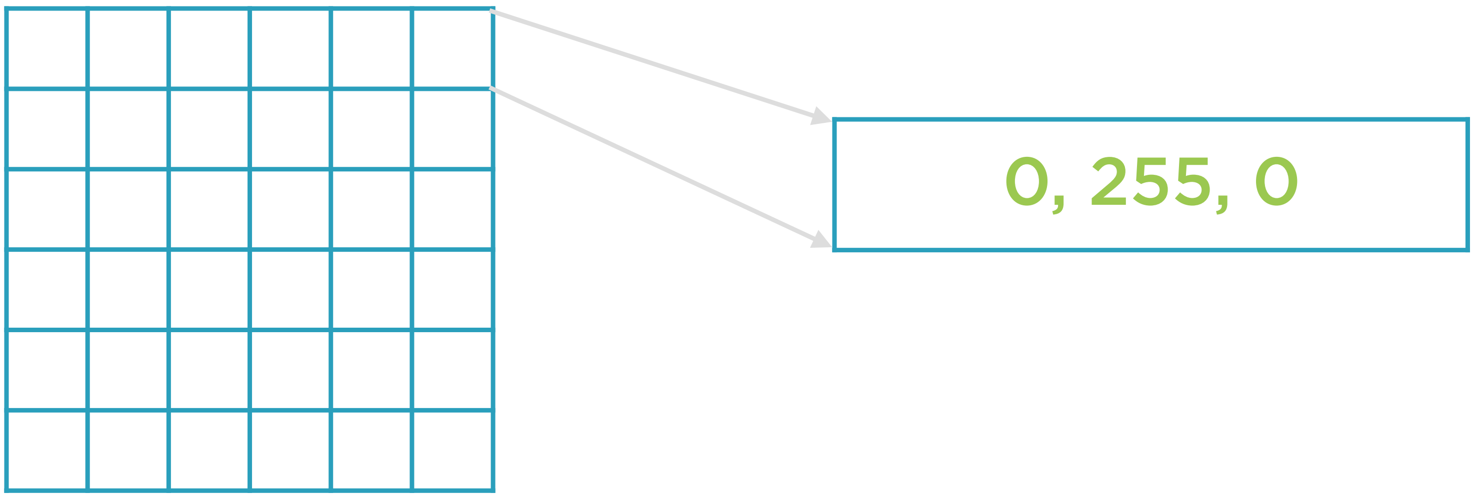
RGB values are for color images

R, G, B: 0-255

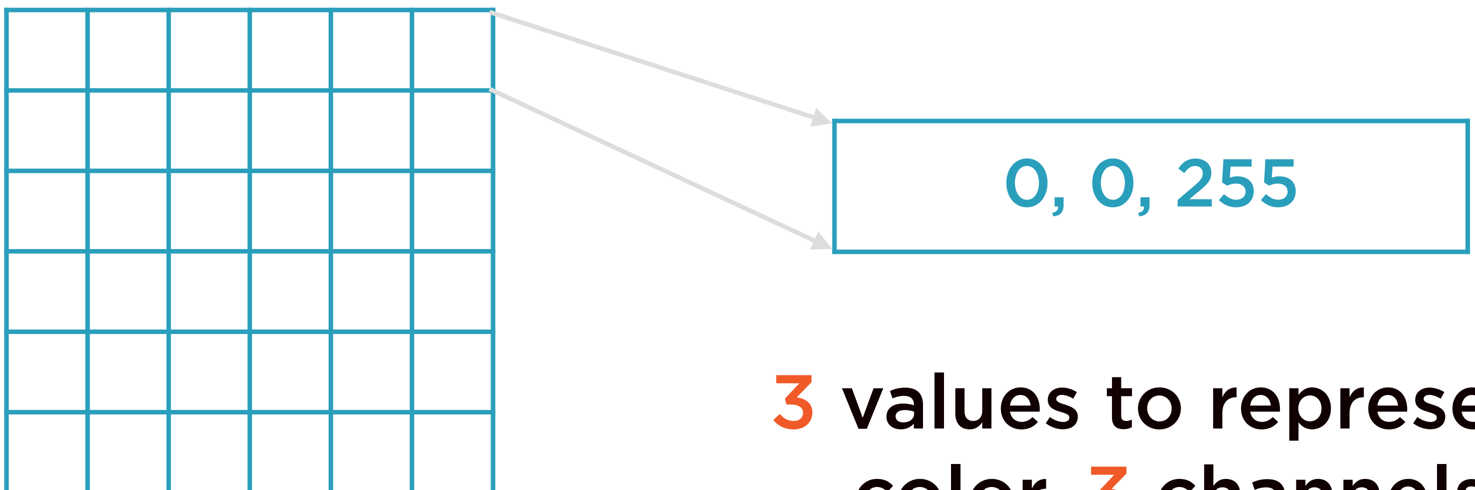
RGB Images



RGB Images

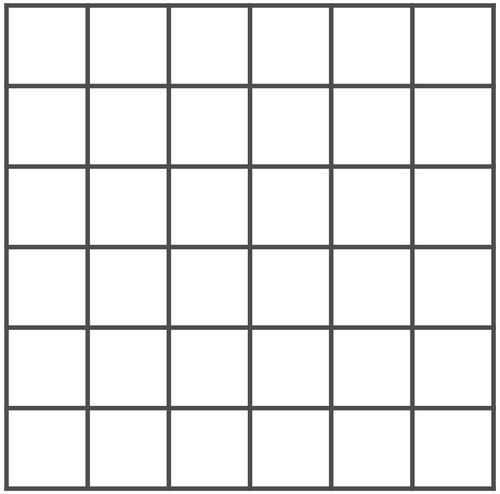


RGB Images

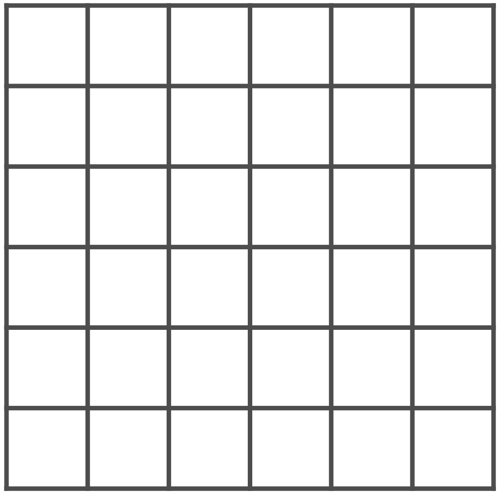


3 values to represent color, 3 channels

Grayscale Images



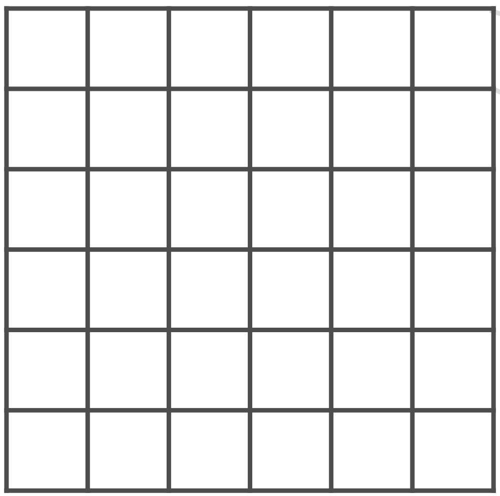
Grayscale Images



Each pixel represents only intensity information

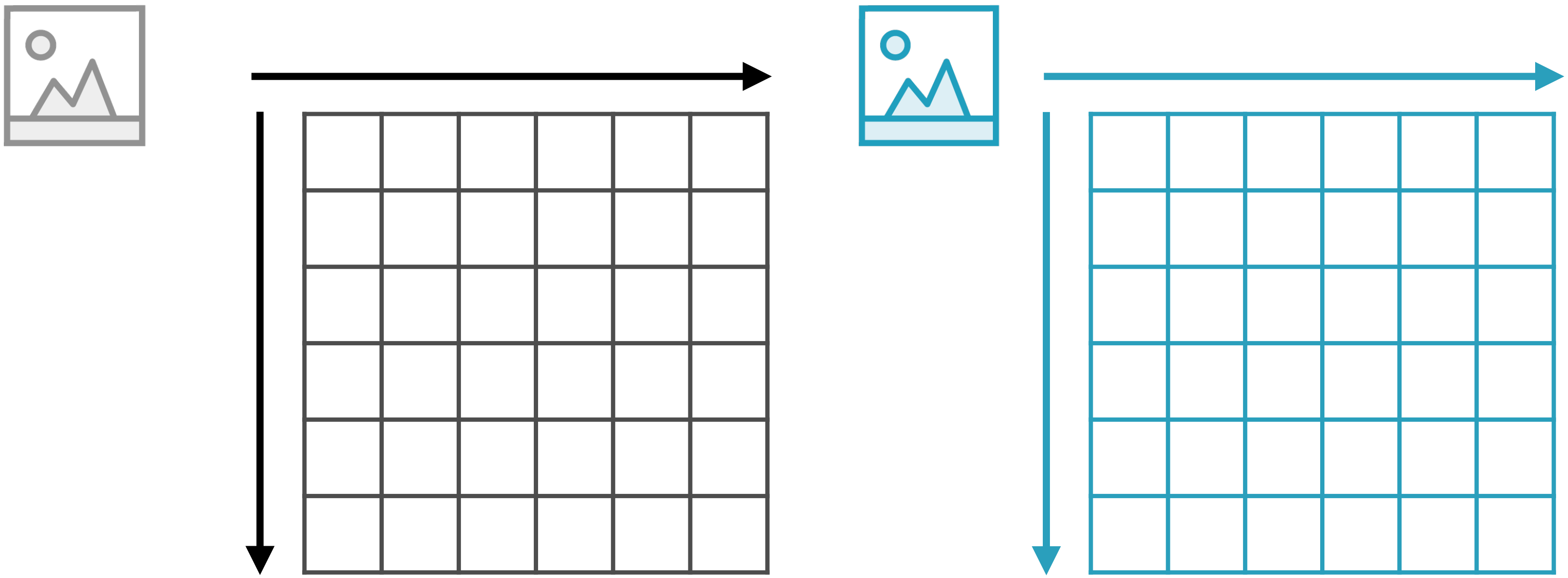
0.0 - 1.0

Grayscale Images



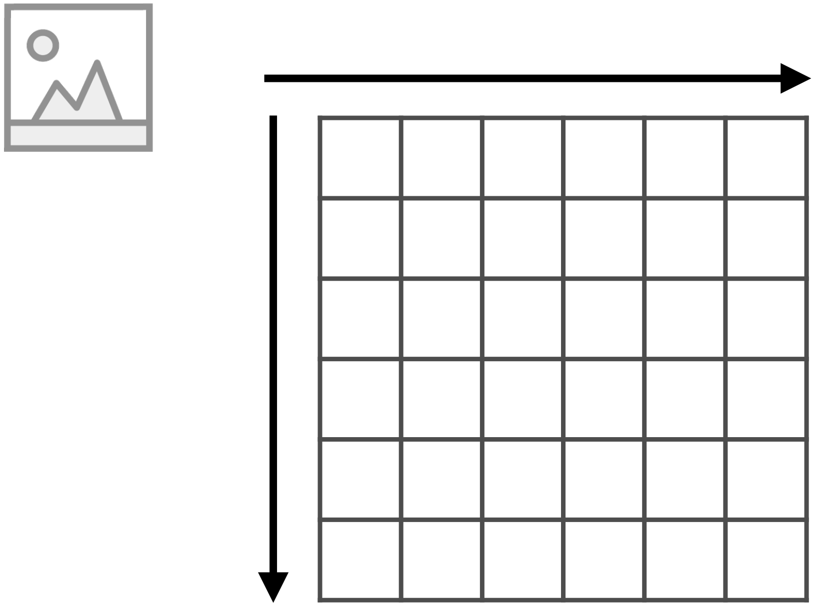
1 value to represent intensity, 1 channel

Images as Matrices

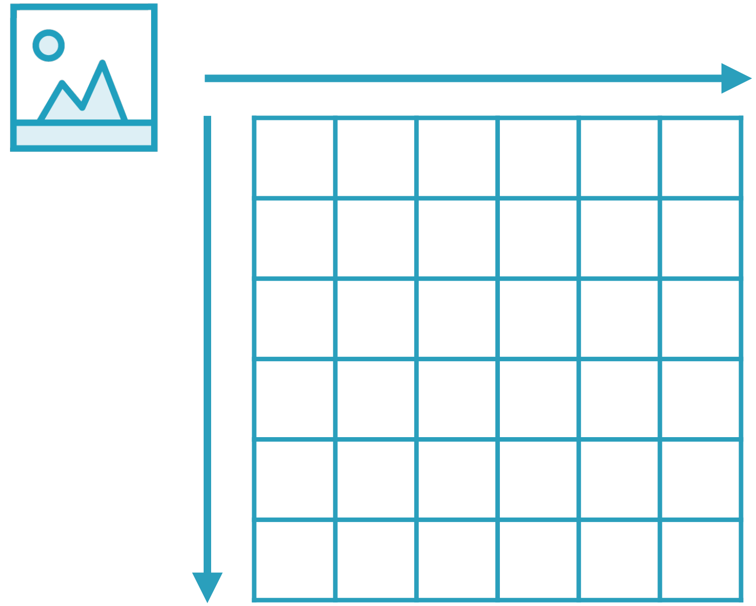


Images can be represented by a 3-D matrix

Images as Tensors



(6, 6, 1)



(6, 6, 3)

List of Images

ML frameworks (e.g. TensorFlow) usually deal with a list of images in one 4-D Tensor

List of Images

The images should all be the same size

List of Images

(10, 6, 6, $\boxed{3}$)

The number of channels

List of Images

(10, $\boxed{6, 6}$, 3)

The height and width of each image in the list

List of Images

$(\boxed{10,}\ 6,\ 6,\ 3)$

The number of images

Numpy

■ Numerical Python = NumPy

○ 벡터, 행렬 연산을 위한 수치해석용 python 라이브러리

- ➢ 빠른 수치 계산을 위한 Structured Array 및 vectorized arithmetic operations (without having to write loops) and sophisticated *broadcasting*을 통한 다차원 배열과 행렬 연산에 필요한 다양한 함수를 제공
  - – Linear algebra, random number generation, and Fourier transform capabilities
  - – 메모리 버퍼에 배열 데이터를 저장하고 처리
    - » list, array 비교하면 NumPy의 ndarray 객체를 사용하면 더 많은 데이터를 더 빠르게 처리
    - » ndarray는 타입을 명시하여 원소의 배열로 데이터를 유지
    - » 다차원 데이터도 연속된 메모리 공간이 할당됨
    - » 많은 연산이 strides를 잘 활용하면 효율적으로 가능
    - » transpose는 strides를 바꾸는 것으로 거의 추가 구현이 필요치 않음
- ➢ C로 구현 (파이썬용 C라이브러리)
- ➢ BLAS/LAPACK 기반

○ 많은 과학 계산 라이브러리가 NumPy를 기반으로 둠

- ➢ scipy, matplotlib, pandas, scikit-learn, statsmodels, etc. • 라이브러리 간의 공통 인터페이스

○ Tools for integrating code written in C, C++, and Fortran

■ Scientific Python = SciPy

○ NumPy 기반 다양한 과학, 공학분야에 활용할 수 있는 함수 제공

statsmodel
Estimate statistical models, and perform tests

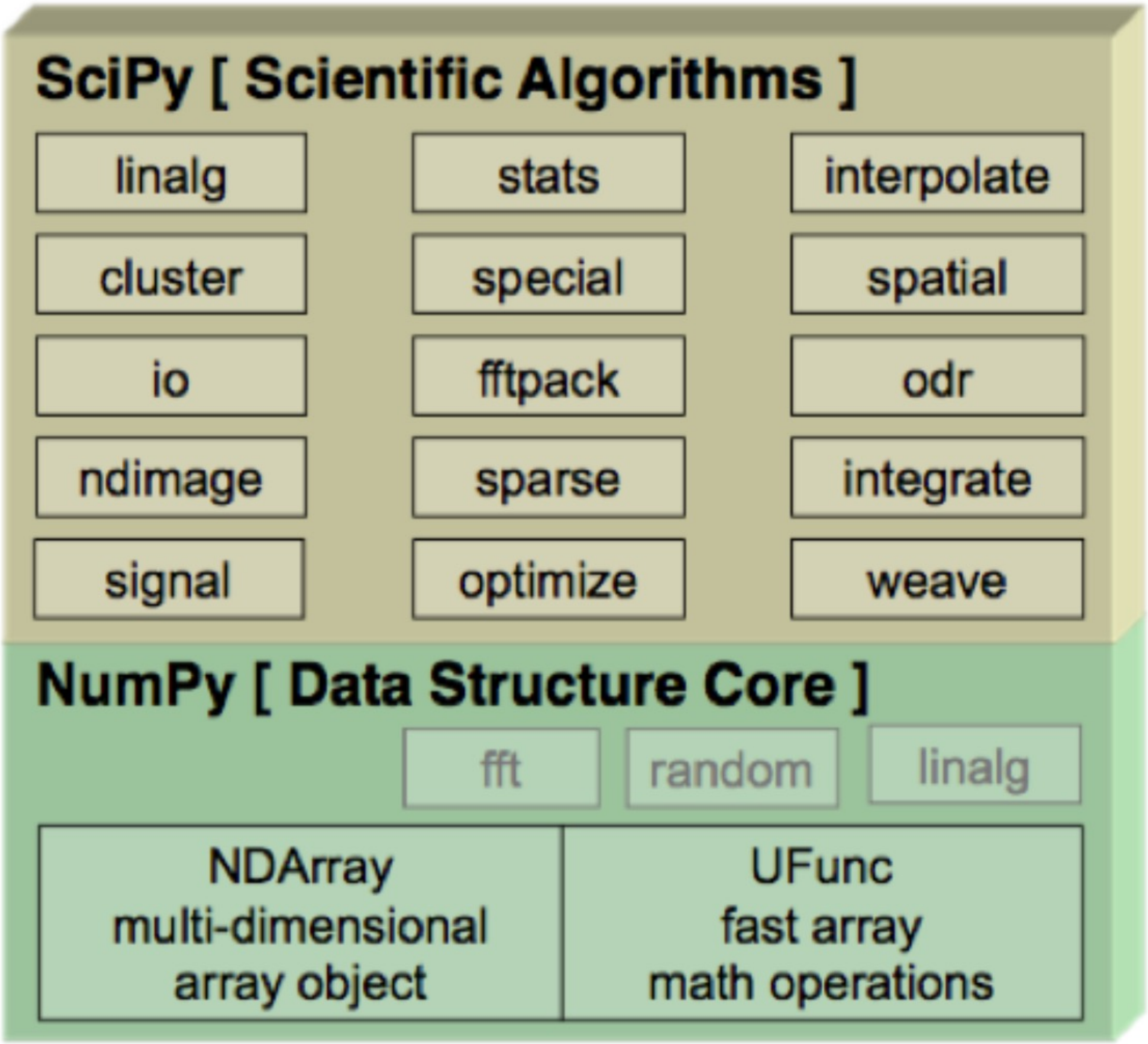
scikit-image
Collection of algorithms for image processing

scikit-learn
Simple and efficient tools for machine learning in Python

pandas
Data analysis and manipulation

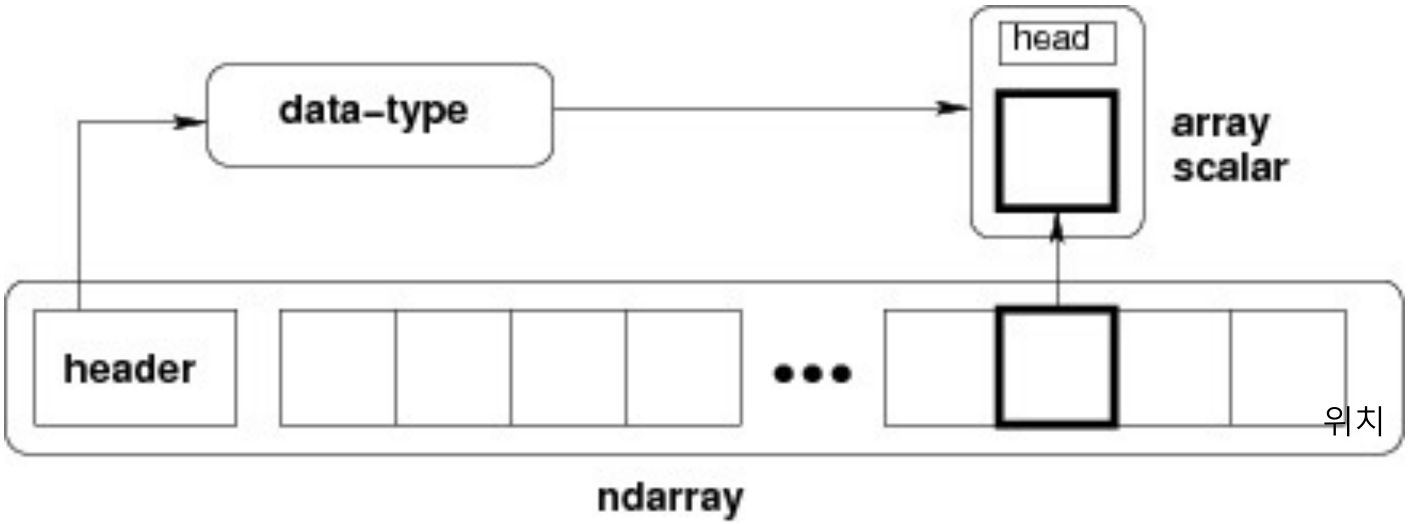
matplotlib
Plotting library for 2D graphs and visualizations

