

NumPy/SciPy

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

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

<https://docs.scipy.org/doc/numpy/user/quickstart.html>

1. Why NumPy?

1. Python is very slow!

Slowest things on earth:



■ Computing Power

- GPU
- 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)

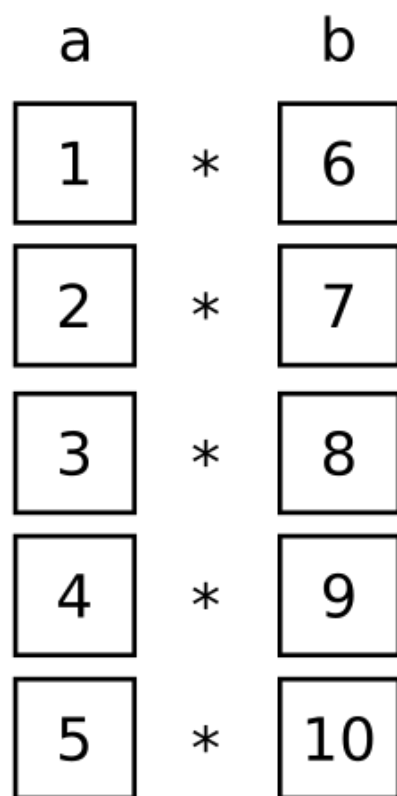
■ Algorithm / Data Structure

	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

■ Vectorization

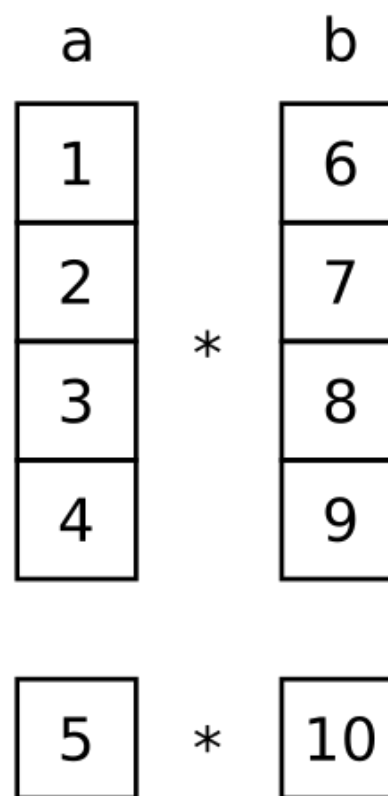
- Arrays are important because they enable you to express **batch operations** on data **without writing any for loops**. This is usually called vectorization. Any arithmetic operations between equal-size arrays applies the operation elementwise.
 - 벡터화하여 계산
- 실제 코딩의 양을 줄일뿐만아니라, 벡터 계산은 병렬 계산이 가능하기 때문에, Multi core 활용 가능
 - CPU 지원 ([*vector processor*](#))
 - <https://blogs.msdn.microsoft.com/nativeconcurrency/2012/04/12/what-is-vectorization/>
- <https://www.labri.fr/perso/nrougier/from-python-to-numpy/>
- NumPy는 싱글 코어와 대형 배열에 최적화된 라이브러리라는 한계가 존재
 - 실제로 배열의 크기가 100개 이내인 경우 NumPy는 순수 파이썬 구현 보다는 오히려 낮은 성능을 보임

not vectorized

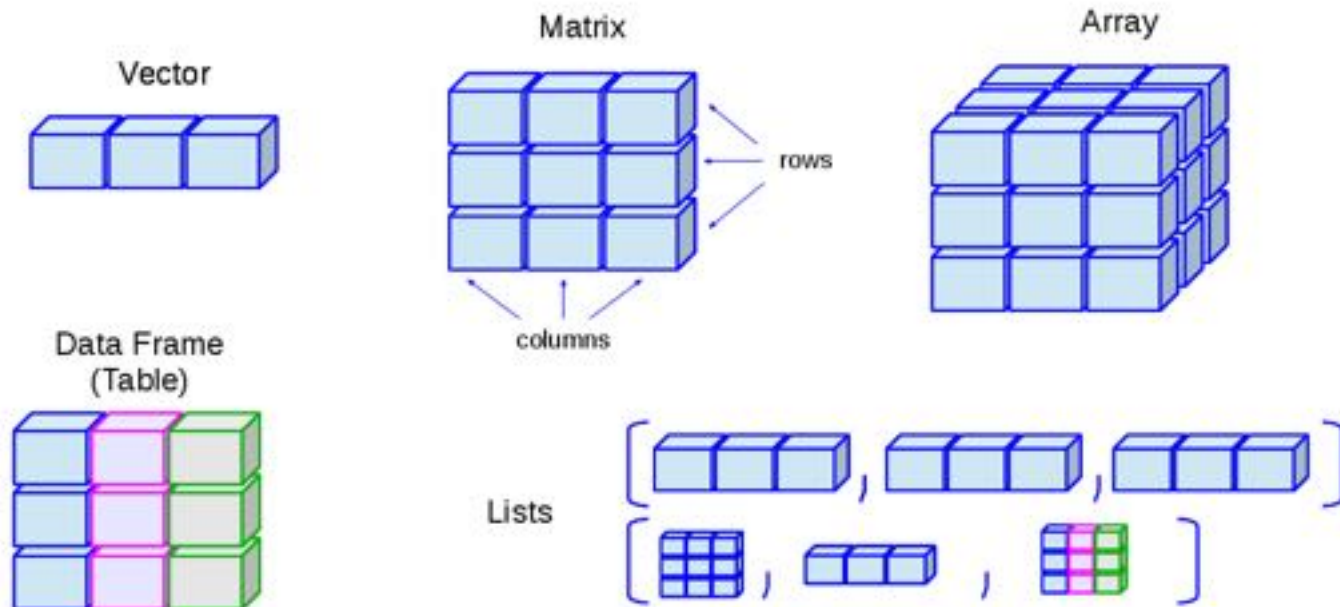


5 operations



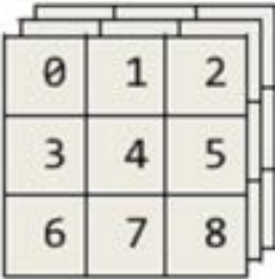
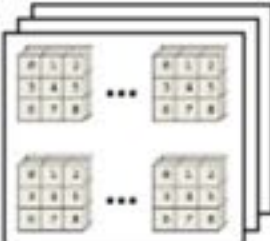
vectorized



2 operations



- "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		Vector
2		Matrix
3		3D Array (3 rd order Tensor)
N		ND Array

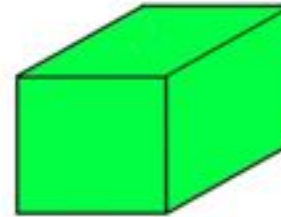
1D TENSOR /
VECTOR



2D TENSOR /
MATRIX

-9	4	2	5	7
3	0	12	8	61
1	23	-6	45	2
22	3	-1	72	6

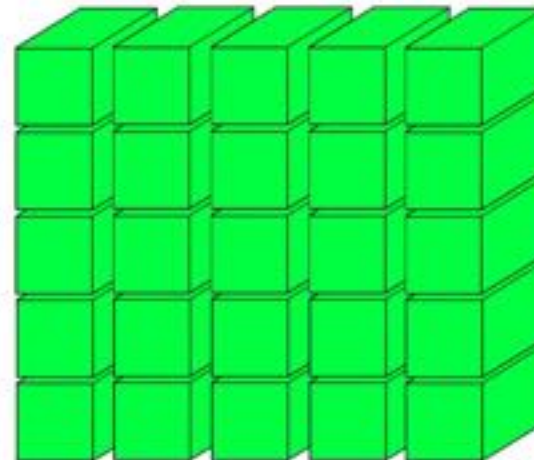
3D TENSOR /
CUBE



-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

■ *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)

■ 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]