NumPy

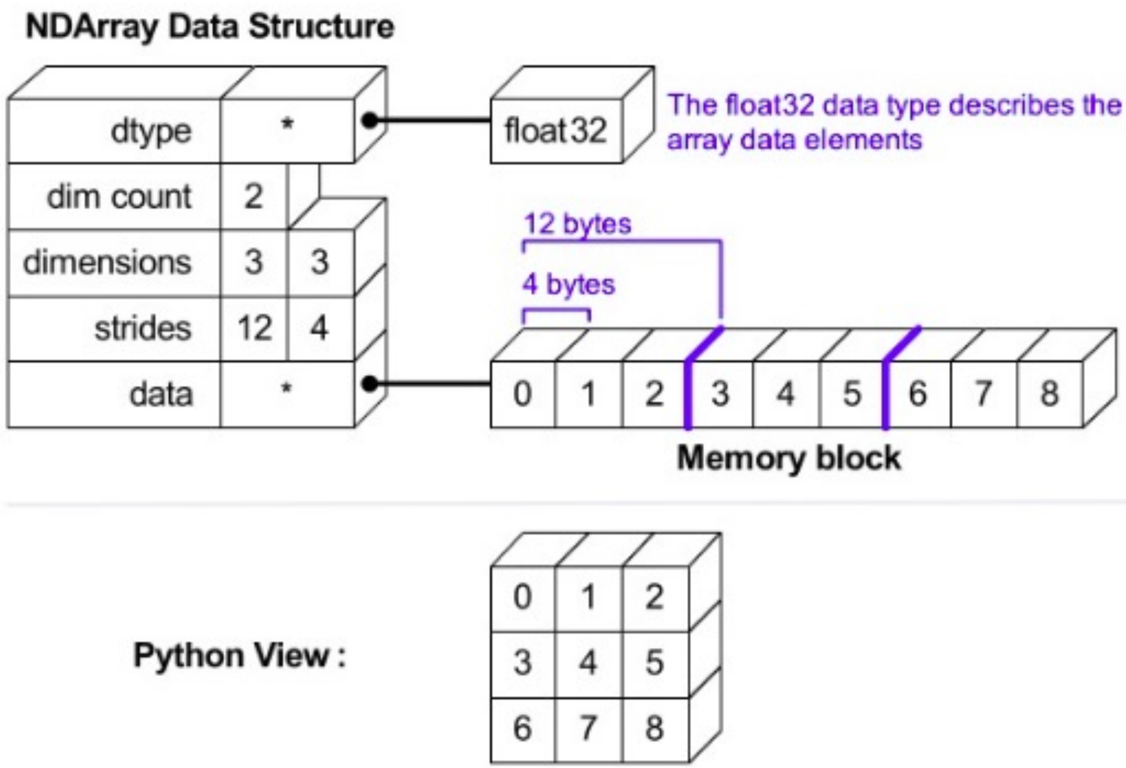

SciPy [Scientific Algorithms]
linalg
stats
interpolate
cluster
special
spatial
io
fftpack
odr
ndimage
sparse
integrate
signal
optimize
weave
NumPy [Data Structure Core]
fft
random
linalg
NDArray
multi-dimensional
array object
UFunc
fast array
math operations

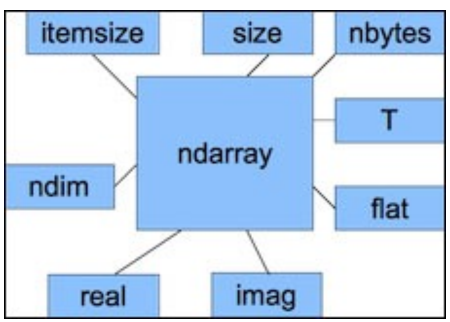
ndarray

Function	Description
ndarray	constructor
array	Convert input data (list, tuple, array, or other sequence type) to an nd array either by inferring a dtype or explicitly specifying a dtype. Copi es the input data by default.
asarray	Convert input to ndarray, but do not copy if the input is already an n darray
arange	Like the built-in range but returns an ndarray instead of a list.
ones, ones_like	Produce an array of all 1's with the given shape and dtype. ones_lik e takes another array and produces a ones array of the same shape a nd dtype.
zeros, zeros_like	Like ones and ones_like but producing arrays of 0's instead
empty, empty_like	Create new arrays by allocating new memory, but do not populate wi th any values like ones and zeros
eye, identity	Create a square N x N identity matrix (1's on the diagonal and 0's els ewhere)
linspace logspace	linspace(start, stop, num=50, endpoint=True, retstep=False)



•*memory block*: may be shared, .base, .data
•*data type descriptor*: structured data, sub-arrays, byte order, castin
g, viewing, .astype(), .view()
•*strided indexing*: strides, C/F-order, slicing w/ integers, as_strided,
broadcasting, stride tricks, diag, CPU cache coherence


NDArray Data Structure
dtype
*
float32
The float32 data type describes the array data elements
dim count
2
dimensions
3
3
strides
12
4
data
*
12 bytes
4 bytes
0 1 2 3 4 5 6 7 8
Memory block
Python View :
0 1 2
3 4 5
6 7 8


itemsize
size
nbytes
T
ndarray
ndim
flat
real
imag

Data Types

- Boolean
- Integer
- Unsigned Integer
- Float
- Complex
- String

Assign/Check/Convert of NumPy Data Types

- 데이터 형태 지정하기 (assigning Data Type)
  : np.array([xx, xx], dtype=np.Type)
- 데이터 형태 확인하기 (checking DataType)
  : object.dtype
- 데이터 형태 변환하기 (converting Data Type)
  : object.astype(np.Type)

Type	Type Code	Description
int8, uint8	i1, u1	Signed and unsigned 8-bit (1 byte) integer types
int16, uint16	i2, u2	Signed and unsigned 16-bit integer types
int32, uint32	i4, u4	Signed and unsigned 32-bit integer types
int64, uint64	i8, u8	Signed and unsigned 32-bit integer types
float16	f2	Half-precision floating point
float32	f4 or f	Standard single-precision floating point. Compatible with C float
float64, float128	f8 or d	Standard double-precision floating point. Compatible with C double and Python floatobject
float128	f16 or g	Extended-precision floating point
complex64, complex128, complex256	c8, c16, c32	Complex numbers represented by two 32, 64, or 128 floats, respectively
bool	?	Boolean type storing True and False values
object	O	Python object type
string_	S	Fixed-length string type (1 byte per character). For example, to create a string dtype with length 10, use 'S10'.
unicode_	U	Fixed-length unicode type (number of bytes platform specific). Same specification semantics as string_ (e.g. 'U10').

np.sctypeDict.keys()

Common data type name	NumPy/pandas object	Pandas string name	Notes
Boolean	`np.bool`	*bool*	Stored as a single byte.
Integer	`np.int`	*int*	Defaulted to 64 bits. Unsigned ints are also available - `np.uint`.
Float	`np.float`	*float*	Defaulted to 64 bits.
Complex	`np.complex`	*complex*	Rarely seen in data analysis.
Object	`np.object`	*O, object*	Typically strings but is a catch-all for columns with multiple different types or other Python objects (tuples, lists, dicts, and so on).
Datetime	`np.datetime64`, `pd.Timestamp`	*datetime64*	Specific moment in time with nanosecond precision.
Timedelta	`np.timedelta64`, `pd.Timedelta`	*timedelta64*	An amount of time, from days to nanoseconds.
Categorical	`pd.Categorical`	*category*	Specific only to pandas. Useful for object columns with relatively few unique values.

Comparison on using float32 instead of float64:

- Half the size in memory and on disk
- Half the memory bandwidth required (may be a bit faster in some operatio ns)

```
In [1]: a = np.zeros((1e6,), dtype=np.float64)
In [2]: b = np.zeros((1e6,), dtype=np.float32)
In [3]: %timeit a*a
1000 loops, best of 3: 1.78 ms per loop
In [4]: %timeit b*b
1000 loops, best of 3: 1.07 ms per loop
```

But: bigger rounding errors — sometimes in surprising places (i.e., don’t use them unless you really need them)

Array siblings

1. chararray
2. maskedarray
3. matrix

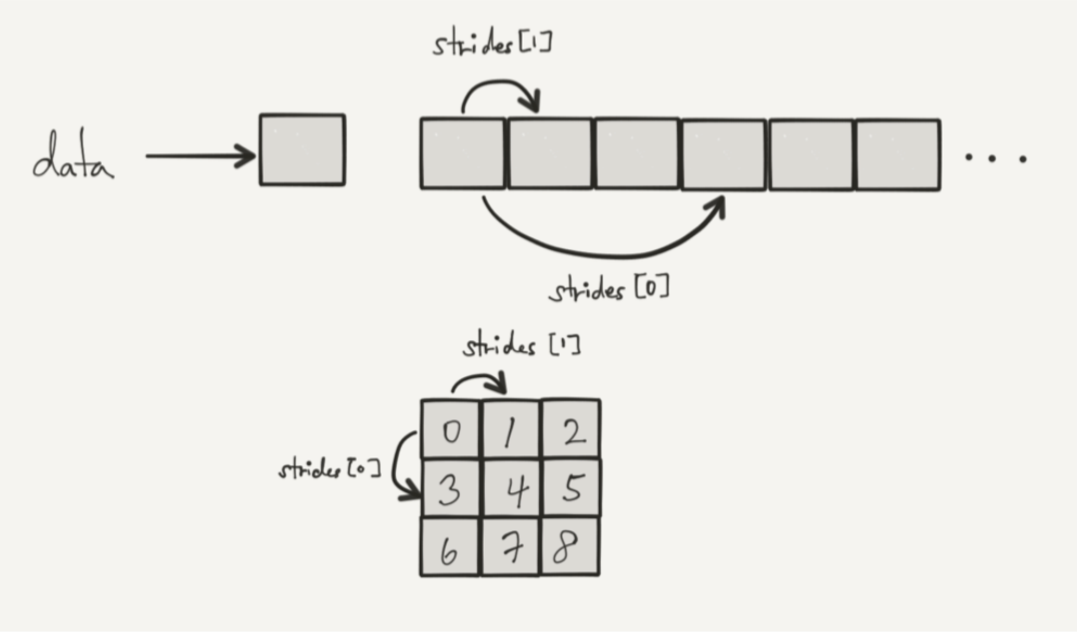
= native

<: little-endian	리틀 엔디안은 최하위 비트(LSB)부터 부호화되어 저장된다. 예를 들면, 숫자 12는 2진수로 나타내면 1100인데 리틀 엔디안은 0011로 각각 저장된다. 좌측부터 저장
>: big-endian	이 방식은 데이터의 최상위 비트가 가장 높은 주소에 저장되므로 그냥 보기에는 역으로 보인다. 빅 엔디안은 최상위 비트(MSB)부터 부호화되어 저장되며 예를 들면, 숫자 12는 2진수로 나타내면 1100인데 빅 엔디안은 1100으로 저장된다. 우측부터 저장
|: not-relevant	문자를 저장할 때 사용 endian가 상관없이 처리 .

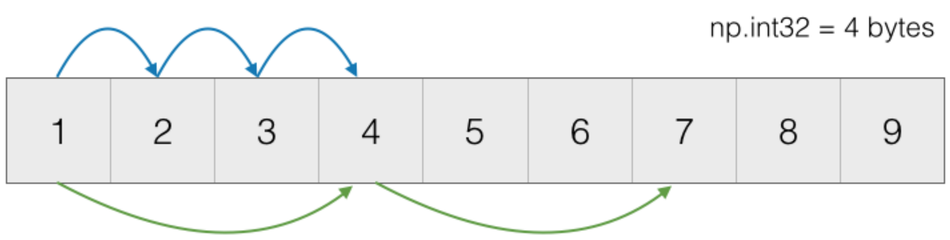
itemsize : 저장되는 메모리의 크기

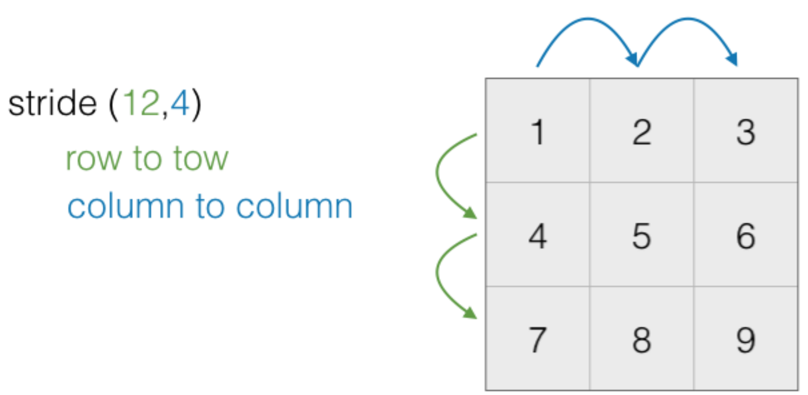
numpy.lib.stride_tricks

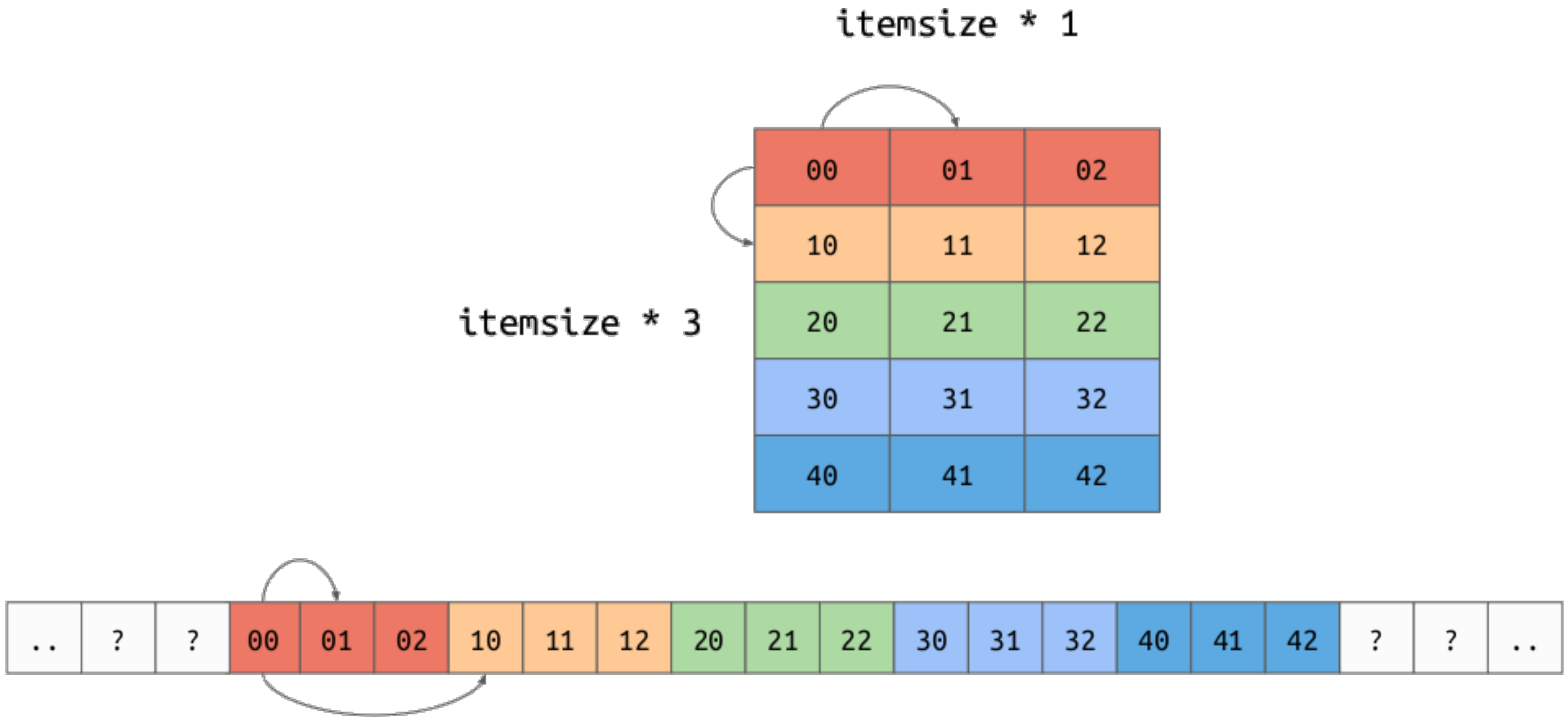
```
data : 4297514880
shape : (3, 3)
strides : (6, 2)
dtype : uint16
```