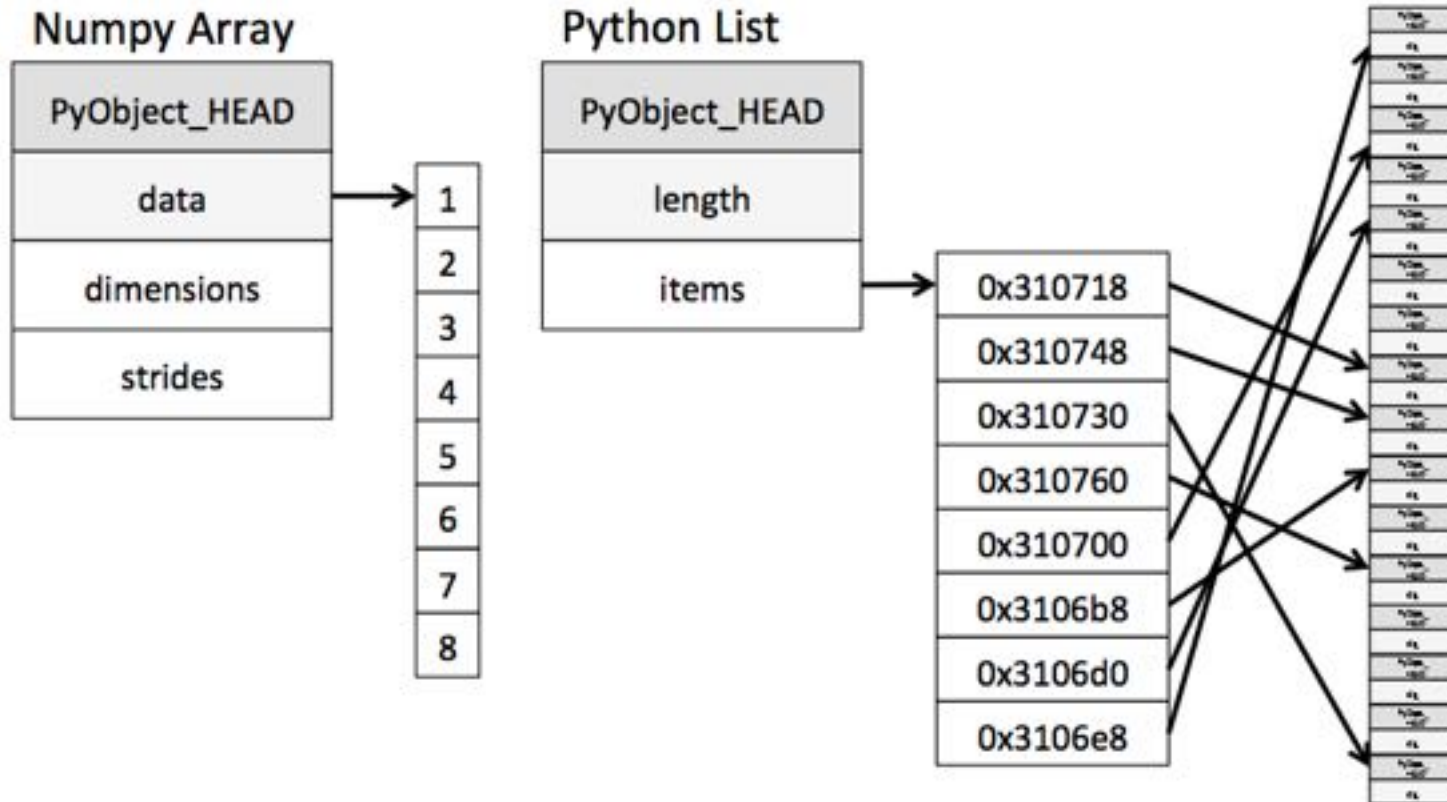
fft

random

linalg

NDArray
multi-dimensional
array object

UFunc
fast array
math operations

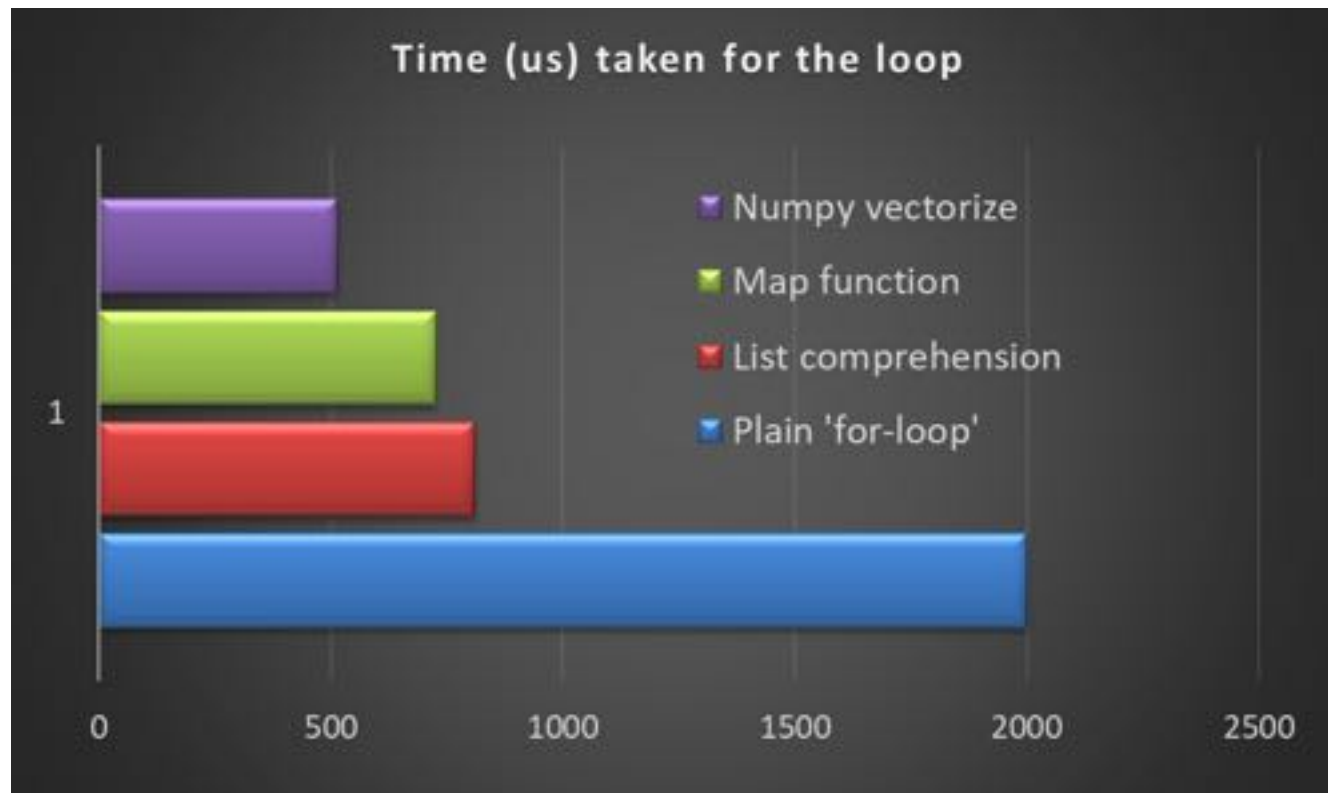


	ndarray	List
메모리 사용 특징	- 한 주소값으로 부터 연속적으로 메모리에 elem값저장	Elem값이 이산적인 주소값에 저장
연산 특징	sequential한 elem값 연산에 유리	Elem 단위 연산에 유리
구현	Array list (C/C++ array)	Linked list

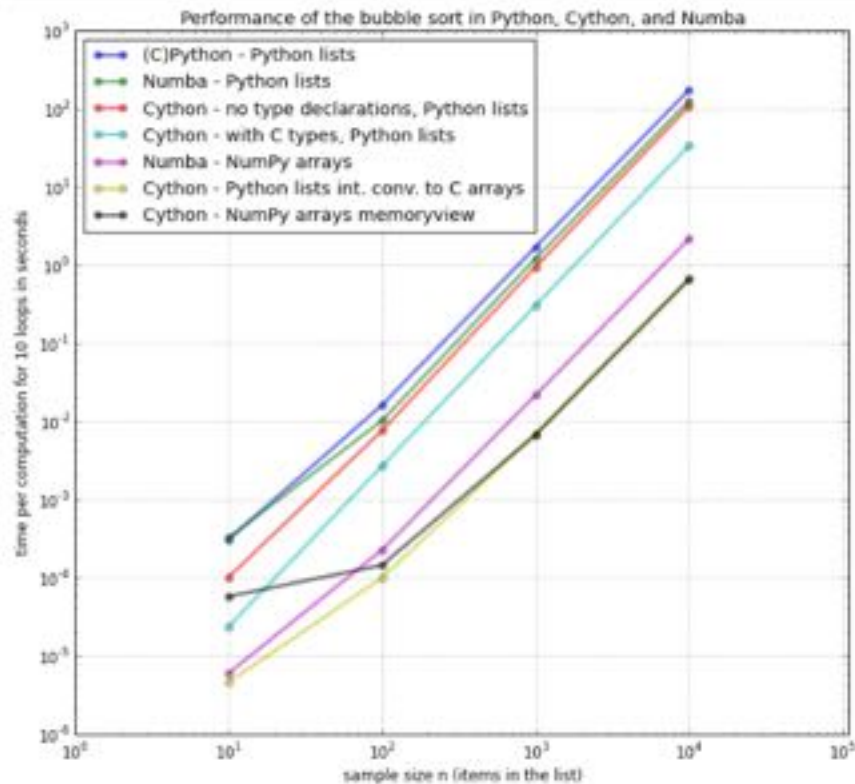
homogeneous

heterogeneous

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

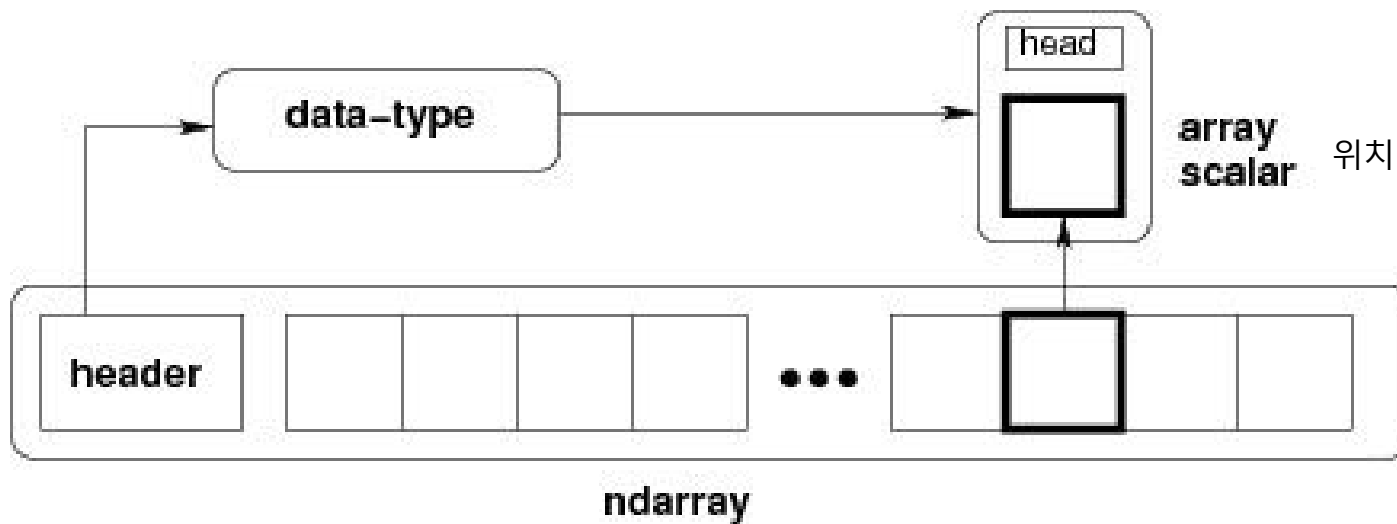


<https://www.codementor.io/tirthajyotisarkar/data-science-with-python-turn-your-conditional-loops-to-numpy-vectors-he1yo9265>

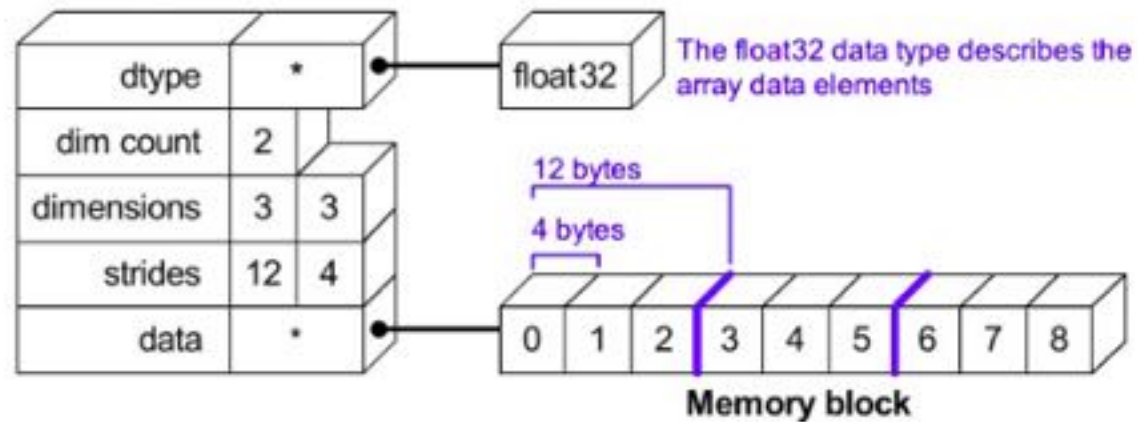


n=10000	bubblesort function	time in sec.	rel. performance gain
	(C)Python - Python lists	173.37	1.0
	Numba - Python lists	125.39	1.38
	Cython - no type declarations, Python lists	106.67	1.63
	Cython - with C types, Python lists	33.39	5.19
	Numba - NumPy arrays	2.16	80.39
	Cython - Python lists int. conv.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

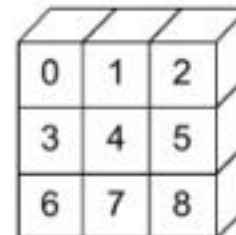
<https://stackoverflow.com/questions/23661636/poorer-performance-of-cython-with-numpy-array-memoryview-compared-to-c-arrays>



NDArray Data Structure



Python View :



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').