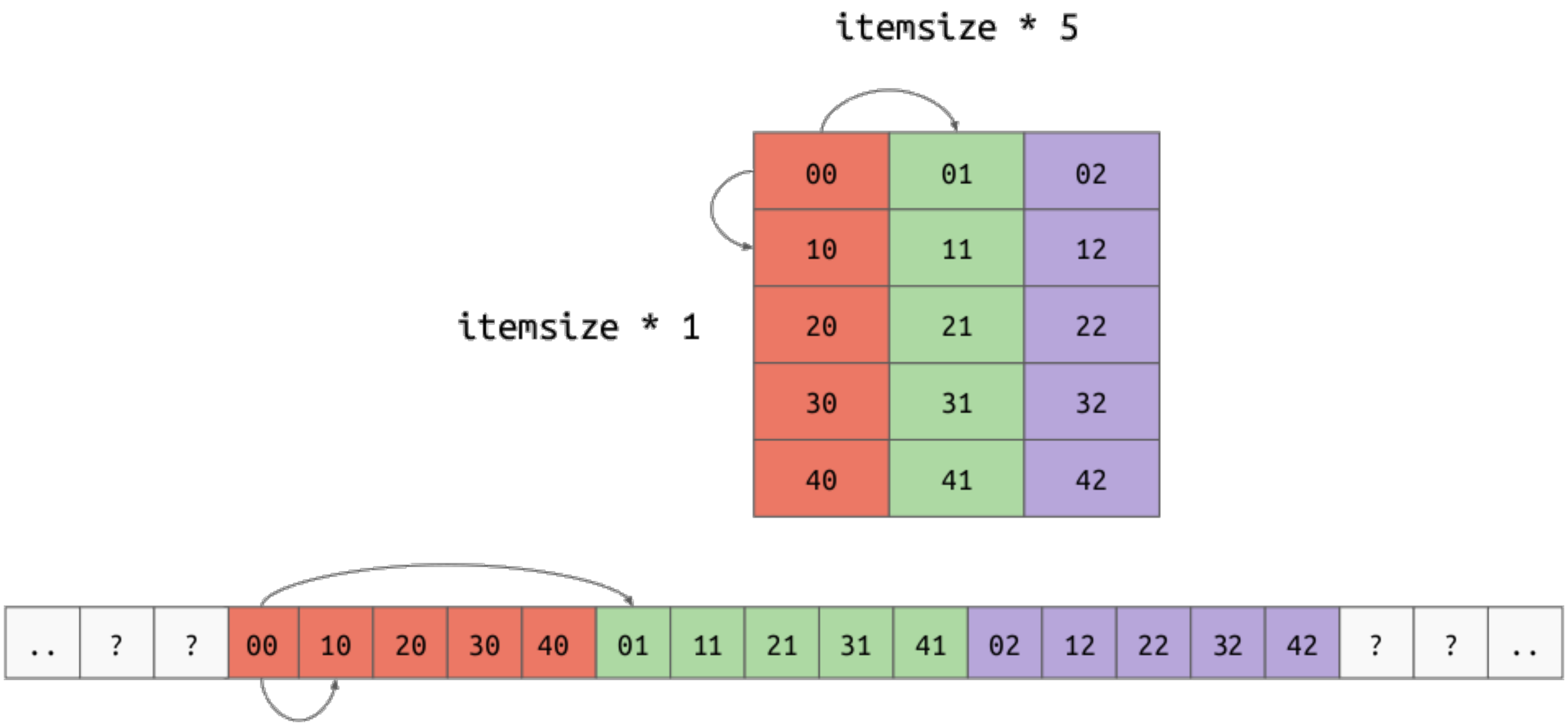


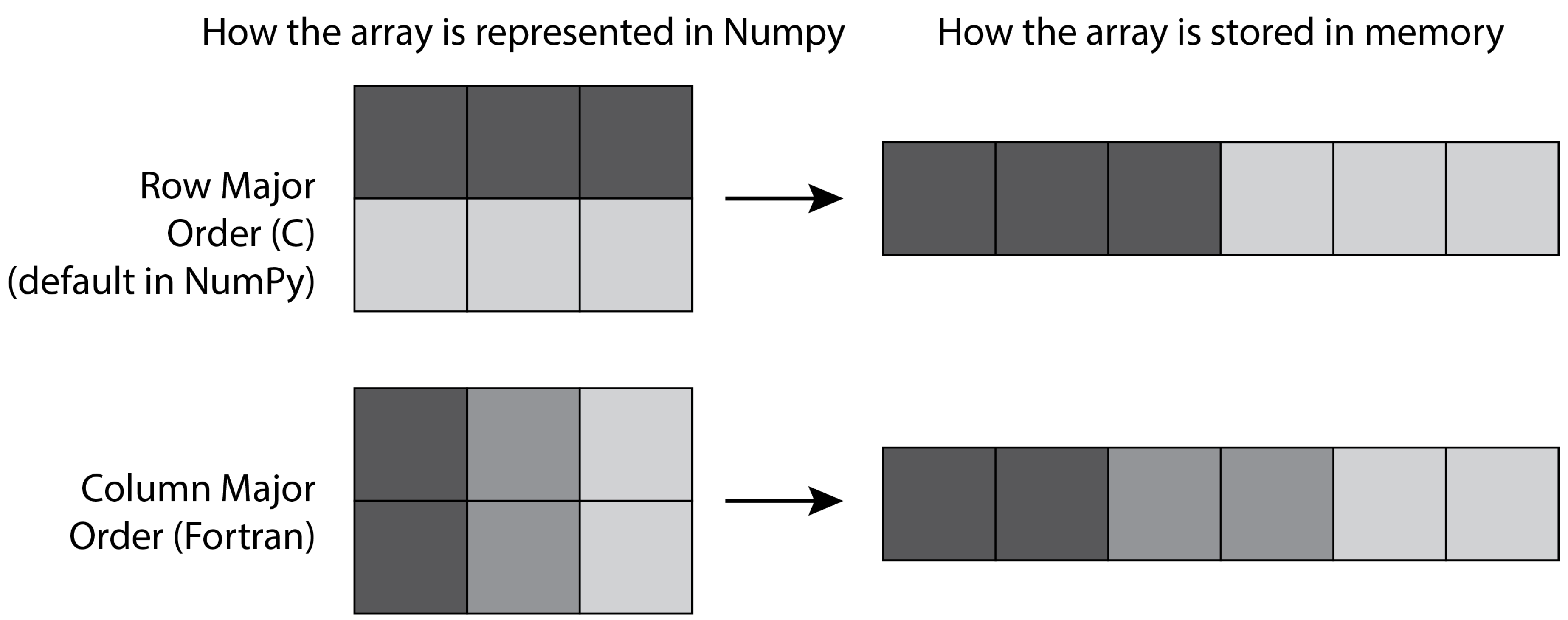

stride (8,)
np.int64 = 8 bytes
1
2
3
4
5
6
7
8
9
ctypes.data

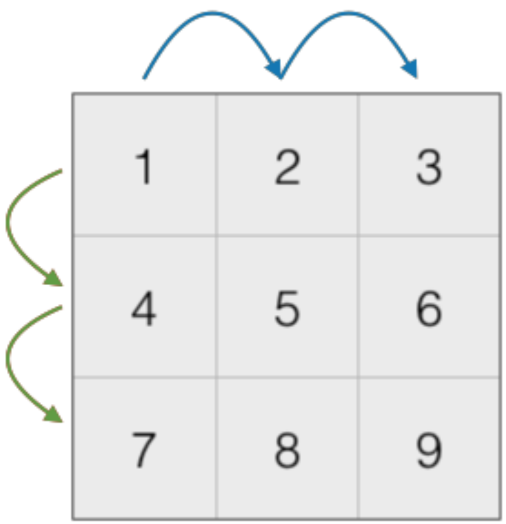

np.int32 = 4 bytes
1
2
3
4
5
6
7
8
9


stride (12,4)
row to tow
column to column
1
2
3
4
5
6
7
8
9

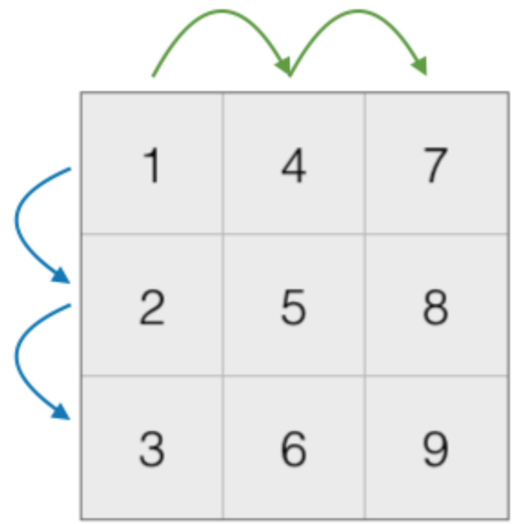

itemsize * 1
itemsize * 3
00
01
02
10
11
12
20
21
22
30
31
32
40
41
42
..
?
?
00
01
02
10
11
12
20
21
22
30
31
32
40
41
42
?
?
..


itemsize * 5
itemsize * 1
00
01
02
10
11
12
20
21
22
30
31
32
40
41
42
..
?
?
00
10
20
30
40
01
11
21
31
41
02
12
22
32
42
?
?
..


How the array is represented in Numpy
How the array is stored in memory
Row Major
Order (C)
(default in NumPy)
Column Major
Order (Fortran)



stride (12,4)



stride (4,12)

C

fortran

```
a=np.arange(30).reshape(10,3)
  =array([[ 0,  1,  2],
          [ 3,  4,  5],
          [ 6,  7,  8],
          [ 9, 10, 11],
axis=1    [12, 13, 14],
          [15, 16, 17],
          [18, 19, 20],
          [21, 22, 23],
          [24, 25, 26],
          [27, 28, 29]])   axis=0
```

(10,3)
axis=1

np.mean(a, axis=1)
=array([1., 4., 7., 10., 13.,
16., 19., 22., 25., 28.])

np.mean(a)=14.5

np.mean(a, axis=0)
=array([13.5, 14.5, 15.5])

(10,3)
axis=0

```
np.mean(np.arange(100).
        reshape(10,2,5))
=49.5
```

```
a=np.arange(100).reshape(10,2,5)
 =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, 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, 61, 62, 63, 64],
          [65, 66, 67, 68, 69]],
         [[70, 71, 72, 73, 74],
          [75, 76, 77, 78, 79]],
         [[80, 81, 82, 83, 84],
          [85, 86, 87, 88, 89]],
         [[90, 91, 92, 93, 94],
          [95, 96, 97, 98, 99]]])
```

axis=2

axis=1

axis=0

```
np.mean(np.arange(100).
        reshape(10,2,5), axis=2)
=array([[  2.,   7.],
        [ 12.,  17.],
        [ 22.,  27.],
        [ 32.,  37.],
        [ 42.,  47.],
        [ 52.,  57.],
        [ 62.,  67.],
        [ 72.,  77.],
        [ 82.,  87.],
        [ 92.,  97.]])
```

(10,2,5)
axis=2

```
np.mean(np.arange(100).
        reshape(10,2,5), axis=1)
=array([[  2.5,   3.5,   4.5,   5.5,   6.5],
        [ 12.5,  13.5,  14.5,  15.5,  16.5],
        [ 22.5,  23.5,  24.5,  25.5,  26.5],
        [ 32.5,  33.5,  34.5,  35.5,  36.5],
        [ 42.5,  43.5,  44.5,  45.5,  46.5],
        [ 52.5,  53.5,  54.5,  55.5,  56.5],
        [ 62.5,  63.5,  64.5,  65.5,  66.5],
        [ 72.5,  73.5,  74.5,  75.5,  76.5],
        [ 82.5,  83.5,  84.5,  85.5,  86.5],
        [ 92.5,  93.5,  94.5,  95.5,  96.5]])
```

(10,2,5)
axis=1

```
np.mean(np.arange(100).
        reshape(10,2,5), axis=0)
=array([[ 45.,  46.,  47.,  48.,  49.],
        [ 50.,  51.,  52.,  53.,  54.]])
```

(10,2,5)
axis=0

구분	ndarray	matrix
차원	다차원 가능	2 차원
* 연산자	요소간 곱	행렬곱
numpy.multiply()	요소간 곱	요소간 곱
numpy.dot()	행렬곱	행렬곱

PEP3118 @연산자
PEP465

2. indexing / slicing

np.may_share_memory(a, b)

Casting and re-interpretation/views

casting
•on assignment
•on array construction
•on arithmetic
•etc.
•and manually: .astype(dtype)

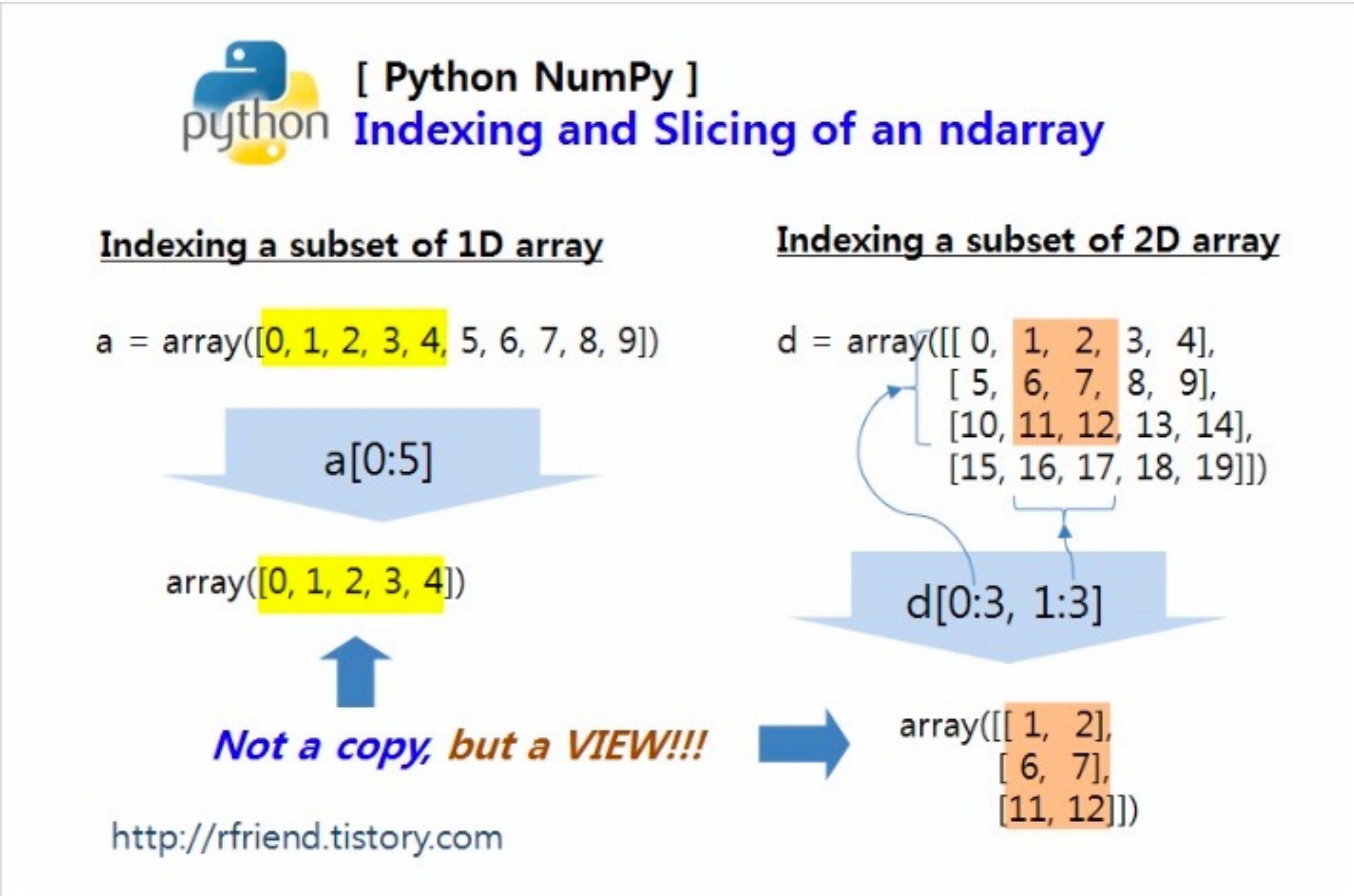
data re-interpretation
•manually: .view(dtype)

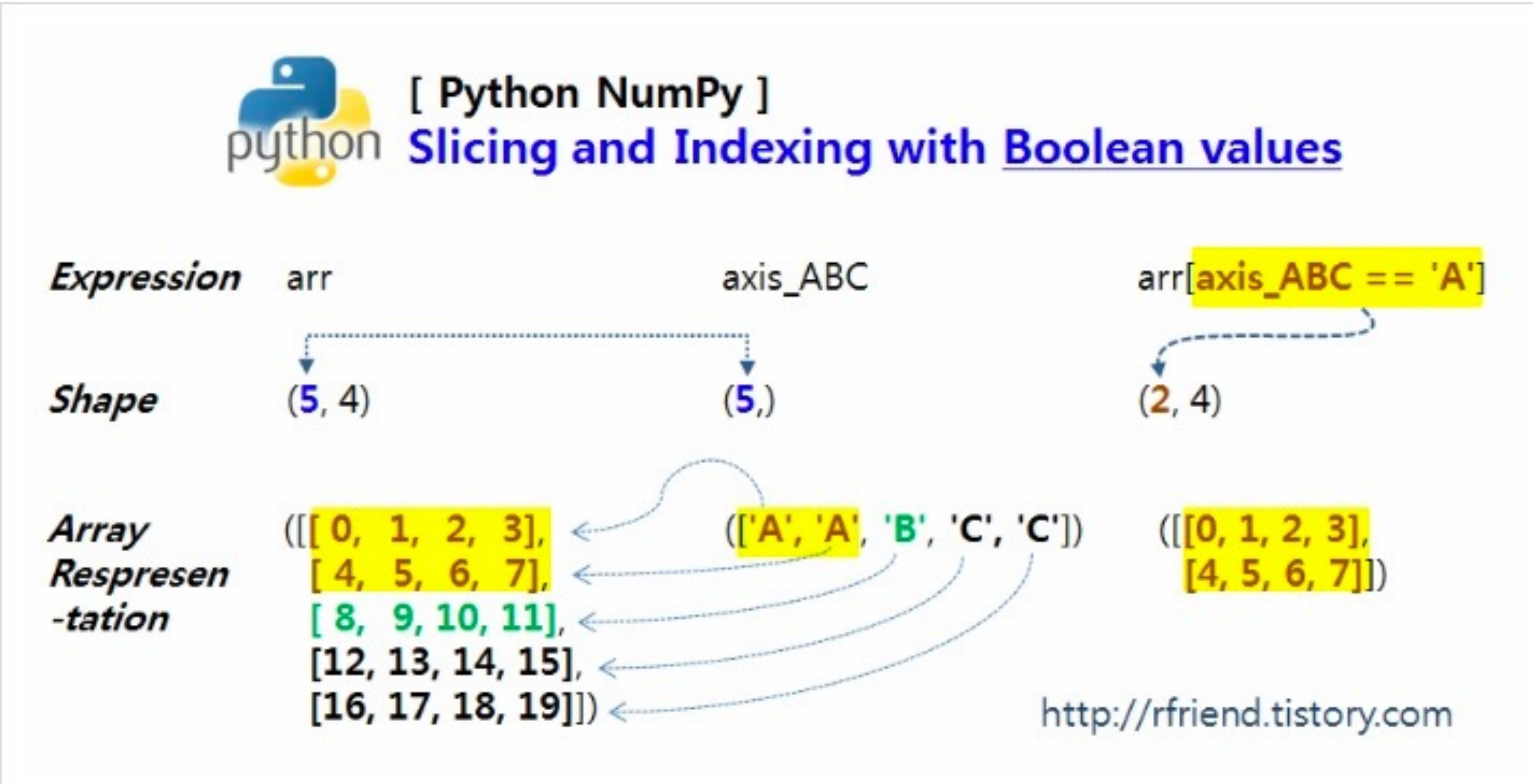
Casting

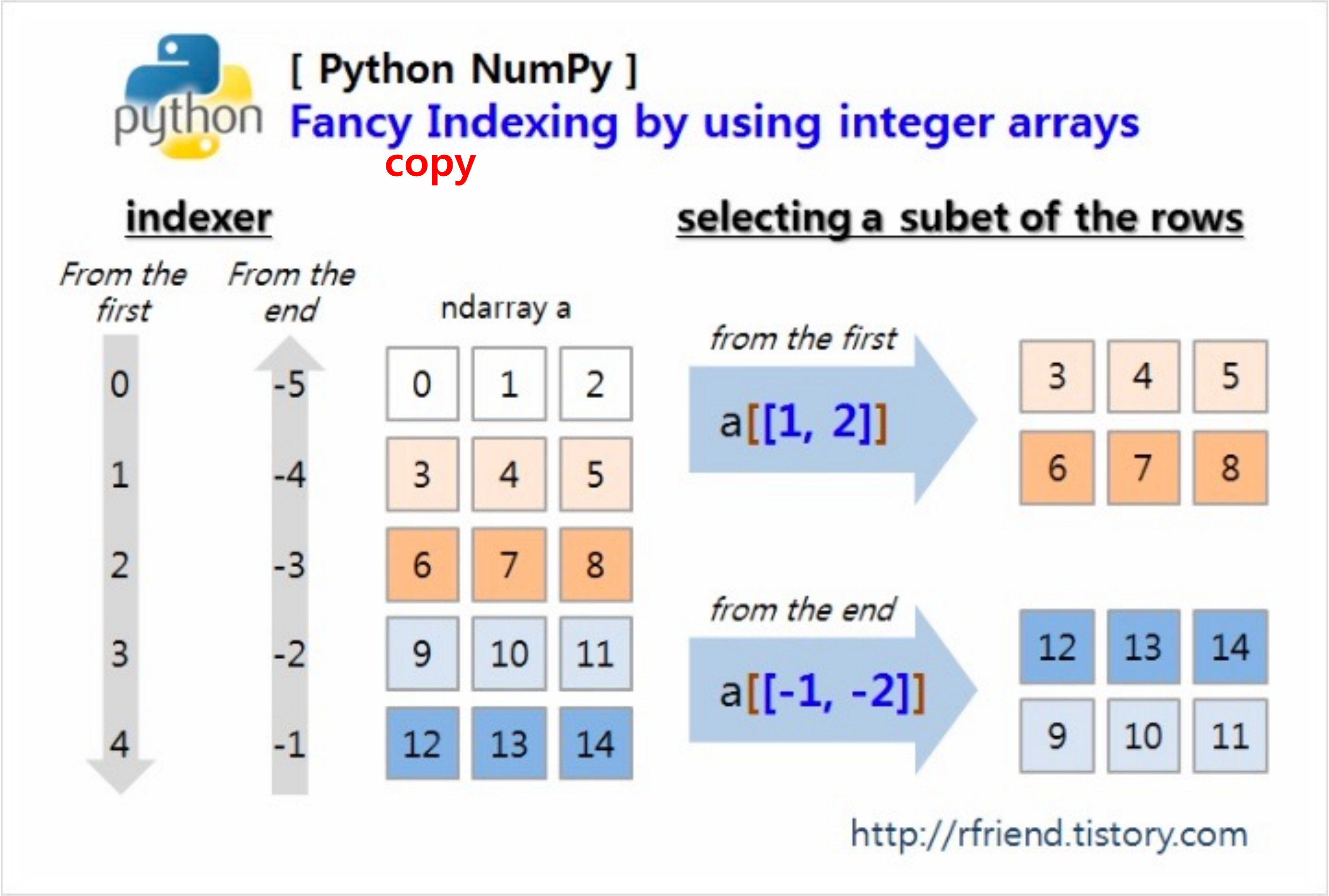
Casting in arithmetic, in nutshell:

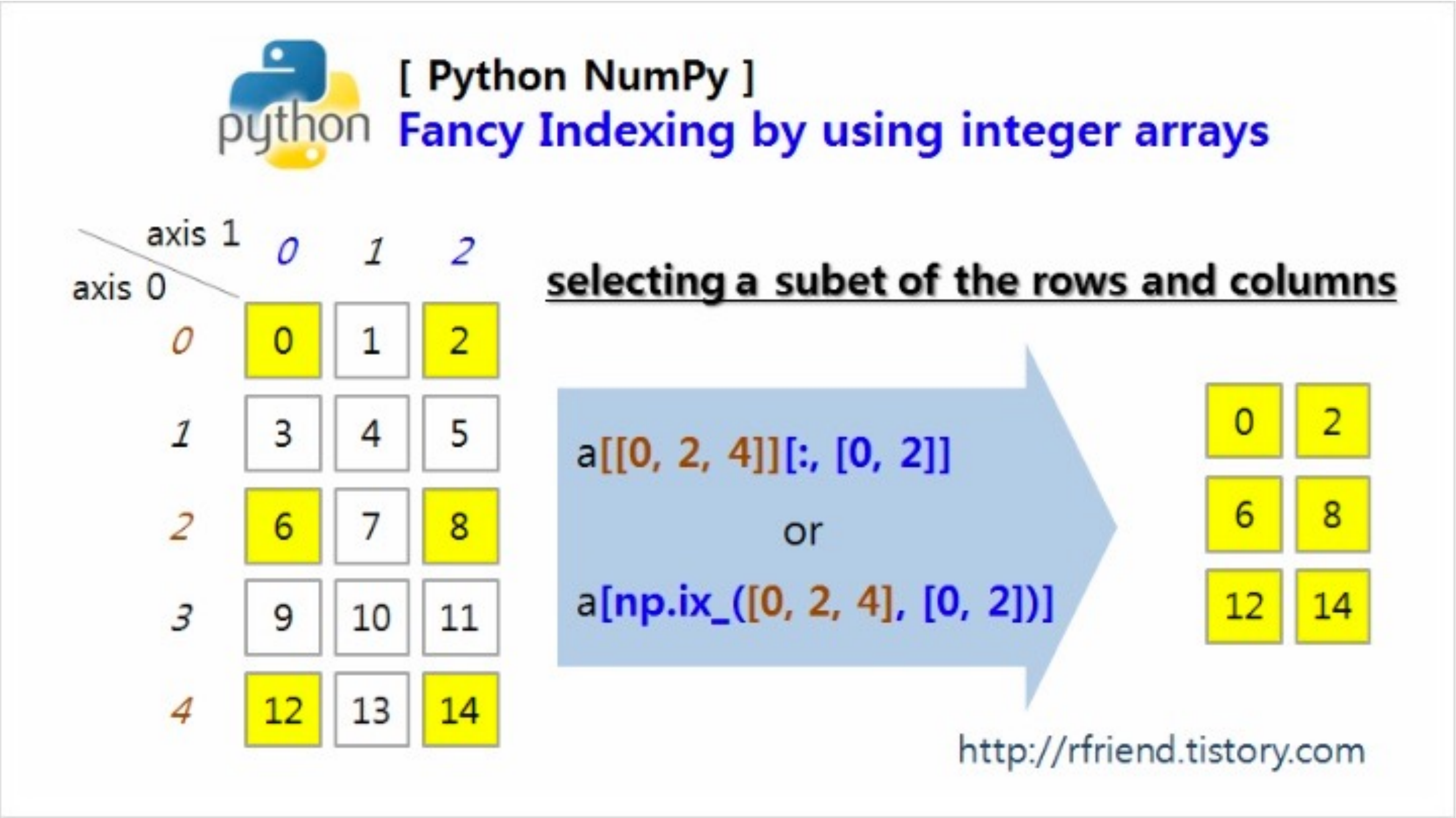
- only type (not value!) of operands matters
- largest “safe” type able to represent both is picked
- scalars can “lose” to arrays in some situations

- ,
- :
- ...
- **Fancy indexing**

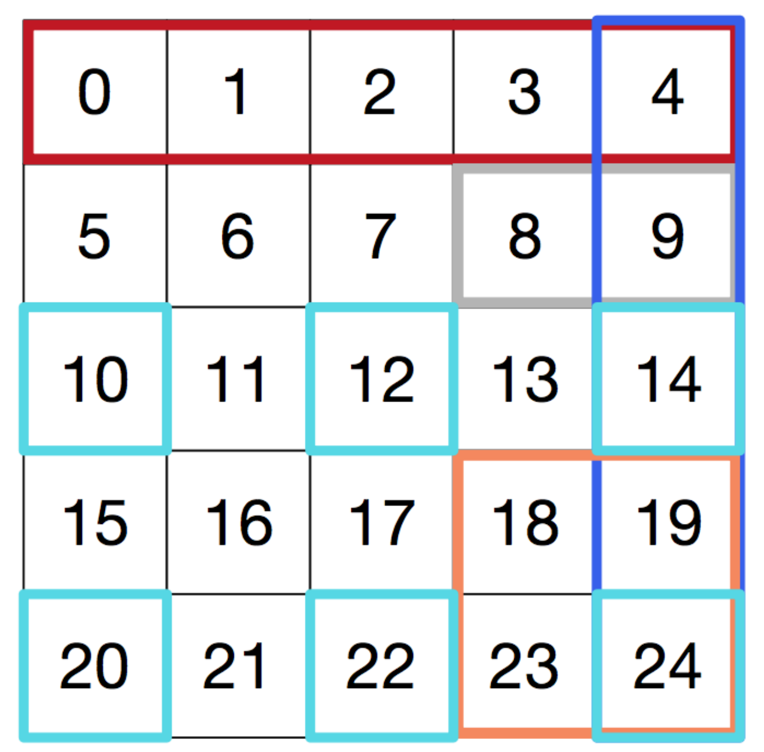

[Python NumPy]
Indexing and Slicing of an ndarray
Indexing a subset of 1D array
a = array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
a[0:5]
array([0, 1, 2, 3, 4])
Not a copy, but a VIEW!!!
http://rfriend.tistory.com
Indexing a subset of 2D array
d = array([[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9],
[10, 11, 12, 13, 14],
[15, 16, 17, 18, 19]])
d[0:3, 1:3]
array([[1, 2],
[6, 7],
[11, 12]])


[Python NumPy]
Slicing and Indexing with Boolean values
Expression
arr
axis_ABC
arr[axis_ABC == 'A']
Shape
(5, 4)
(5,)
(2, 4)
Array
Respresen
-tation
([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11],
[12, 13, 14, 15],
[16, 17, 18, 19]])
(['A', 'A', 'B', 'C', 'C'])
([[0, 1, 2, 3],
[4, 5, 6, 7]])
http://rfriend.tistory.com


[Python NumPy]
Fancy Indexing by using integer arrays
copy
indexer
selecting a subet of the rows
From the first
From the end
ndarray a
0 1 2 3 4
-5 -4 -3 -2 -1
0 1 2
3 4 5
6 7 8
9 10 11
12 13 14
from the first
a[[1, 2]]
3 4 5
6 7 8
from the end
a[[-1, -2]]
12 13 14
9 10 11
http://rfriend.tistory.com


[Python NumPy]
Fancy Indexing by using integer arrays
axis 1
axis 0
0
1
2
0
1
2
3
4
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
selecting a subet of the rows and columns
a[[0, 2, 4]][:, [0, 2]]
or
a[np.ix_([0, 2, 4], [0, 2])]
0
2
6
8
12
14
http://rfriend.tistory.com

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

>>> a[1, 3:5]
array([8, 9])

>>> a[:, 4]
array([ 4,  9, 14, 19, 24])

>>> a[-2:, -2:]
array([[18, 19],
       [23, 24]])

>>> a[2::2, ::2]
array([[10, 12, 14],
       [20, 22, 24]])
```

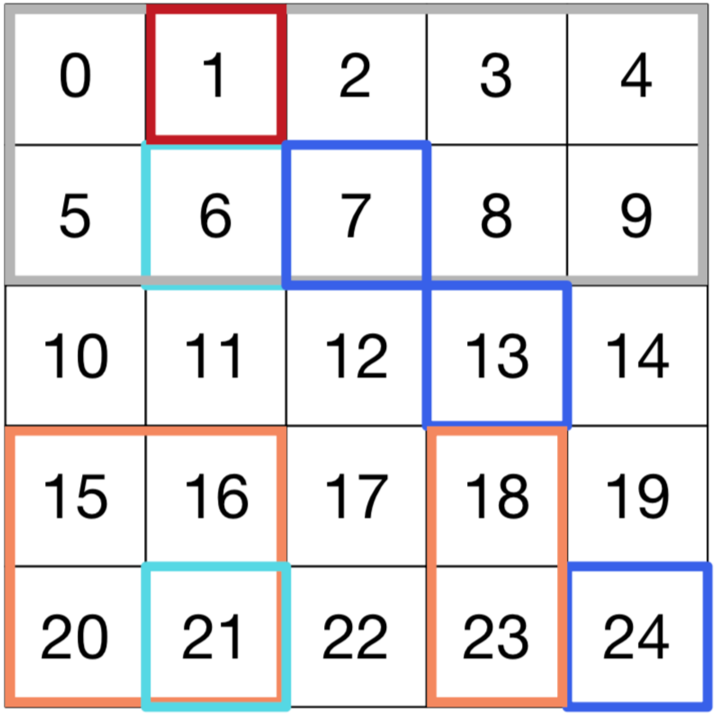


```
>>> a[0, 1]
1
```

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

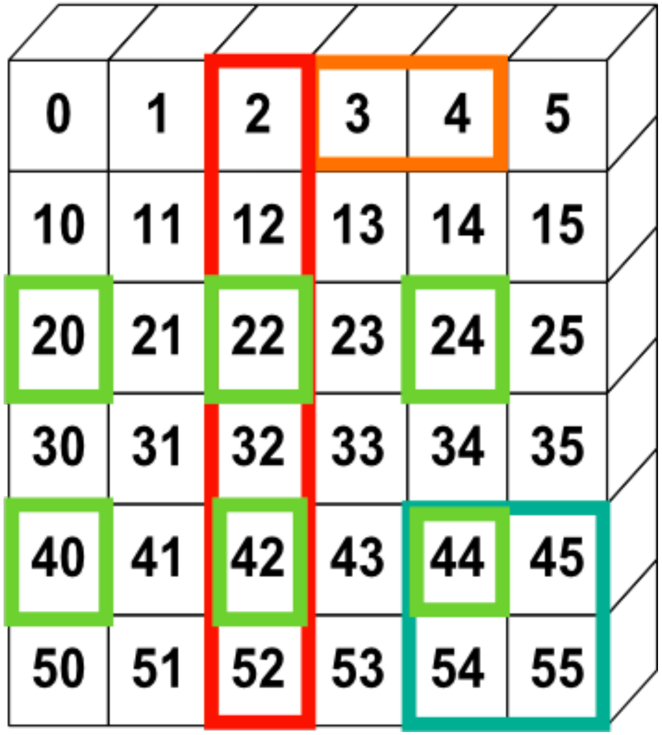
```
>>> a[[1, 2, 4], [2, 3, 4]]
array([ 7, 13, 24])
```

```
>>> a[3:, [0, 1, 3]]
array([[15, 16, 18],
       [20, 21, 23]])
```

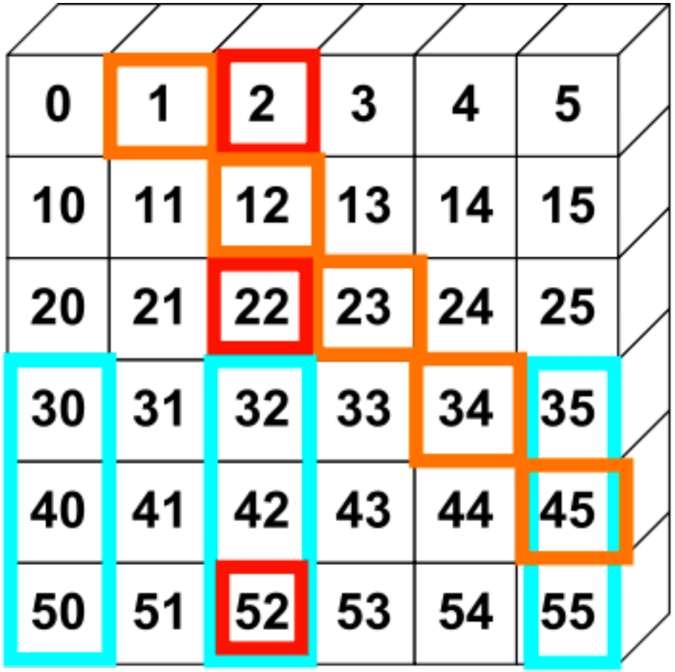



```
>>> mask = np.array([0, 1, 0, 0, 1], dtype=np.bool)
>>> a[mask, 1]
array([ 6, 21])
```

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



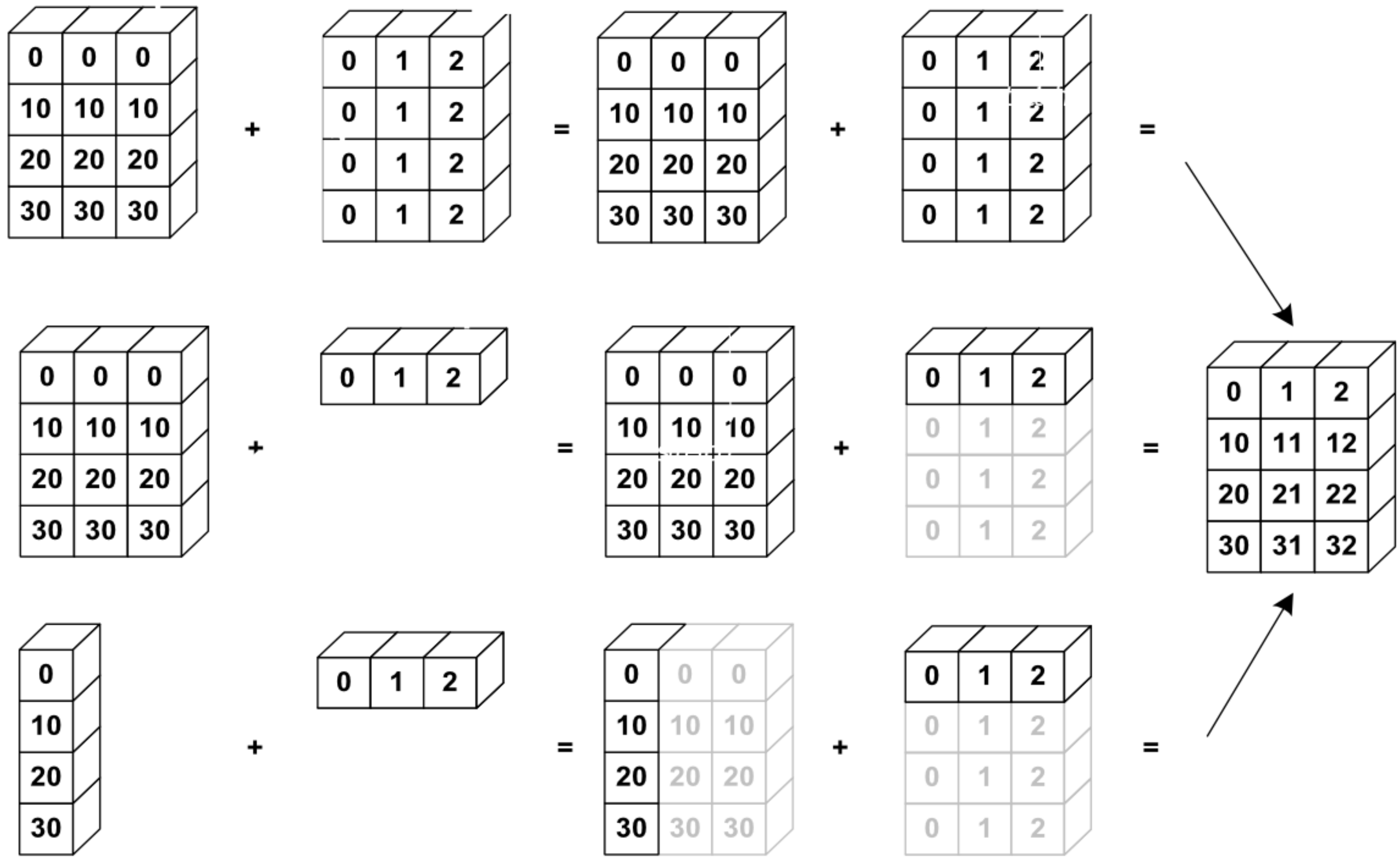
```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45]])
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```



■ 이름

https://numpy.org/doc/stable/user/basics.rec.html

3. broadcasting



0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 1 2
10 11 12
20 21 22
30 31 32
0
10
20
30