= 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

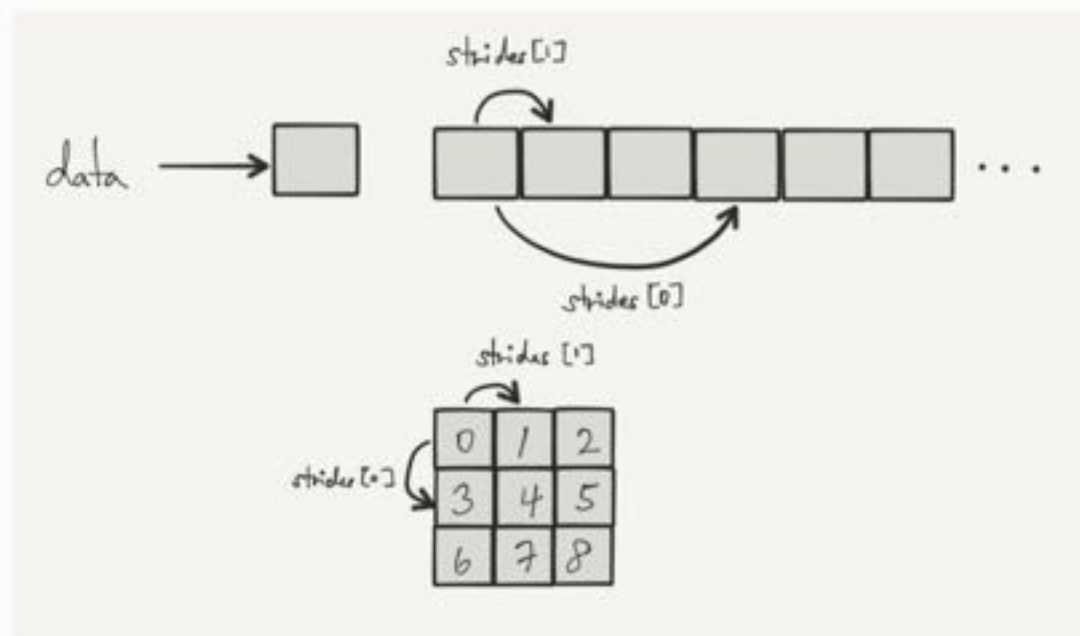
이 방식은 데이터의 최상위 비트가 가장 높은 주소에 저장되므로 그냥 보기에는 역으로 보인다. 빅 엔디안은 최상위 비트(MSB)부터 부호화되어 저장되며 예를 들면, 숫자 12는 2진수로 나타내면 1100인데 빅 엔디안은 1100으로 저장된다. 우측부터 저장

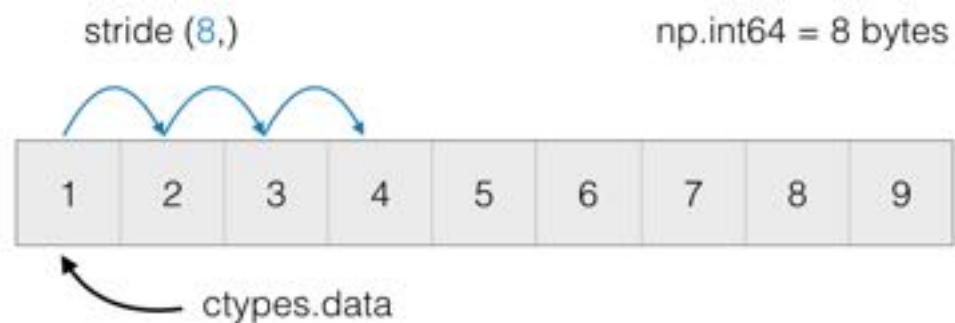
|: not-relevant

문자를 저장할 때 사용 endian가 상관없이 처리

itemsizes : 저장되는 메모리의 크기

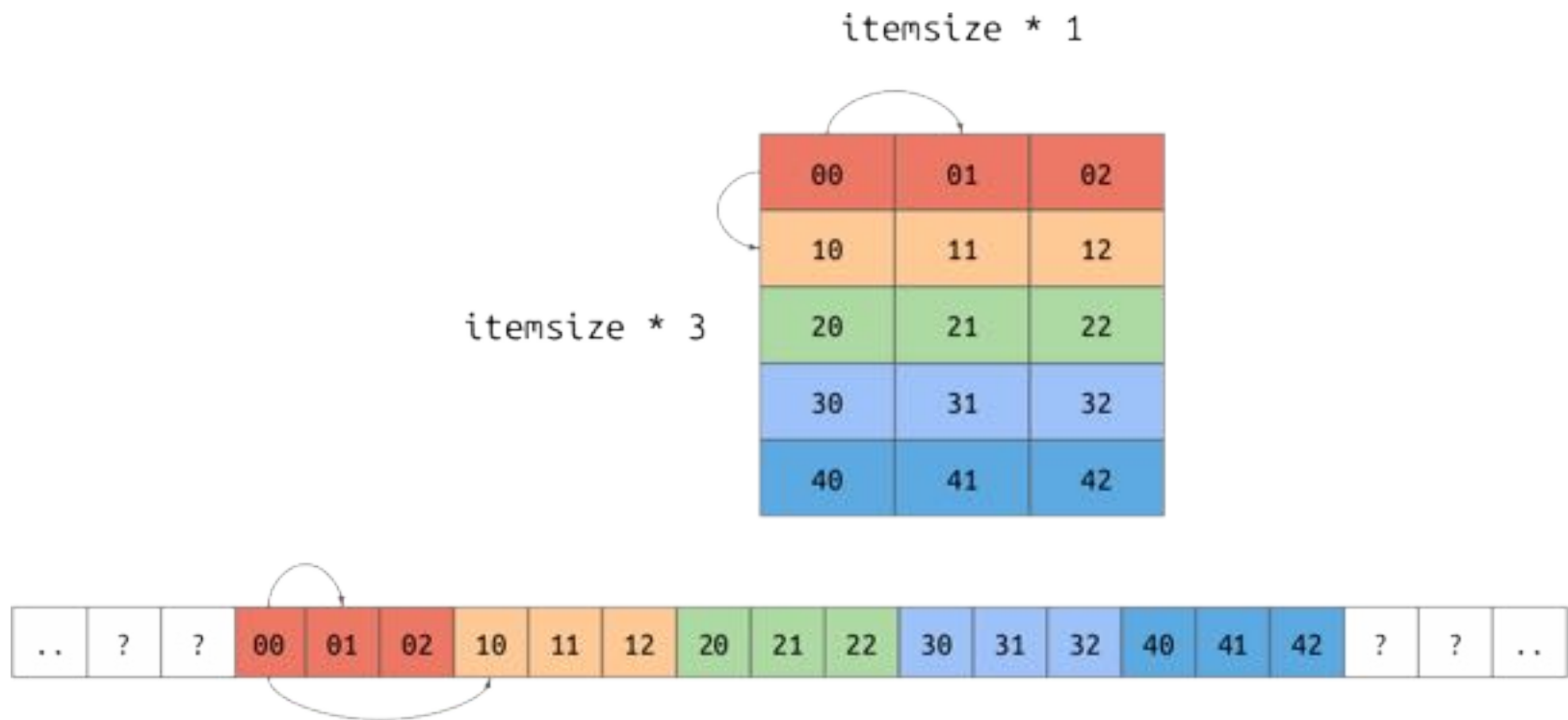
```
data : 4297514880  
shape : (3, 3)  
strides : (6, 2)  
dtype : uint16
```

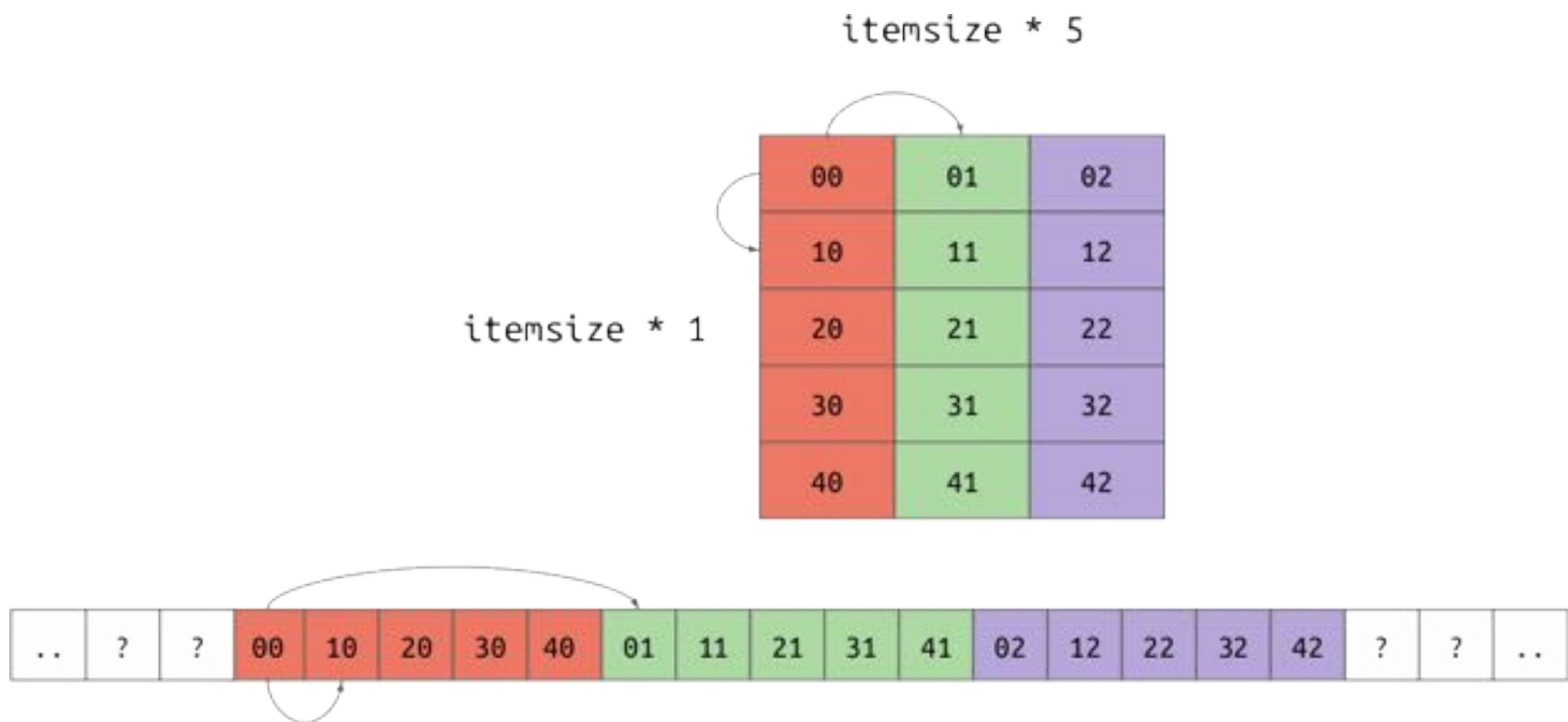




stride (12,4)
row to row
column to column



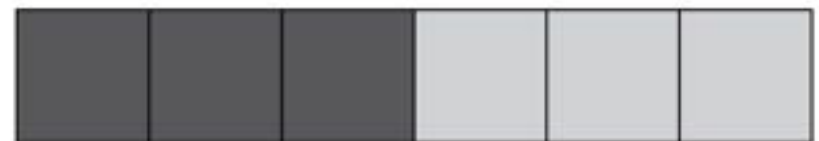
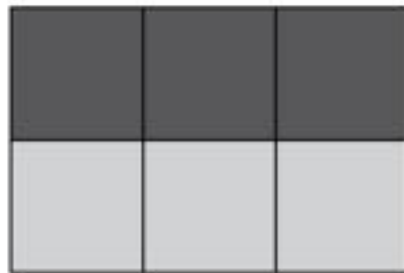