■ **기본 연산은 개별 원소마다 적용**

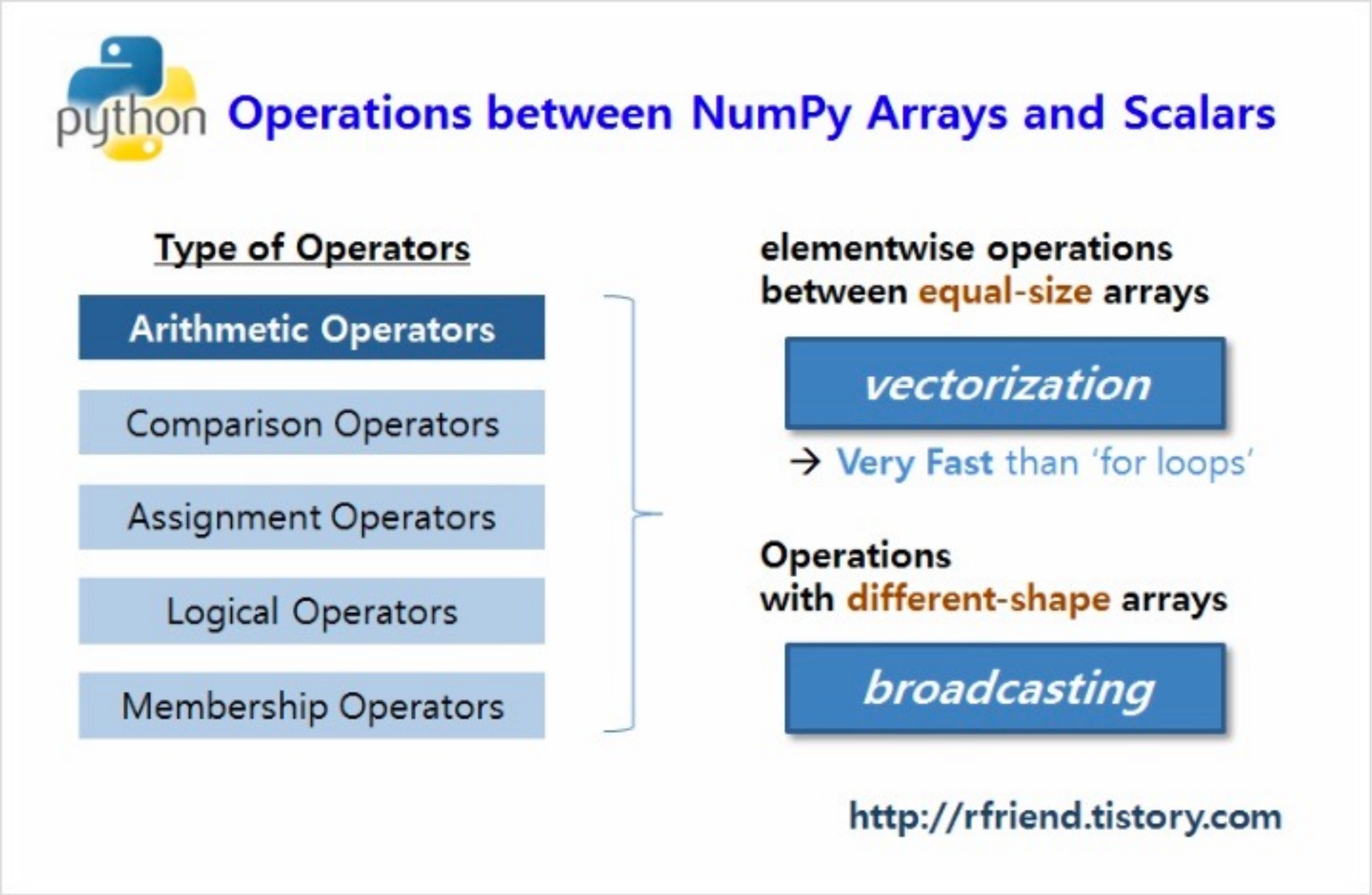
○ 그러므로 모양이 다른 배열 간의 연산이 불가능

○ 하지만 특정 조건이 만족되면 배열 변환이 자동으로 일어나서 연산 가능

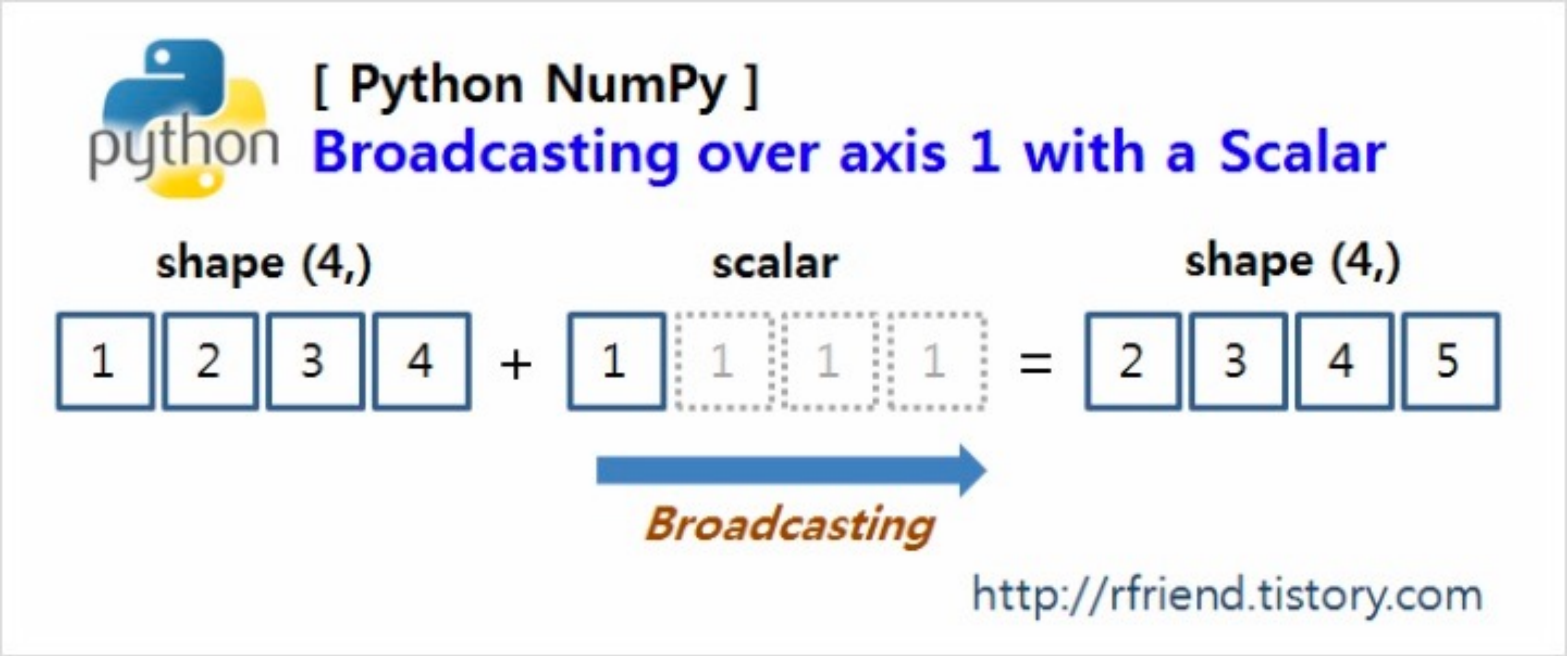
○ 이를 브로드캐스팅 (broadcasting) 이라 함

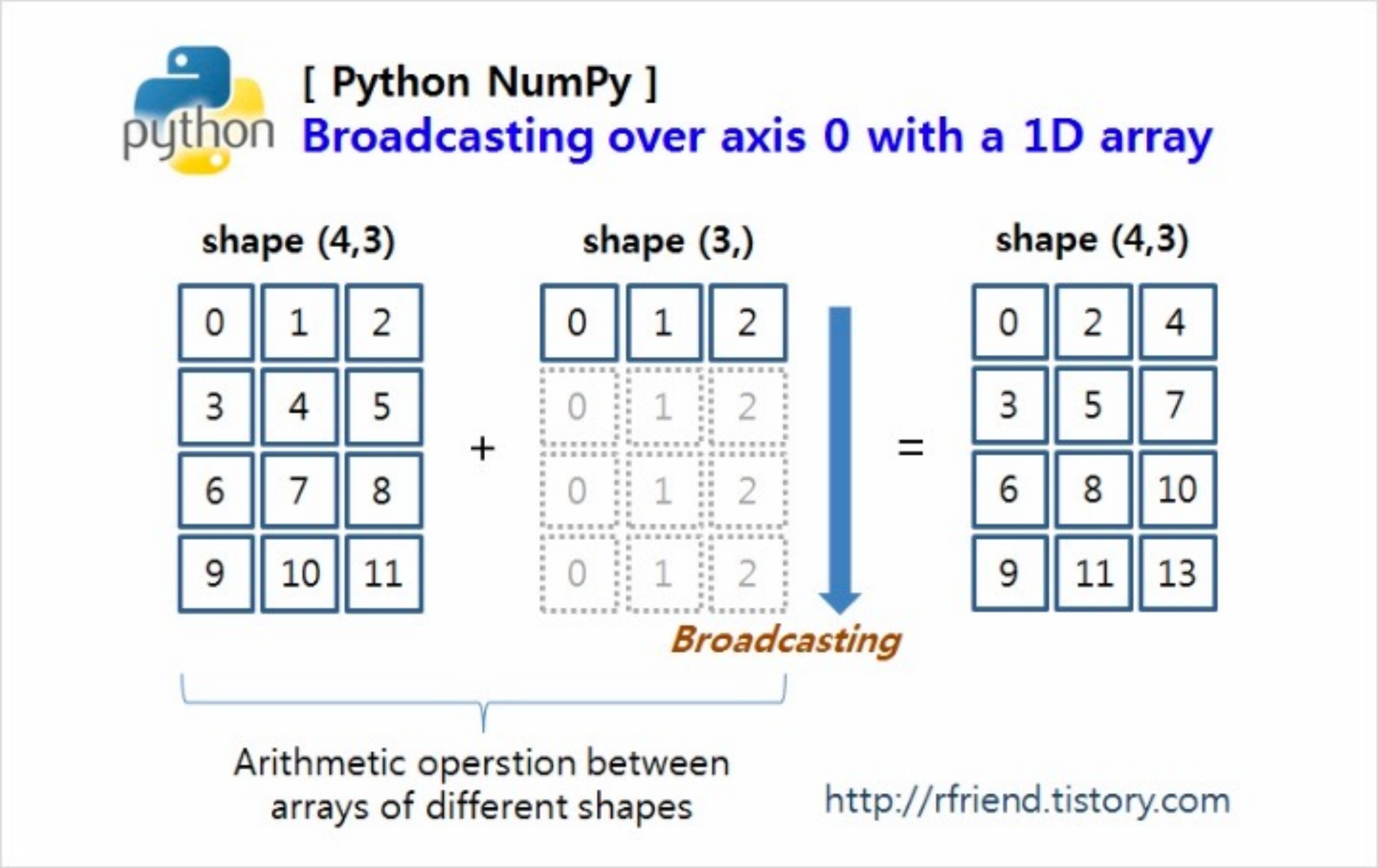
■ **두 배열을 오른쪽 정렬 I 차원 개수가 작은 배열은 왼쪽 차원을 1로 채움 I 각각 짝이 되는 차원의 크기가 같으면 연산 가능 I 차원의 크기가 1이면 연산 가능 • 잡아 당겨서 같은 같은 크기가 되도록 변환**

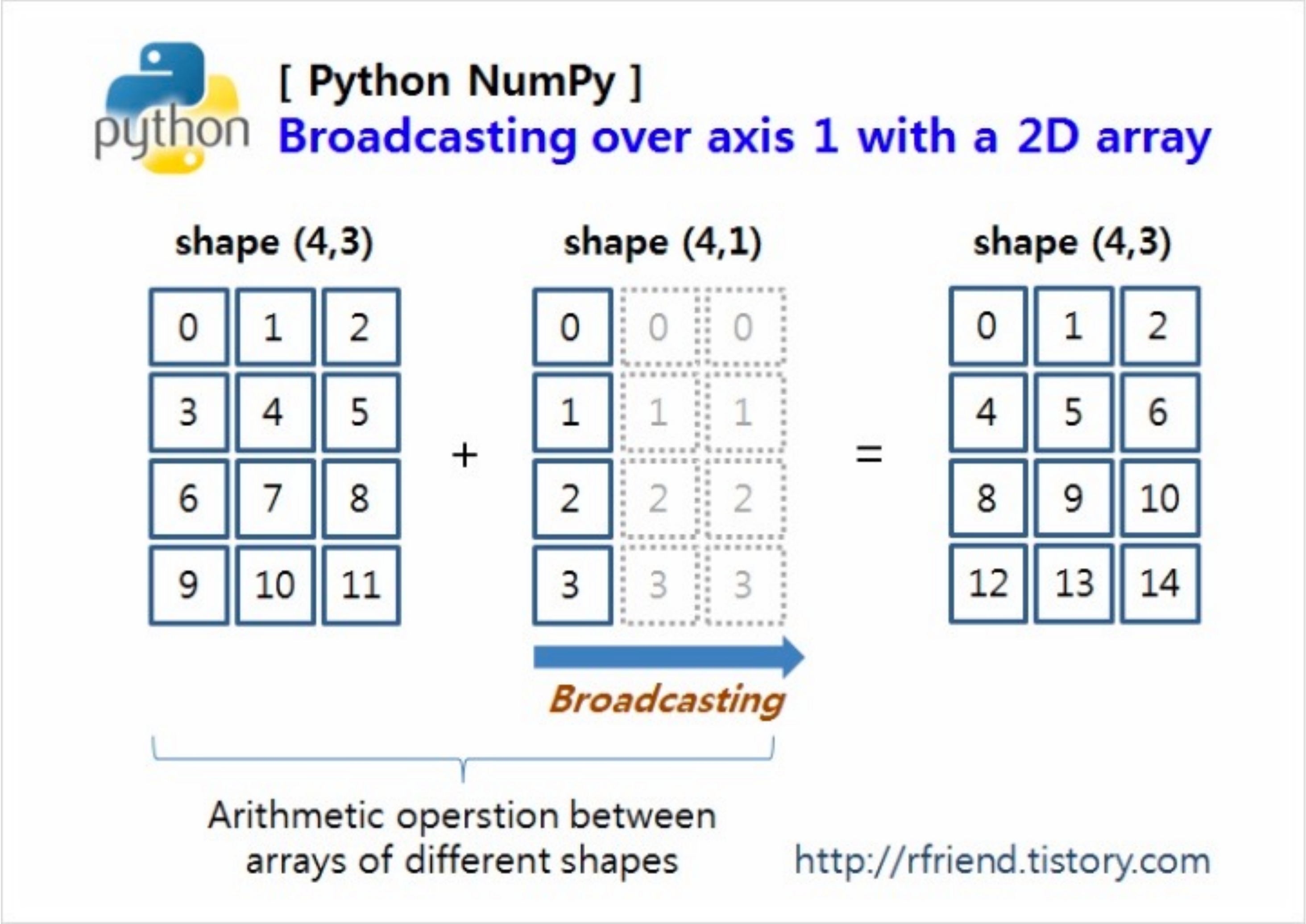
■ **잡아당기는 연산은 명시적인 메모리 복사가 일어나지 않음 • 속도/메모리 이득 • 브로드캐스팅은 좋은 아이디어다 – 가능하면 사용하자 I 브로드캐스팅과 꼴 (shape) 변환을 활용하면 루프를 피할 수 있음 • 뒤의 최단거리 이웃 예제 참조 • 고차원에서 생각하라**

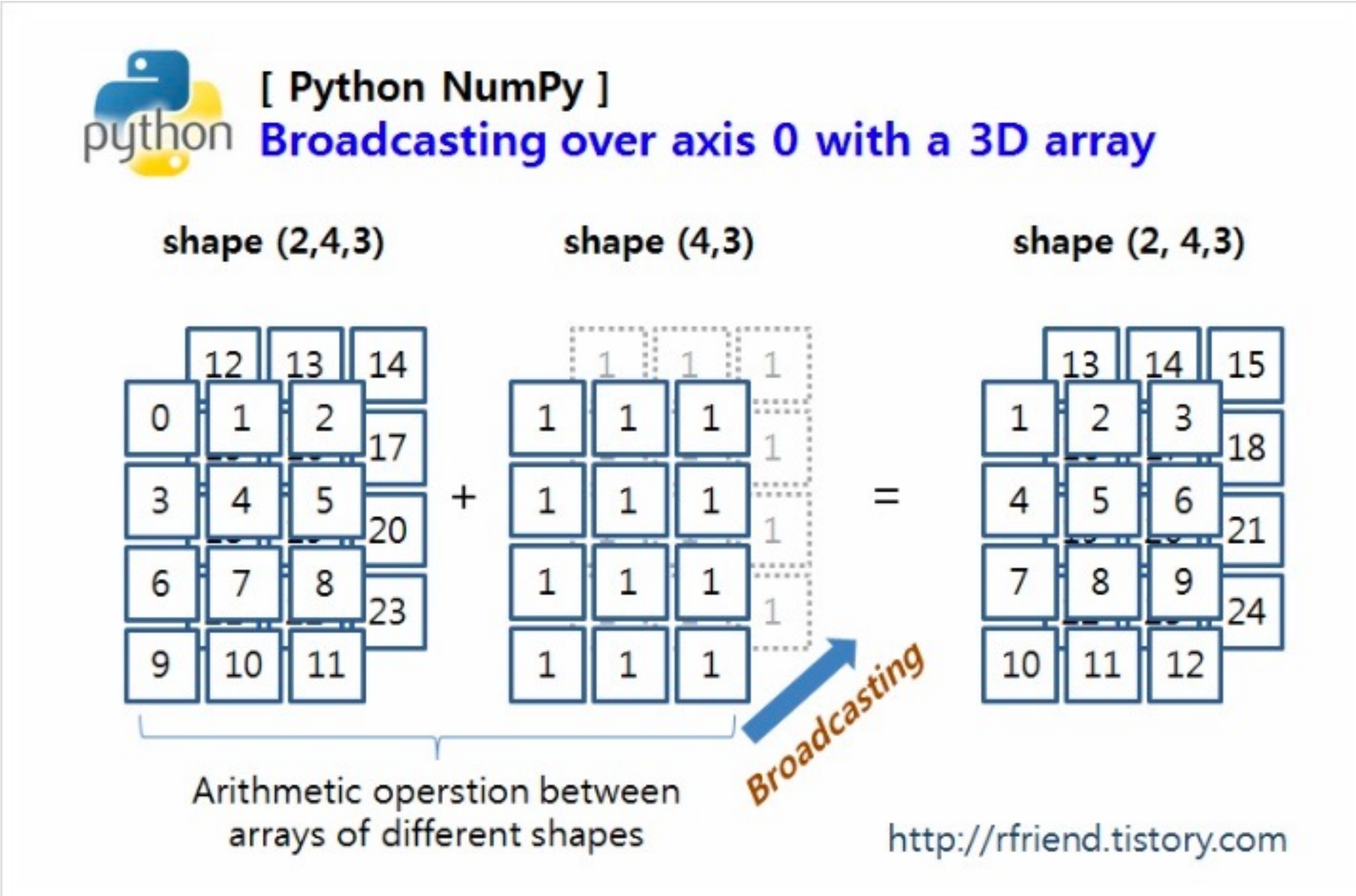


python
Operations between NumPy Arrays and Scalars
Type of Operators
Arithmetic Operators
Comparison Operators
Assignment Operators
Logical Operators
Membership Operators
elementwise operations
between equal-size arrays
vectorization
→ Very Fast than 'for loops'
Operations
with different-shape arrays
broadcasting
http://rfriend.tistory.com


[Python NumPy]
Broadcasting over axis 1 with a Scalar
shape (4,)
scalar
shape (4,)
1 2 3 4 + 1 1 1 1 = 2 3 4 5
Broadcasting
http://rfriend.tistory.com


[Python NumPy]
Broadcasting over axis 0 with a 1D array
shape (4,3)
shape (3,)
shape (4,3)
0 1 2
3 4 5
6 7 8
9 10 11
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 2 4
3 5 7
6 8 10
9 11 13
Broadcasting
Arithmetic operstion between
arrays of different shapes
http://rfriend.tistory.com


[Python NumPy]
Broadcasting over axis 1 with a 2D array
shape (4,3)
0 1 2
3 4 5
6 7 8
9 10 11
+
shape (4,1)
0 0 0
1 1 1
2 2 2
3 3 3
=
shape (4,3)
0 1 2
4 5 6
8 9 10
12 13 14
Broadcasting
Arithmetic operstion between
arrays of different shapes
http://rfriend.tistory.com


python
[Python NumPy]
Broadcasting over axis 0 with a 3D array
shape (2,4,3)
shape (4,3)
shape (2, 4,3)
0 1 2
3 4 5
6 7 8
9 10 11
12 13 14
17
20
23
+
1 1 1
1 1 1
1 1 1
1 1 1
=
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15
18
21
24
Broadcasting
Arithmetic operstion between
arrays of different shapes
http://rfriend.tistory.com

4. ufunc

- **vectorized된 연산**
  - ○ NumPy는 이미 컴파일된 코드로 만들어진 다양한 벡터화된 연산을 제공
  - ○ 모든 연산은 기본적으로 개별원소마다(elementwise)적용
  - ○ 이런 종류의 연산을 ufunc라고 함
  - ○ 더 간결하고 효율적
  - ○ **원하는 것을 설명하는 편이 명령을 내리는 것보다 낫다 (데이터 타입을 사용하자)**
  - ○ 이를 통해 컴파일된 언어의 성능을 끌어다 쓸수 있음
  - ○ **벡터화한 것이 명시적인 루프보다 좋다**
  - ○ 파이썬에서 기본적으로 제공하는 함수와 섞어 쓰지 않을것

- np.add, np.subtract, scipy.special.*, ...
- Automatically support: broadcasting, casting, ...
- The elementwise operation needs to be implemented in C (or, e.g., Cython)

- **np.info()**
- **42 ***

PEP 3118, generalized ufuncs

[Python NumPy]

Universal Functions (or ufunc) : Fast Element-wise Array Functions

1개의 배열에 대한 ufunc 함수 : Unary univerasal functions	2개의 배열 간 ufunc 함수 : Binary universal functions
input x = np.array([-2.1, 0, 1.5, 3.7])	m = np.array([0, 1, 2, 3]) n = np.array([1, 1, 2, 2])
ufunc np.abs(x)	np.add(m, n)
output array([2.1, 0. , 1.5, 3.7])	array([1, 2, 4, 5])

 # element-wise arithmetic computation
np.abs(x), np.fabs(x)
np.ceil(x), np.floor(x)
np.modf(x), np.rint(x)
np.log(x), np.log10(x), np.log2(x), np.log1p(x)
np.exp(x), np.sqrt(x), np.square(x)

 # returns boolean array
np.isanan(x), np.isfinite(x)
np.logical_not(x[, out])

 # returns the sign of each element
np.sign(x)

 # regular or inverse trigonometric functions
np.sin(x), np.cos(x), np.tan(x)
np.arcsin(x), np.arccos(x), np.arctan(x)

 # element-wise arithmetic operations b/w arrays
np.add(m, n), np.subtract(m, n)
np.multiply(m, n), np.divide(m, n)
np.floor_divide(m, n), np.mod(m, n)
np.power(m, n)
np.maximum(m, n), np.fmax(m, n)
np.minimum(m, n), np.fmin(m, n)

 # comparison operations b/w arrays
np.greater(m, n), np.greater_equal(m, n)
np.less(m, n), np.less_equal(m, n)
np.equal(m, n), np.not_equal(m, n)

 # copy sign of values
np.copysign(m, n), np.copysign(n, m)

http://rfriend.tistory.com

[Python NumPy]

Unary Universal Functions
: Fast Element-wise Array Functions

 # 배열 원소 간 곱 계산 범용 함수 (product ufucns)

```
np.prod(c, axis=0), np.prod(c, axis=1)
np.nanprod(d, axis=0), np.nanprod(d, axis=1)
np.cumprod(f, axis=0), np.cumprod(f, axis=1)
```

 # 배열 원소 간 합 계산 범용 함수 (sum ufuncs)

```
np.sum(c, axis=0), np.sum(c, axis=1)
np.nansum(d, axis=0), np.nansum(d, axis=1)
np.cumsum(f, axis=0), np.cumsum(f, axis=1)
```

 # 배열 원소간 차분 계산 범용 함수 (differnece ufuncs)

```
np.diff(g), np.diff(h, axis=0), np.diff(h, n=2, axis=1)
np.ediff1d(g),
np.ediff1d(h, to_begin=np.array([-100, -99]), to_end=np.array([99, 100]))
```

 # 배열 원소 간 기울기 계산 범용 함수 (gradient ufuncs)

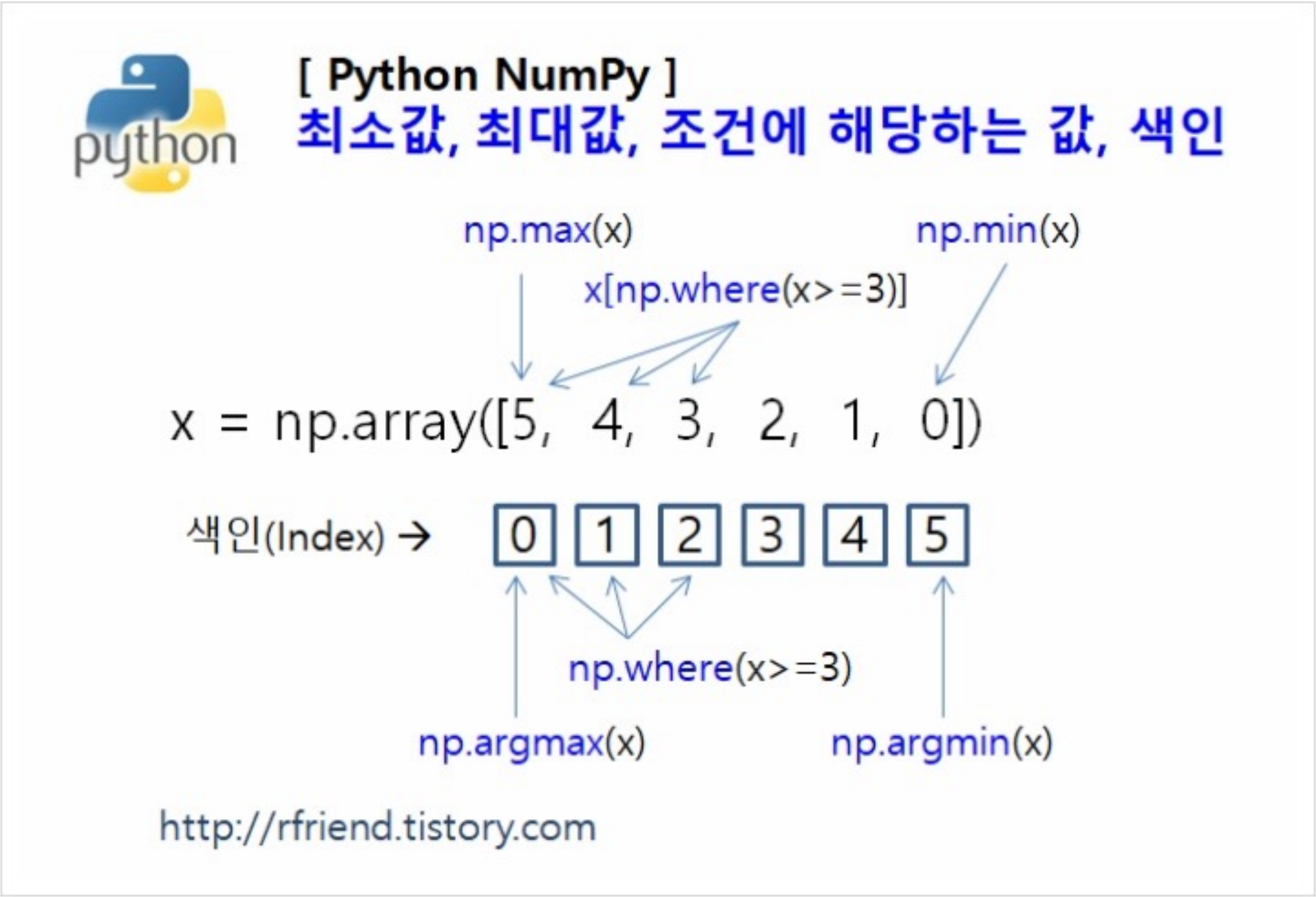
```
np.gradient(g), np.gradient(g, 2)
np.gradient(g, edge_order=2)
np.gradient(h, axis=0), np.gradient(h, axis=1)
```

http://rfriend.tistory.com

Unary ufuncs

Function	Description
abs, fabs	Compute the absolute value element-wise for integer, floating point, or complex values. Use fabs as a faster alternative for non-complex-valued data
sqrt	Compute the square root of each element. Equivalent to arr ** 0.5
square	Compute the square of each element. Equivalent to arr ** 2
exp	Compute the exponent e^x of each element
log, log10, log2, log1p	Natural logarithm (base e), log base 10, log base 2, and log(1 + x), respectively
sign	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
ceil	Compute the ceiling of each element, i.e. the smallest integer greater than or equal to each element
floor	Compute the floor of each element, i.e. the largest integer less than or equal to each element
rint	Round elements to the nearest integer, preserving thedtype
modf	Return fractional and integral parts of array as separate array
isnan	Return boolean array indicating whether each value isNaN (Not a Number)
isfinite, isinf	Return boolean array indicating whether each element is finite (non-inf, non-NaN) or infinite, respectively
cos, cosh, sin, sinh, tan, tanh	Regular and hyperbolic trigonometric functions
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of not x element-wise. Equivalent to -arr.

Function	Description
add	Add corresponding elements in arrays
subtract	Subtract elements in second array from first array
multiply	Multiply array elements
divide, floor_divide	Divide or floor divide (truncating the remainder)
power	Raise elements in first array to powers indicated in second array
maximum, fmax	Element-wise maximum. fmax ignores NaN
minimum, fmin	Element-wise minimum. fmin ignores NaN
mod	Element-wise modulus (remainder of division)
copysign	Copy sign of values in second argument to values in first argum ent
greater, greater_equal, less, less_equal, equal, not_equal	Perform element-wise comparison, yielding boolean array. Equiva lent to infix operators >, >=, <, <=, ==, !=
logical_and, logical_or, logical_xor	Compute element-wise truth value of logical operation. Equivale nt to infix operators & |, ^


[Python NumPy]
최소값, 최대값, 조건에 해당하는 값, 색인
python
np.max(x)
np.min(x)
x[np.where(x>=3)]
x = np.array([5, 4, 3, 2, 1, 0])
색인(Index) →
0 1 2 3 4 5
np.where(x>=3)
np.argmax(x)
np.argmin(x)
http://rfriend.tistory.com

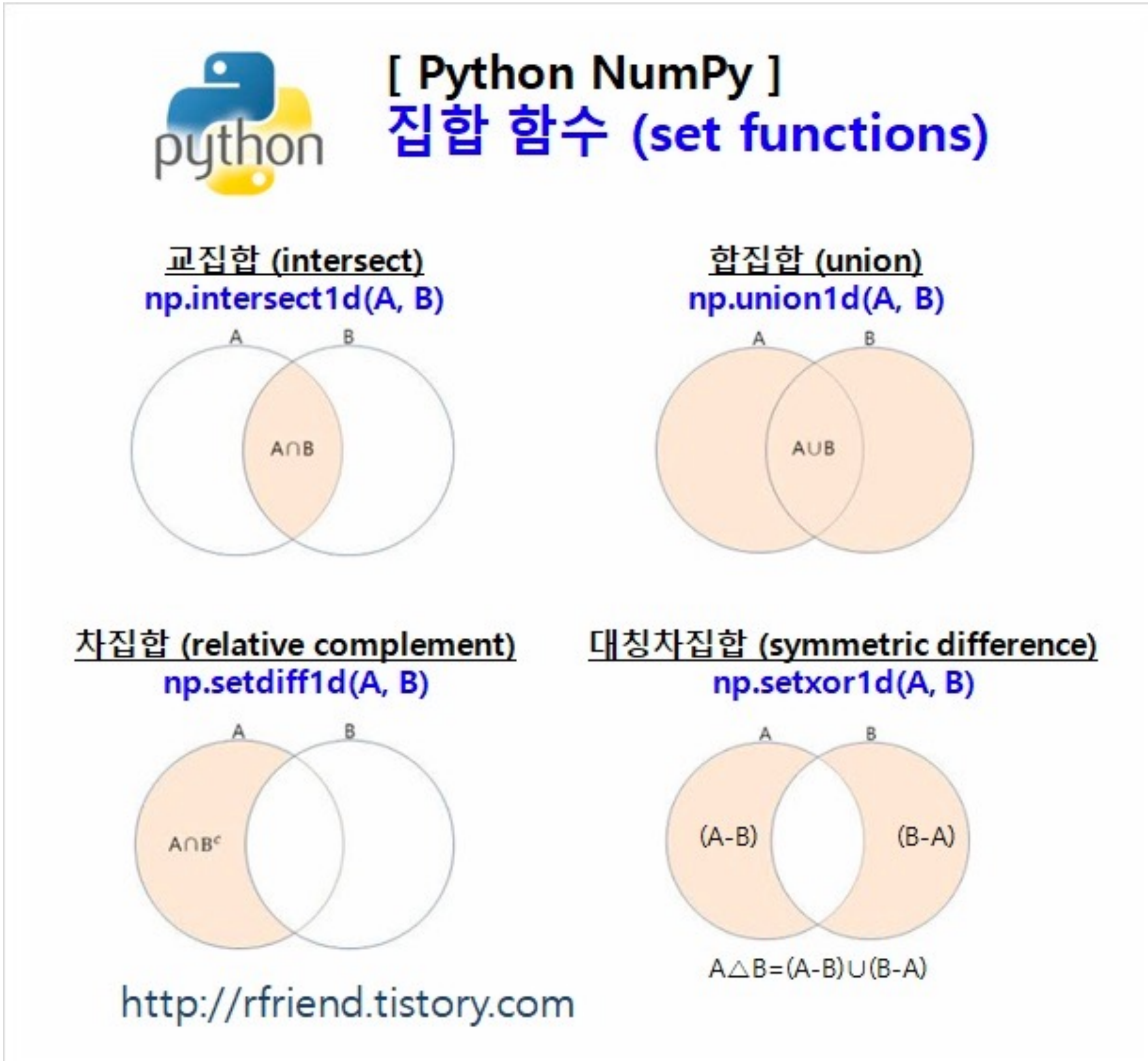
Basic array statistical methods

Method	Description
sum	Sum of all the elements in the array or along an axis. Zero-length arrays have sum 0.
mean	Arithmetic mean. Zero-length arrays have NaN mean.
std, var	Standard deviation and variance, respectively, with optional degrees of freedom adjustment (default denominator n).
min, max	Minimum and maximum.
argmin, argmax	Indices of minimum and maximum elements, respectively.
cumsum	Cumulative sum of elements starting from 0
cumprod	Cumulative product of elements starting from 1

[Python numpy]
배열에서 0보다 작은 수는 0으로 대체하기

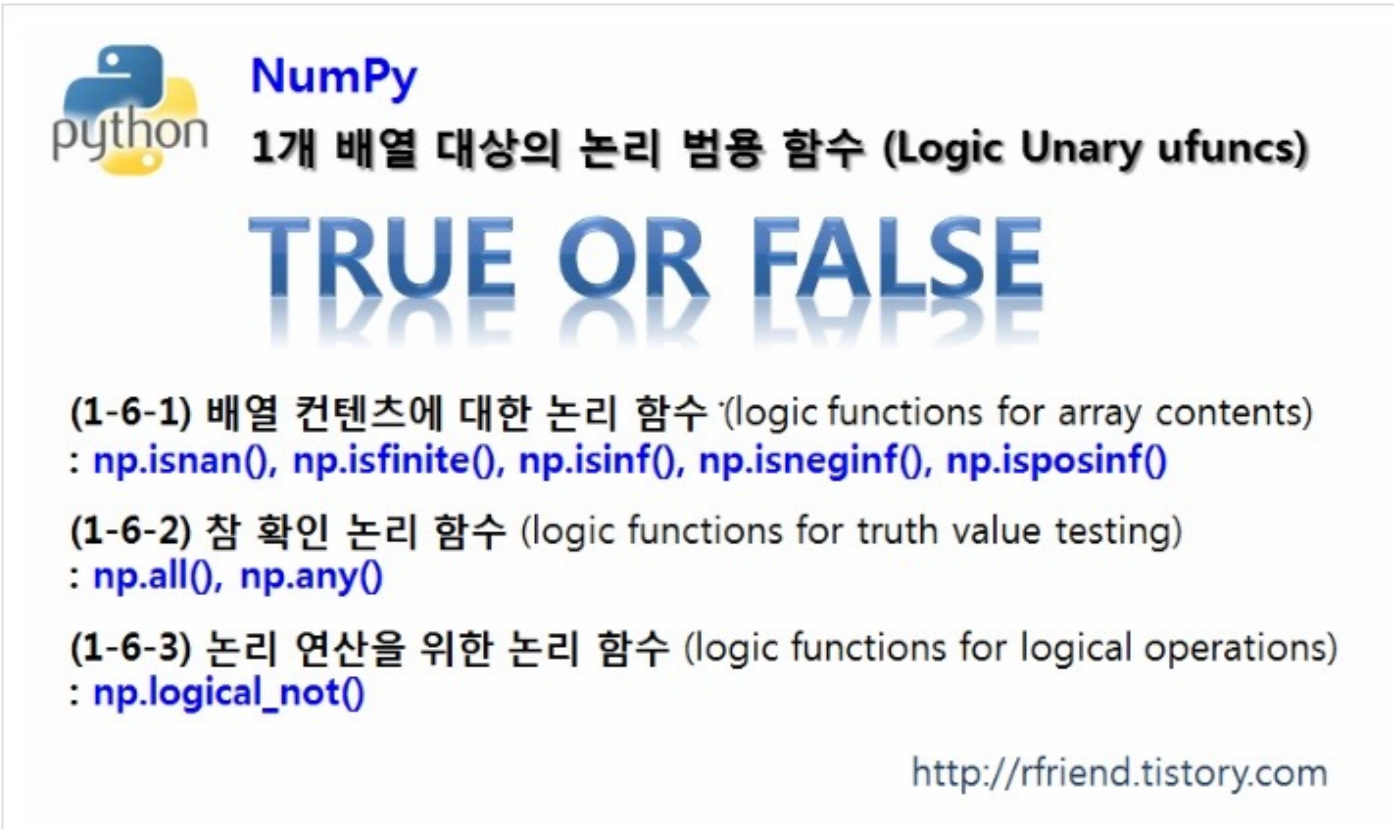
```
a = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]

# list comprehension
[0 if i < 0 else i for i in a]

# Indexing
a[a < 0] = 0

# np.where()
np.where(a < 0, 0, a)

# np.clip()
np.clip(a, 0, 4)  or  a.clip(0)

[0, 0, 0, 0, 0, 0, 1, 2, 3, 4]
```

http://rfriend.tistory.com


[Python NumPy]
집합 함수 (set functions)
교집합 (intersect)
np.intersect1d(A, B)
A
B
A∩B
합집합 (union)
np.union1d(A, B)
A
B
A∪B
차집합 (relative complement)
np.setdiff1d(A, B)
A
B
A∩B^c
대칭차집합 (symmetric difference)
np.setxor1d(A, B)
A
B
(A-B)
(B-A)
A△B=(A-B)∪(B-A)
http://rfriend.tistory.com

Array set operations

Method	Description
unique(x)	Compute the sorted, unique elements in x
intersect1d(x, y)	Compute the sorted, common elements in x and y
union1d(x, y)	Compute the sorted union of elements
in1d(x, y)	Compute a boolean array indicating whether each element of x is contained in y
setdiff1d(x, y)	Set difference, elements in x that are not in y
setxor1d(x, y)	Set symmetric differences; elements that are in either of the arrays, but not both


NumPy
1개 배열 대상의 논리 범용 함수 (Logic Unary ufuncs)
TRUE OR FALSE
(1-6-1) 배열 컨텐츠에 대한 논리 함수 (logic functions for array contents)
: np.isnan(), np.isfinite(), np.isinf(), np.isneginf(), np.isposinf()
(1-6-2) 참 확인 논리 함수 (logic functions for truth value testing)
: np.all(), np.any()
(1-6-3) 논리 연산을 위한 논리 함수 (logic functions for logical operations)
: np.logical_not()
http://rfriend.tistory.com

[Python NumPy]

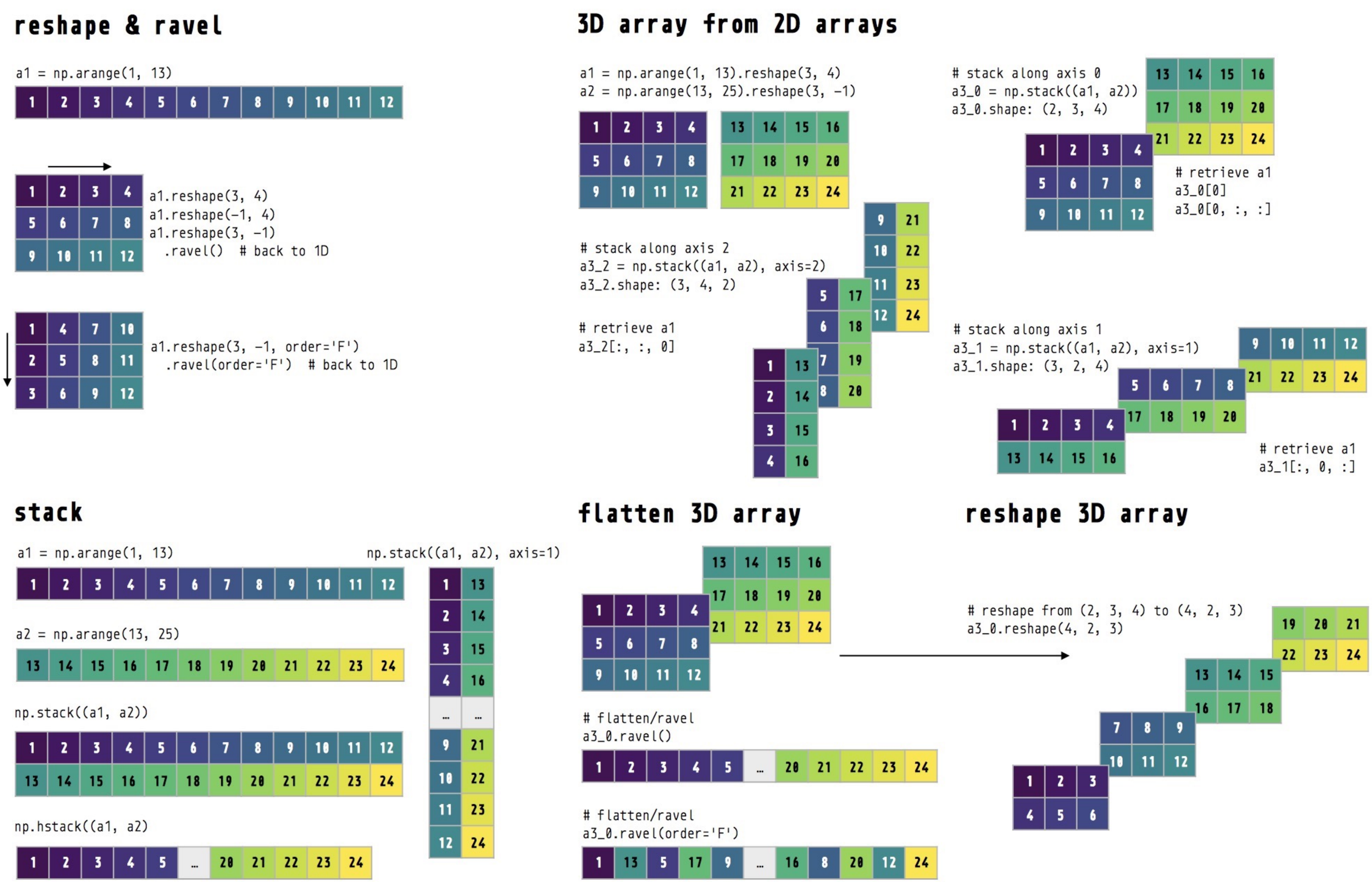
선형대수 (Linear Algebra)