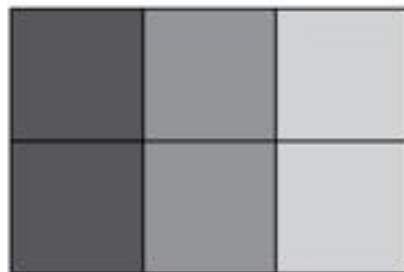
How the array is represented in Numpy

How the array is stored in memory

Row Major
Order (C)
(default in NumPy)



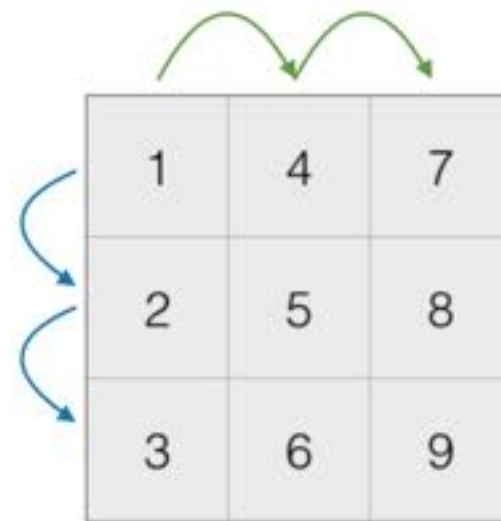
Column Major
Order (Fortran)





stride (12,4)

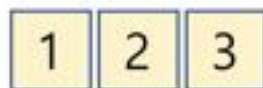
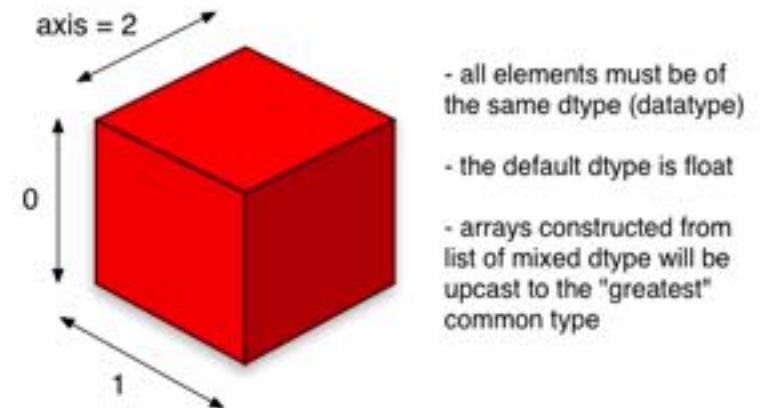
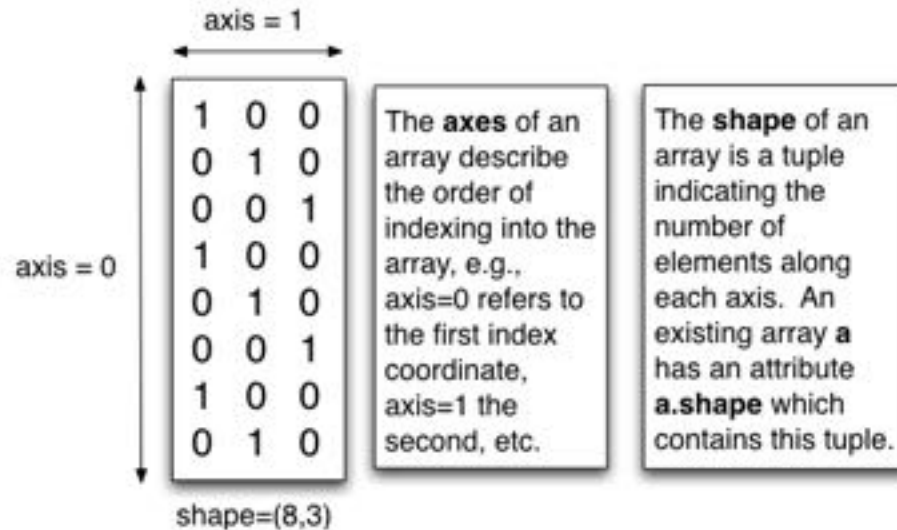
C



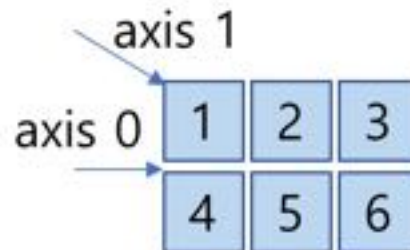
stride (4,12)

fortran

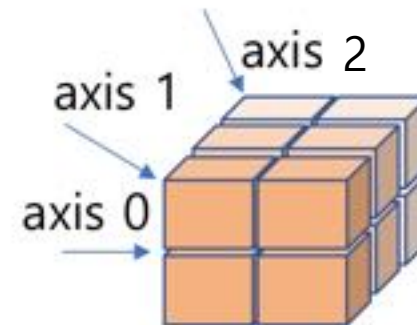
Anatomy of an array



1D array



2D array



3D array

```
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]])
```

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

```
=array([ 1.,  4.,  7., 10., 13.,  
       16., 19., 22., 25., 28.])
```

(10, ~~3~~)
axis=1

`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]]])
```

```
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.]])
```

```
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]])
```

```
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=1

(10,2,5)
axis=2

(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

@연산자 pep465

Function	Description
ndarray	생성자
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. Copies the input data by default.
asarray	Convert input to ndarray, but do not copy if the input is already an ndarray
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_like takes another array and produces a ones array of the same shape and 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 with any values like ones and zeros
eye, identity	Create a square N x N identity matrix (1's on the diagonal and 0's elsewhere)
linspace	<code>linspace(start, stop, num=50, endpoint=True, retstep=False)</code>