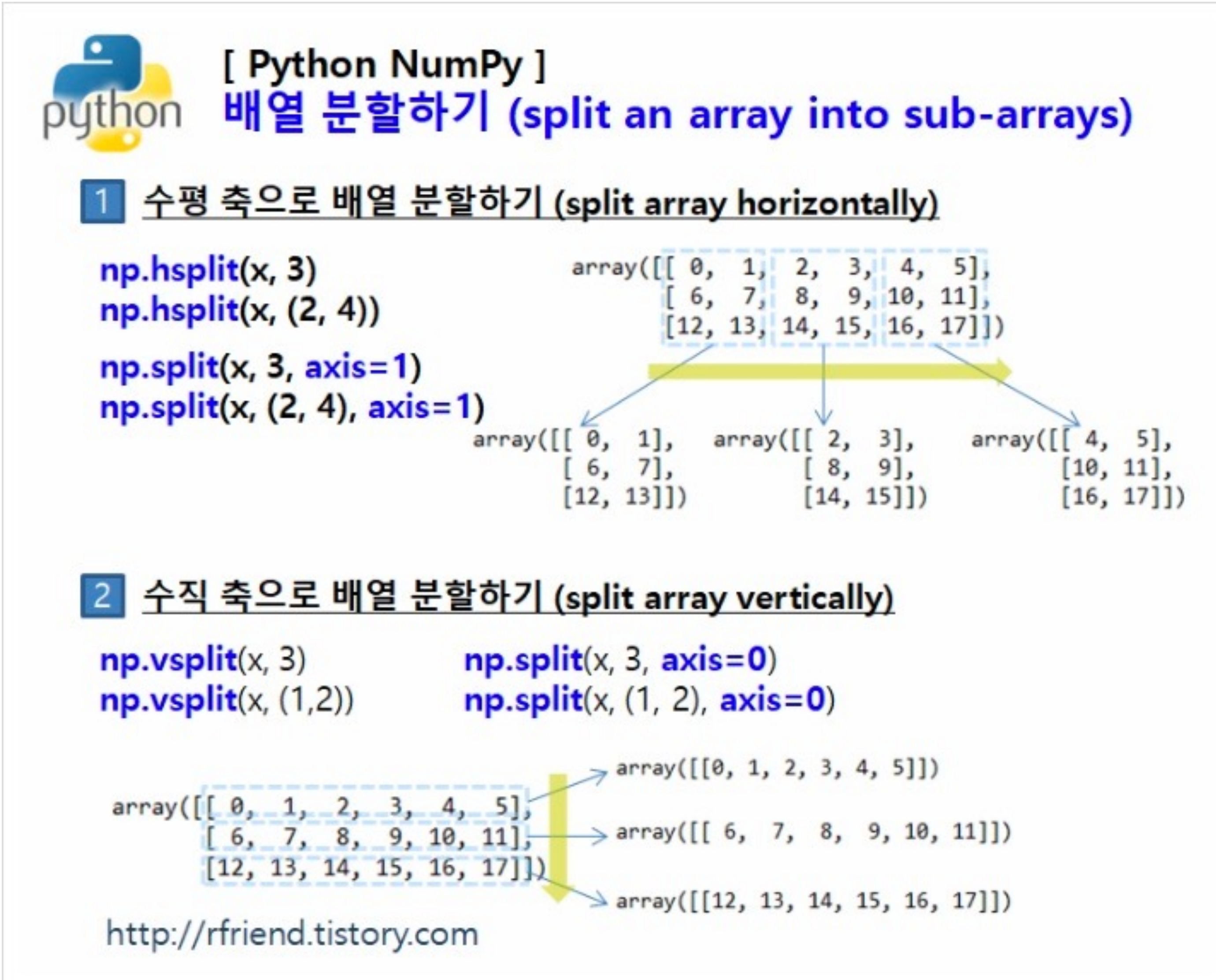
- **대각행렬** (Diagonal matrix): **np.diag(x)**
- **내적** (Dot product, Inner product): **np.dot(a, b)**
- **대각합** (Trace): **np.trace(x)**
- **행렬식** (Matrix Determinant): **np.linalg.det(x)**
- **역행렬** (Inverse of a matrix): **np.linalg.inv(x)**
- **고유값** (Eigenvalue), **고유벡터** (Eigenvector): **w, v = np.linalg.eig(x)**
- **특이값 분해** (Singular Value Decomposition): **u, s, vh = np.linalg.svd(A)**
- **연립방정식 해 풀기** (Solve a linear matrix equation): **np.linalg.solve(a, b)**
- **최소자승 해 풀기** (Compute the Least-squares solution)
  : **m, c = np.linalg.lstsq(A, y, rcond=None)[0]**

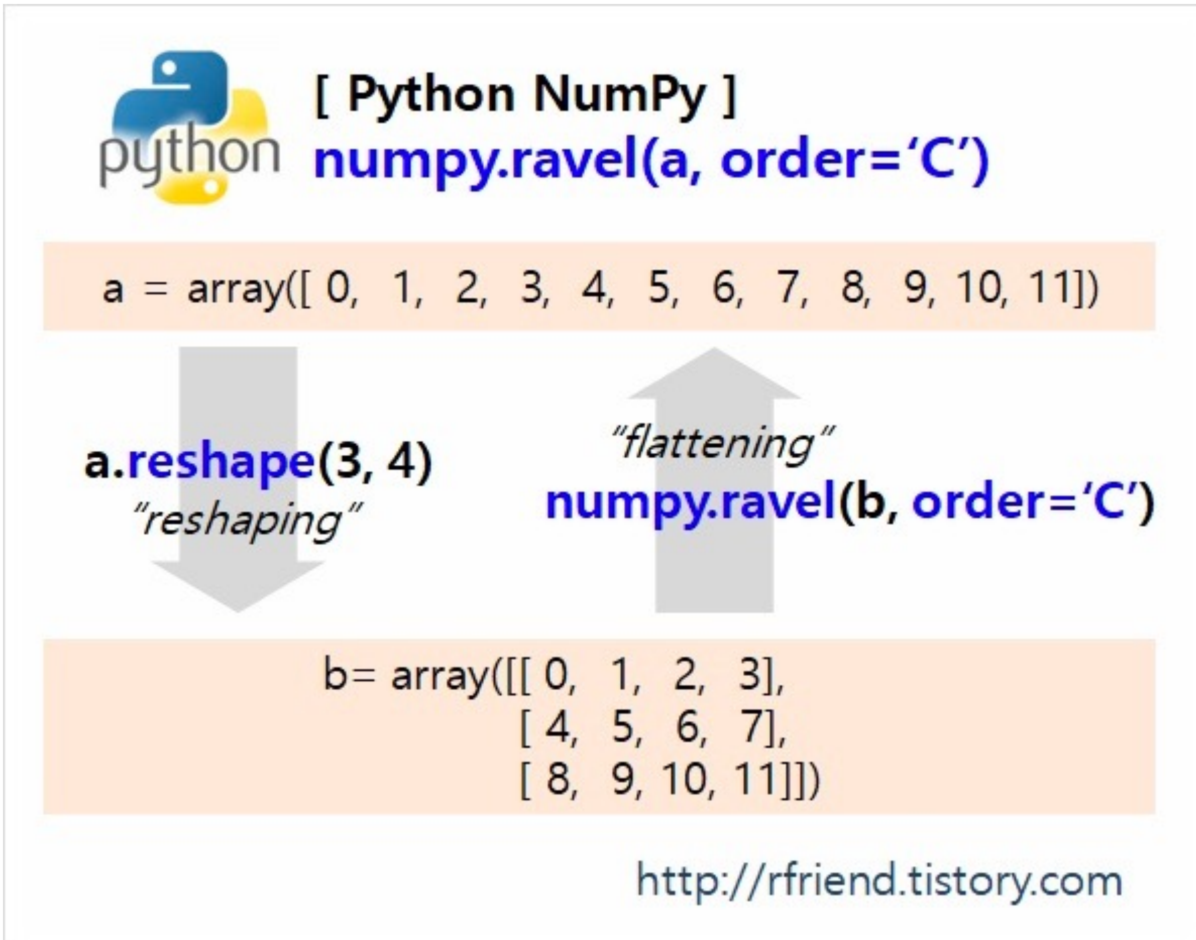
http://rfriend.tistory.com

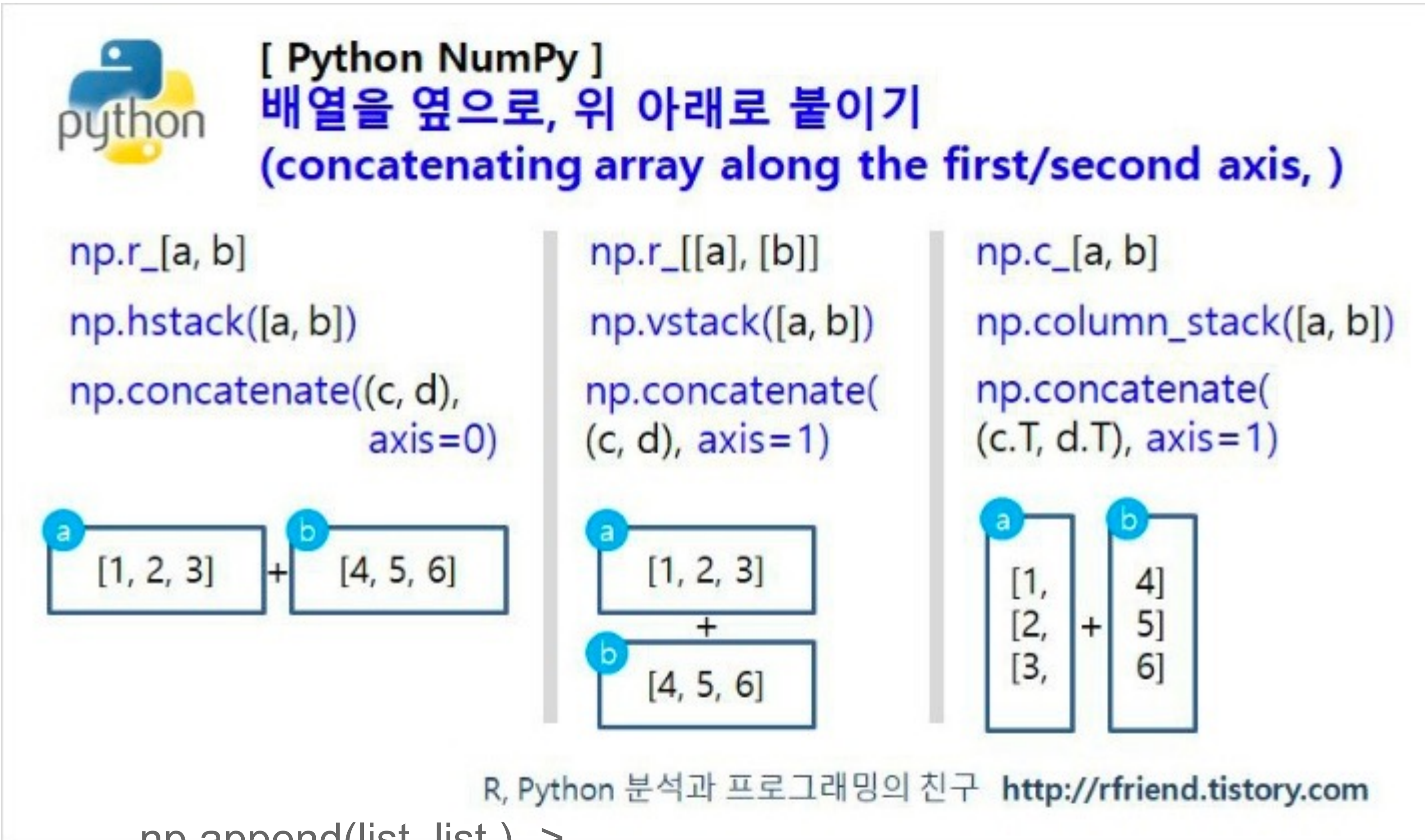
5. shape

Python numpy reshape and stack cheatsheet



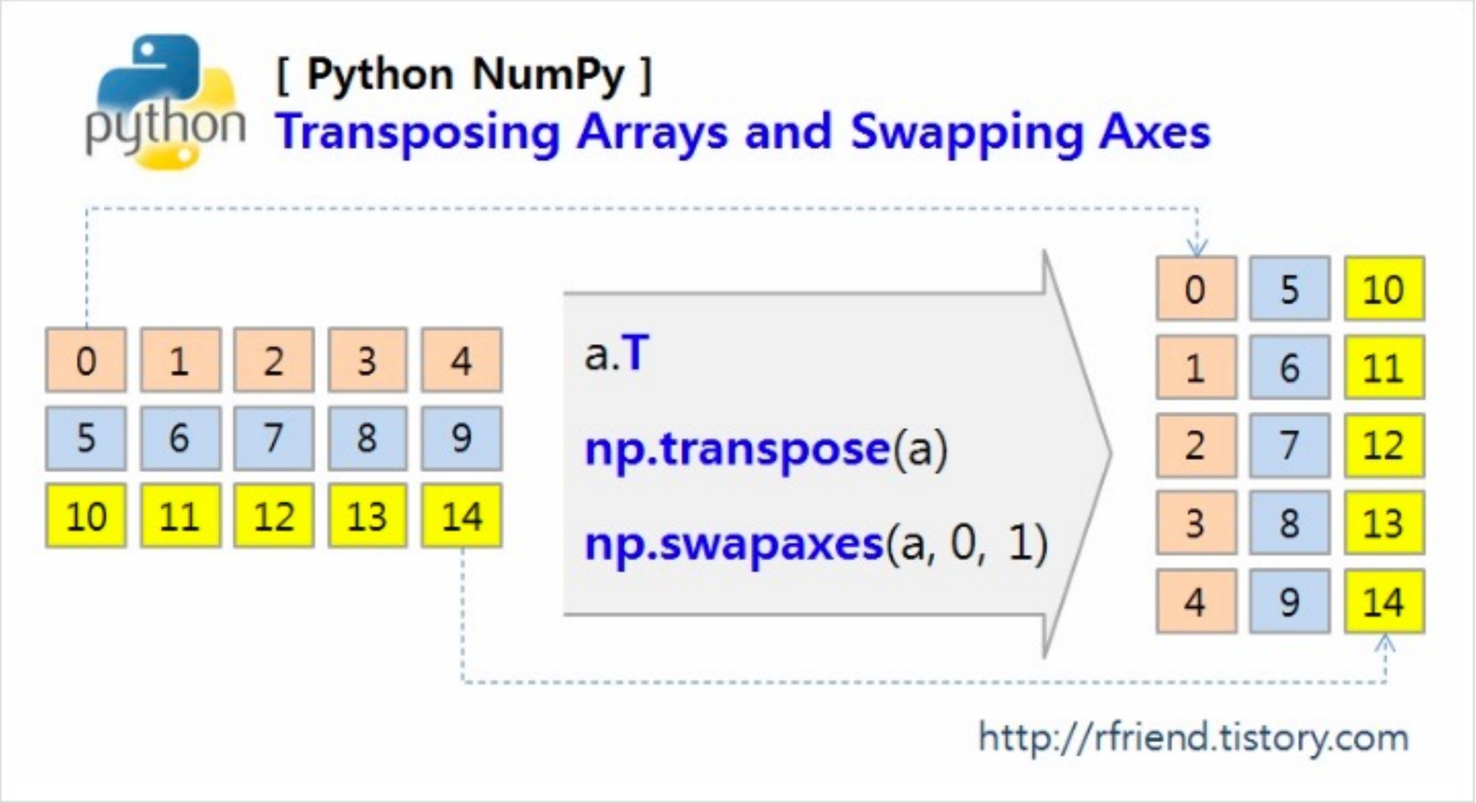

[Python NumPy]
배열 분할하기 (split an array into sub-arrays)
1 수평 축으로 배열 분할하기 (split array horizontally)
np.hsplit(x, 3)
np.hsplit(x, (2, 4))
np.split(x, 3, axis=1)
np.split(x, (2, 4), axis=1)
array([[0, 1, 2, 3, 4, 5],
[6, 7, 8, 9, 10, 11],
[12, 13, 14, 15, 16, 17]])
array([[0, 1],
[6, 7],
[12, 13]])
array([[2, 3],
[8, 9],
[14, 15]])
array([[4, 5],
[10, 11],
[16, 17]])
2 수직 축으로 배열 분할하기 (split array vertically)
np.vsplit(x, 3)
np.vsplit(x, (1,2))
np.split(x, 3, axis=0)
np.split(x, (1, 2), axis=0)
array([[0, 1, 2, 3, 4, 5],
[6, 7, 8, 9, 10, 11],
[12, 13, 14, 15, 16, 17]])
array([[0, 1, 2, 3, 4, 5]])
array([[6, 7, 8, 9, 10, 11]])
array([[12, 13, 14, 15, 16, 17]])
http://rfriend.tistory.com


[Python NumPy]
numpy.ravel(a, order='C')
a = array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
a.reshape(3, 4)
"reshaping"
"flattening"
numpy.ravel(b, order='C')
b= array([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]])
http://rfriend.tistory.com

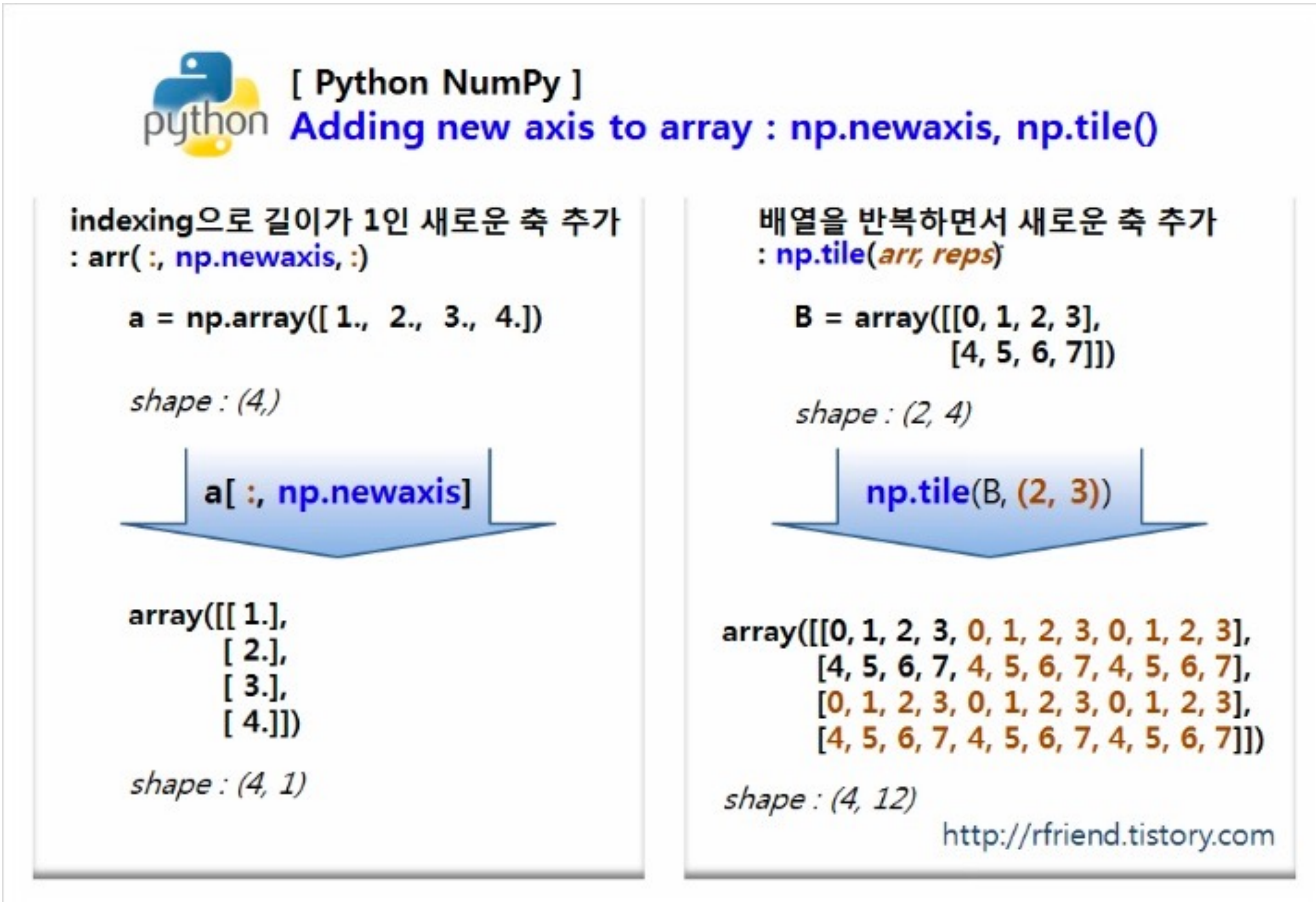


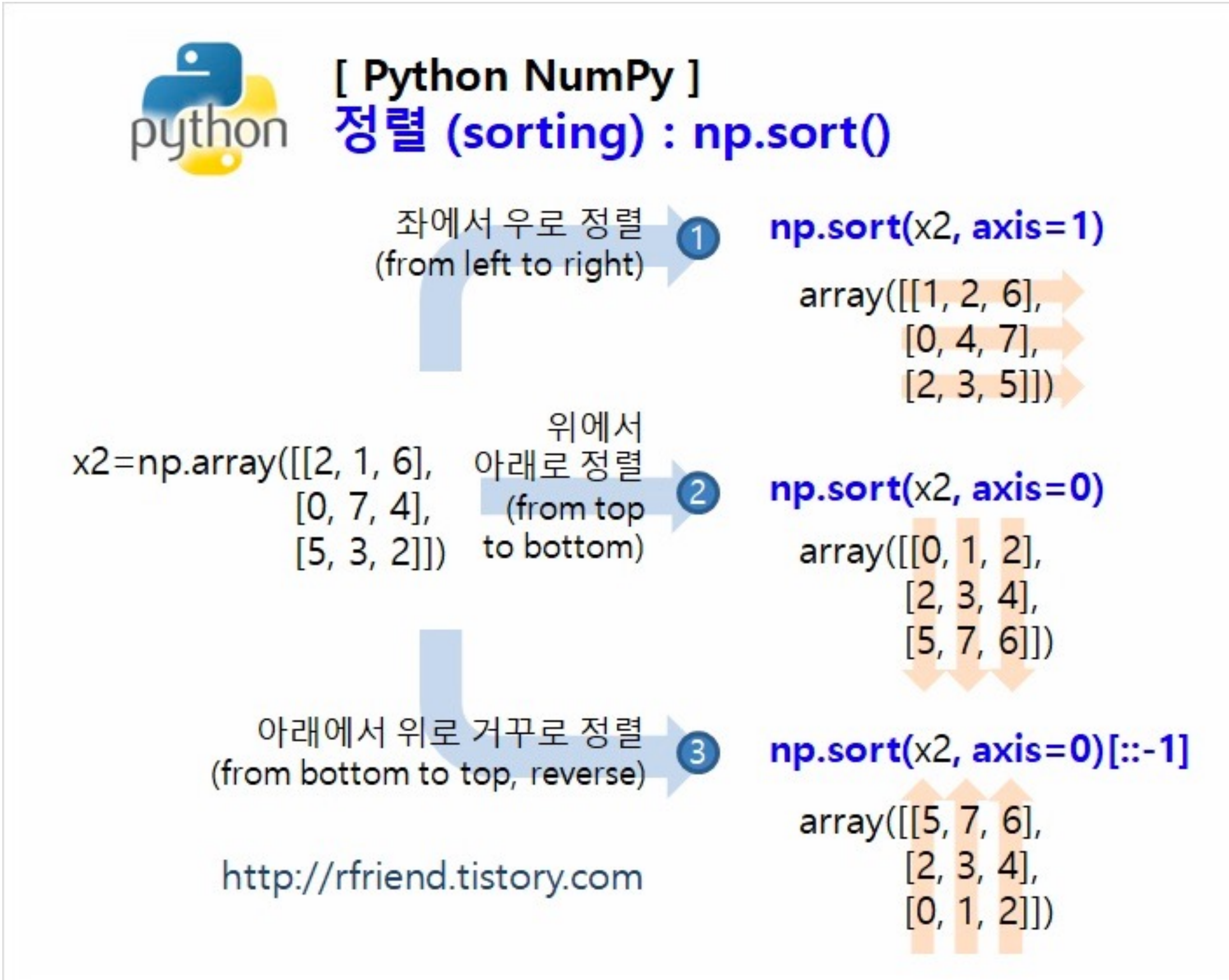
np.append(list, list) ->
list , np.append(list, numpy) -> numpy

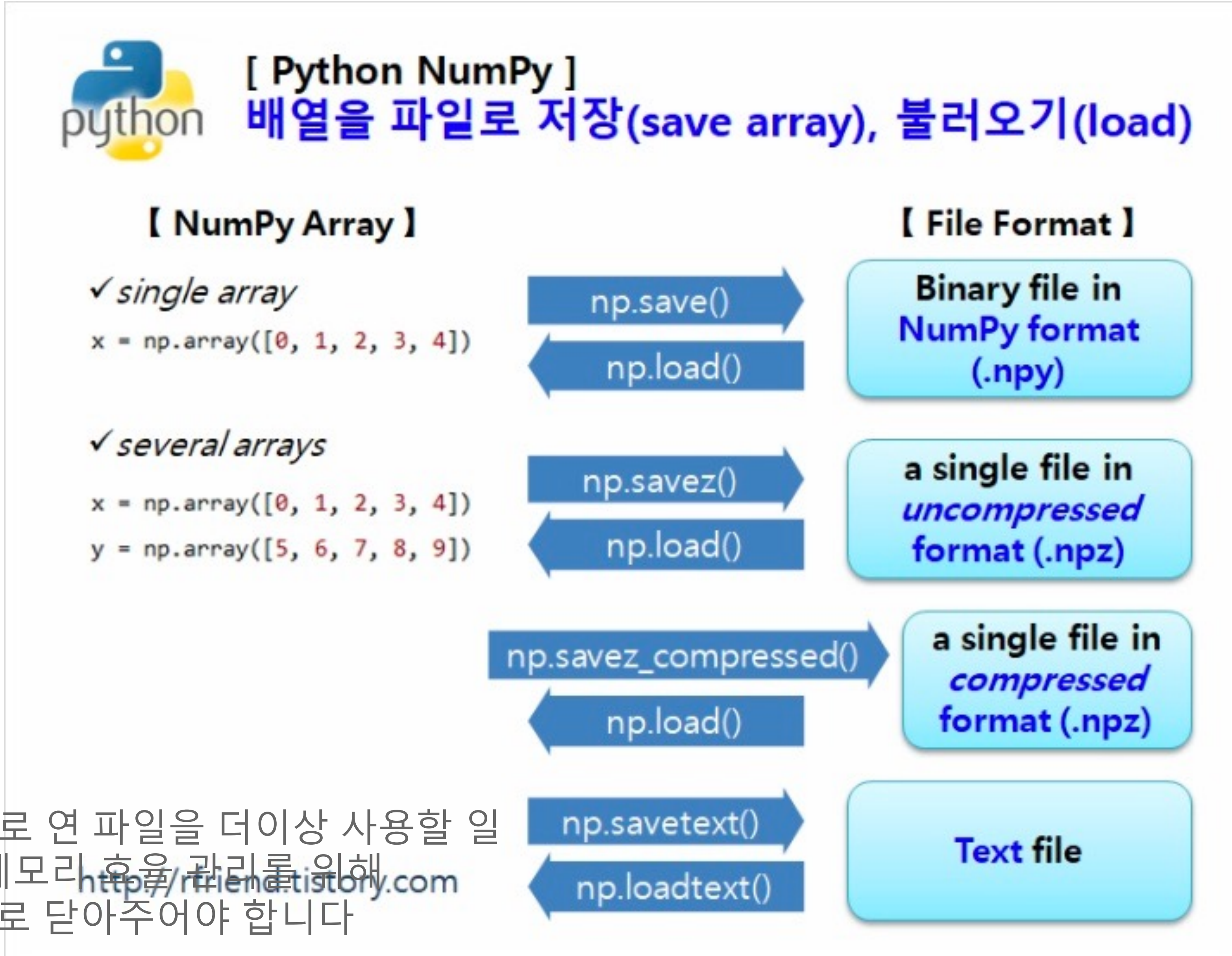
=np.concatenate((c, d), axis = 1)
<- 1D AxisError. 2D 이상 배열에 사용


[Python NumPy]
Transposing Arrays and Swapping Axes
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
a.T
np.transpose(a)
np.swapaxes(a, 0, 1)
0 5 10
1 6 11
2 7 12
3 8 13
4 9 14
http://rfriend.tistory.com

Vs np.repeat


[Python NumPy]
Adding new axis to array : np.newaxis, np.tile()
indexing으로 길이가 1인 새로운 축 추가
: arr(:, np.newaxis, :)
a = np.array([1., 2., 3., 4.])
shape : (4,)
a[:, np.newaxis]
array([[1.],
[2.],
[3.],
[4.]])
shape : (4, 1)
배열을 반복하면서 새로운 축 추가
: np.tile(arr, reps)
B = array([[0, 1, 2, 3],
[4, 5, 6, 7]])
shape : (2, 4)
np.tile(B, (2, 3))
array([[0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3],
[4, 5, 6, 7, 4, 5, 6, 7, 4, 5, 6, 7],
[0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3],
[4, 5, 6, 7, 4, 5, 6, 7, 4, 5, 6, 7]])
shape : (4, 12)
http://rfriend.tistory.com


[Python NumPy]
정렬 (sorting) : np.sort()
좌에서 우로 정렬
(from left to right)
np.sort(x2, axis=1)
array([[1, 2, 6],
[0, 4, 7],
[2, 3, 5]])
x2=np.array([[2, 1, 6],
[0, 7, 4],
[5, 3, 2]])
위에서
아래로 정렬
(from top
to bottom)
np.sort(x2, axis=0)
array([[0, 1, 2],
[2, 3, 4],
[5, 7, 6]])
아래에서 위로 거꾸로 정렬
(from bottom to top, reverse)
np.sort(x2, axis=0)[::-1]
array([[5, 7, 6],
[2, 3, 4],
[0, 1, 2]])
http://rfriend.tistory.com



d() 함수로 연 파일을 더이상 사용할 일
없으면 메모리 효율 관리를 위해
close() 로 닫아주어야 합니다

Einsum is All You Need

https://rockt.github.io/2018/04/30/einsum

■ Einsum 표기법

○ Domain Specific Language를 이용해, tensor 연산을 표기하는 방법

○ 간결하며, 쉬운 표기법

- 거의 모든 연산에 대해 중간 연산 없이 계산을 할 수 있음

○ 많은 프레임워크가 Einsum에 대한 최적화가 잘 되어있음

- numpy via np.einsum,
- PyTorch via torch.einsum
- TensorFlow via tf.einsum

einsum(equation,operands)
equation is a string representing the Einstein summation
operands is a sequence of tensor

In numpy and TensorFlow, operands can be a variable-length argument list whereas in PyTorch it needs to be a list.

- *Vector inner product:* "a,a->" (Assumes two vectors of same length)
- *Vector element-wise product:* "a,a->a" (Assumes two vectors of same length)
- *Vector outer product:* "a,b->ab" (Vectors not necessarily same length.)
- *Matrix transposition:* "ab->ba"
- *Matrix diagonal:* "ii->i"
- *Matrix trace:* "ii->"
- *1-D Sum:* "a->"
- *2-D Sum:* "ab->"
- *3-D Sum:* "abc->"
- *Matrix inner product* "ab,ab->" (If you pass twice the same argument, it becomes a matrix L2 norm)
- *Left-multiplication Matrix-Vector:* "ab,b->a"
- *Right-multiplication Vector-Matrix:* "a,ab->b"
- *Matrix Multiply:* "ab,bc->ac"
- *Batch Matrix Multiply:* "Yab,Ybc->Yac"
- *Quadratic form / Mahalanobis Distance:* "a,ab,b->"
- ...

What type of argument(s) do "->" and ",->" take, and what do they do with them?

```python
#  1: Imports.
import numpy as np
```

```python
#  2: Invocation. Requires format string + any # of input args.
arg0 = np.random.normal(...)
arg1 = np.random.normal(...)
...
argn = np.random.normal(...)

# ...

dst  = np.einsum("", arg0, arg1, ..., argn)
```

```python
#  3: Format string. Incomplete example with 3 input args.
dst  = np.einsum(",,->", arg0, arg1, arg2)
```

```
#  4: Format string. Incomplete example with 3 input args.
dst  = np.einsum("␣␣,␣␣␣,␣␣->␣␣␣␣", arg0, arg1, arg2)
assert arg0.ndim == len("␣␣")                    # (Order-2 Tensor)
assert arg1.ndim == len("␣␣␣")                   # (Order-3 Tensor)
assert arg2.ndim == len("␣␣")                    # (Order-2 Tensor)
assert dst .ndim == len("␣␣␣␣")                  # (Order-4 Tensor)
```

```
#  5: Format string. Complete examples with 1 and 2 args.
s = np.einsum("a->",        v    )
T = np.einsum("ij->ji",     M    )
C = np.einsum("mn,np->mp", A, B)


assert v.ndim == len("a")
assert s.ndim == len("")

assert M.ndim == len("ij")
assert T.ndim == len("ji")

assert A.ndim == len("mn")
assert B.ndim == len("np")
assert C.ndim == len("mp")
```

```
#   6: Elaborated Example. Matrix multiplication.
#     C     =     A     *     B
#   Ni x Nj   =  Ni x Nk   *  Nk x Nj
#               |    ^-----^     |
#               |     Match      |
#               |                |
#               -------- ---------
#                       V
#   Ni x Nj  <------- Ni x Nj

C = np.empty((Ni,Nj))
for i in  range(Ni):
      for j in  range(Nj):
             total = 0

             for k in  range(Nk):
                   total += A[i,k]*B[k,j]

             C[i,j] = total

C = np.einsum("ik,kj->ij", A, B)
```

Free Indices: i,j

Summation Indices: all non-free indices (k)

```python
#  7. Free Indices
C = np.empty((Ni,Nj))
for i in  range(Ni):
    for j in  range(Nj):
        total = 0

        for k in  range(Nk):

            total += A[i,k]*B[k,j]

        C[i,j] = total




C = np.einsum("ik,kj->ij", A, B)
```

```python
#  8. Summation Indices
C = np.empty((Ni,Nj))
for i in  range(Ni):
      for j in  range(Nj):
             total = 0

             for k in  range(Nk):

                   total += A[i,k]*B[k,j]

             C[i,j] = total




C = np.einsum("ik,kj->ij", A, B)
```

```
#  9. Elaborated Example. Matrix diagonal extraction.
d = np.empty((Ni))
for i in xrange(Ni):
        total = 0


        total +=      M[i,i]

        d[i] = total




d = np.einsum("ii->i", M)
```

} *Free Indices:* i

} *Summation Indices:* None

```
# 10. Elaborated Example. Matrix trace.
Tr = 0   # Scalar! Has dimension 0 and no indices


total = 0

for i in xrange(Ni):
      total +=       M[i,i]

Tr = total





Tr = np.einsum("ii->", M)
```

} *Free Indices:* None

} *Summation Indices:* i

```
# 11. Elaborated Example. Quadratic Form.
x = 0


total = 0

for s in xrange(Ns):
        for t in xrange(Nt):
                total += v[s]*M[s,t]*v[t]

x = total




x = np.einsum("s,st,t->", v, M, v)
```

Free Indices: None

Summation Indices: s, t

```
# 12. Elaborated example. Batched outer product.

R = np.empty((NB,ni,nj))
for B in xrange(NB):
    for i in xrange(Ni):
        for j in xrange(Nj):
            total = 0


            total += P[B,i]*Q[B,j]

            R[B,i,j] = total




R = np.einsum("Bi,Bj->Bij", P, Q)
```

Free Indices: B,i,j

Summation Indices: None

```
# 13. Natural consequences of einsum definition.
#     Requirements on size of individual axes.
C =               np.einsum("ik,kj->ij", A, B)



assert A.shape == (Ni,Nk)
assert B.shape ==        (Nk,Nj)
assert C.shape ==               (Ni,Nj)

#     Requirement for identical size of certain axes due to
#     shared index label.
#         Example 1: Matrix Multiplication
C =               np.einsum("ik,kj->ij", A, B)



assert            A.shape[1] == B.shape[0]         #  == Nk
assert            A.shape[0] == C.shape[0]         #  == Ni
assert            B.shape[1] == C.shape[1]         #  == Nj

#         Example 2: Matrix Diagonal Extraction
d =               np.einsum("ii->i", D)



assert            D.shape[0] == D.shape[1]         #  == Ni
assert            D.shape[1] == d.shape[0]         #  == Ni
```

```
# 14: Format Strings. Rules.

#    Bad. Number of input index groups doesn't match number of
#         arguments.
np.einsum("ab,bc->ac", A)

#    Bad. Indexes must be ASCII upper/lowercase letters.
np.einsum("012,1^%->:;?", A, B)

#    Bad. Argument 0 has 3 dimensions but only 2 indices are
#         given.
A = np.random.normal(size = (2,3,4))
B = np.random.normal(size = (4,5,6))
np.einsum("ab,bcd->a", A, B)

#    Bad. One of the output indices isn't in the set of all
#         input indices.
np.einsum("ab,bc->acz", A, B)

#    Bad. Output has a repeated index.
np.einsum("ab,bc->baa", A, B)

#    Bad. Mismatches in the sizes of input argument axes
#         that are labelled with the same index.
A = np.random.normal(size = (2,3,4))
B = np.random.normal(size = (3,4,5))
np.einsum("ckj,cqq->c", A, B)

assert      A.shape[0] == B.shape[0]           # ERROR: 2 != 3
assert      B.shape[1] == B.shape[2]           # ERROR: 4 != 5
```

```
# 15: MLP Backprop done easily (stochastic version).
#     h = sigmoid(Wx + b)
#     y = softmax(Vh + c)
Ni = 784
Nh = 500
No =  10

W  = np.random.normal(size = (Nh,Ni))  # Nh x Ni
b  = np.random.normal(size = (Nh,))    # Nh
V  = np.random.normal(size = (No,Nh))  # No x Nh
c  = np.random.normal(size = (No,))    # No

# Load x and t...
x, t  = train_set[k]

# With a judicious, consistent choice of index labels, we can
# express fprop() and bprop() extremely tersely; No thought
# needs to be given about the details of shoehorning matrices
# into np.dot(), such as the exact argument order and the
# required transpositions.
#
# Let
#
#     'i' be the input  dimension label.
#     'h' be the hidden dimension label.
#     'o' be the output dimension label.
#
# Then

# Fprop
ha    = np.einsum("hi, i -> h", W, x) + b
h     = sigmoid(ha)
ya    = np.einsum("oh, h -> o", V, h) + c
y     = softmax(ya)

# Bprop
dLdya = y - t
dLdV  = np.einsum("h , o -> oh", h, dLdya)
dLdc  = dLdya
dLdh  = np.einsum("oh, o -> h ", V, dLdya)
dLdha = dLdh * sigmoidgrad(ha)
dLdW  = np.einsum("i,  h -> hi", x, dLdha)
dLdb  = dLdha
```

```python
# 16: MLP Backprop done easily (batch version).
#      But we want to exploit hardware with a batch version!
#      This is trivially implemented with simple additions
#      to np.einsum's format string, in addition to the usual
#      averaging logic required when handling batches. We
#      implement even that logic with einsum for demonstration
#      and elegance purposes.
batch_size = 128

# Let
#      'B' be the batch  dimension label.
#      'i' be the input  dimension label.
#      'h' be the hidden dimension label.
#      'o' be the output dimension label.
#
# Then

# Fprop
ha     = np.einsum("hi, Bi -> Bh", W, x) + b
h      = sigmoid(ha)
ya     = np.einsum("oh, Bh -> Bo", V, h) + c
y      = softmax(ya)

# Bprop
dLdya = y - t
dLdV  = np.einsum("Bh, Bo -> oh", h, dLdya) / batch_size
dLdc  = np.einsum("Bo     -> o ",    dLdya) / batch_size
dLdh  = np.einsum("oh, Bo -> Bh", V, dLdya)
dLdha = dLdh * sigmoidgrad(ha)
dLdW  = np.einsum("Bi, Bh -> hi", x, dLdha) / batch_size
dLdb  = np.einsum("Bh     -> h ",    dLdha) / batch_size
```

- Give each axis that notionally exists within the problem its own label. It is best if memorable ones can be chosen, like in the MLP problem above.
- If you want element-wise multiplication between the axes of two arguments, use the same index for both ("a,a->a").
- If you want summation along a given axis, **don't** put its index in the output specification ("a->").
- If you want an inner product, which is element-wise multiplication between two axes followed by the summing-out of those axes, then *do both of the above* ("a,a->").
- If a tensor is used as argument to einsum(), simply copy-paste its specification from the einsum() that created it. Input transpositions are automagically handled.
- For the output, simply state what is the form of the tensor that you want. The genie in einsum() will give it to you, and you have infinite wishes.

■ Zen of NumPy (and Pandas)

- ○ Strided is better than scattered
- ○ Contiguous is better than strided
- ○ Descriptive is better than imperative (use data-types)
- ○ Array-oriented and data-oriented is often better than object-oriented
- ○ Broadcasting is a great idea –use where possible
- ○ Split-apply-combine is a great idea – use where possible
- ○ Vectorized is better than an explicit loop
- ○ Write more ufuncs and generalized ufuncs (numba can help)
- ○ Unless it’s complicated — then use numba
- ○ Think in higher dimensions

■ Zen of Data Science

○ Get More and better data.

○ Better data is determined by better models.

○ How you compute matters.

○ Put the data in the hands and minds of people with knowledge.

○ Fail quickly and often—but not in the same way.

○ Where and how the data is stored is secondary to analysis and understanding.

○ Premature horizontal scaling is the root of all evil.

○ When you must scale —data locality and parallel algorithms are the key.

○ Learn to think in building blocks that can be parallelized.