2. indexing / slicing



[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

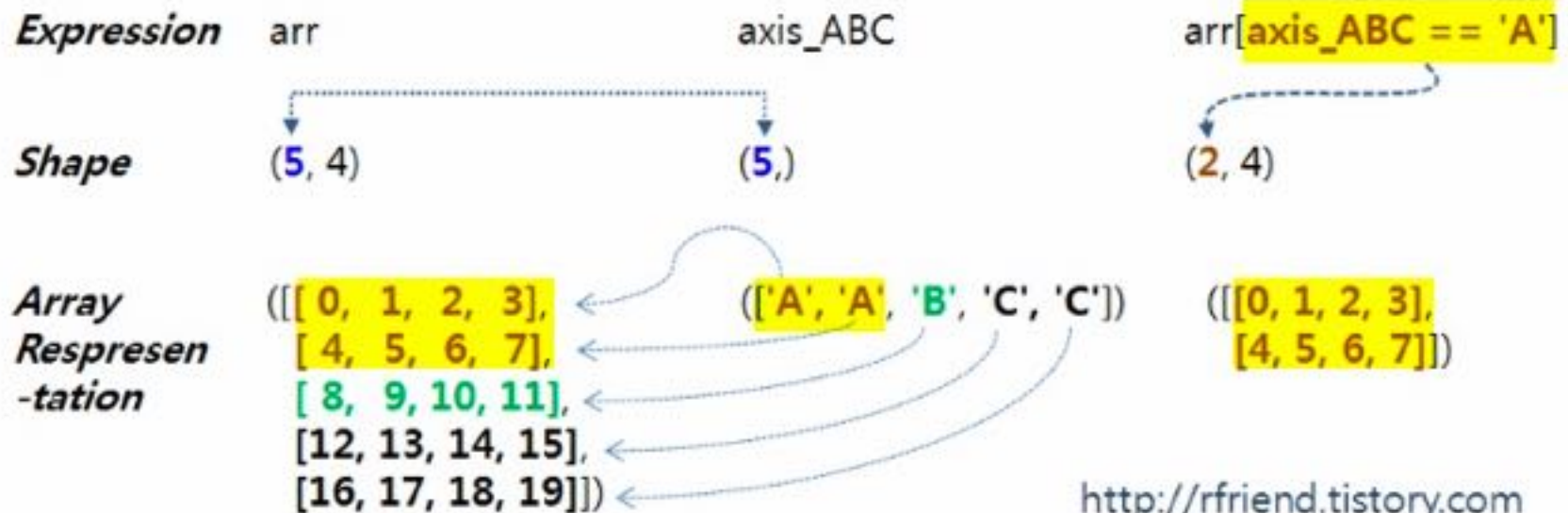
```
d = array([[ 0,  1,  2,  3,  4],  
          [ 5,  6,  7,  8,  9],  
          [10, 11, 12, 13, 14],  
          [15, 16, 17, 18, 19]])
```

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

```
array([[ 1,  2],  
       [ 6,  7],  
       [11, 12]])
```



[Python NumPy]

Slicing and Indexing with Boolean values



[Python NumPy]

Fancy Indexing by using integer arrays**copy****indexer***From the first*0
1
2
3
4*From the end*-5
-4
-3
-2
-1

ndarray a

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

selecting a subset of the rows*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	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11
4	12	13	14

selecting a subset 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]])
```

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


```
>>> 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]])
```

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

```
>>> 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]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

```
>>> 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])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

이름

```
dt = np.dtype([('a', np.str_, 10), ('b', np.float64)])
print(dt['a'])
print(dt['b'])
```

```
a = np.array([('aaa', 10000)], dtype=dt)
print(a)
print(a['a'])
print(a['b'])
```

```
import numpy as np
```

```
dict_type = {'names': ['col1', 'col2', 'col3'], 'formats': [ np
```

```
a = np.zeros(3, dtype=dict_type)
print(a)
print(a.dtype)
a['col1'] = (10, 22.3, 44.5)
print(a)
```

```
dict_type1 = {'surname': ('S25', 0), 'age': (np.uint8, 25)}
```

```
b = np.array([('dahl', 20), ('park', 30)], dtype=dict_type1)
print(b)
print(b["surname"])
```

```
[(0, 0.0, 0.0) (0, 0.0, 0.0) (0, 0.0, 0.0)]
[('col1', '<i4'), ('col2', '<f8'), ('col3', '<f8')]
[(10, 0.0, 0.0) (22, 0.0, 0.0) (44, 0.0, 0.0)]
[(b'dahl', 20) (b'park', 30)]
[b'dahl' b'park']
```

```
dt = np.dtype([('a', 'S10'), ('b', 'f8')])
print(dt['a'])
print(dt['b'])
```

```
a = np.array([('aaa', 10000)], dtype=dt)
print(a)
print(a['a'])
print(a['b'])
```

```
import numpy as np
```

```
x = np.zeros(3, dtype={'col1': ('i1', 0, 'title 1'), 'col2': ('f4', 1, 'title 2')})
```

```
print x.dtype.names
```

```
x.dtype.names = ('x', 'y') # 필드명 변경
```

```
('col1', 'col2')
```

```
import numpy as np

x = np.array([(1,2.,'Hello'), (2,3.,'World')],
             dtype=[('foo', 'i4'), ('bar', 'f4'), ('baz', 'S10')])
print x
```

```
print x[0]
print x[0]['foo']
print x[0]['bar']
print x[0]['baz']
```

```
print x['foo']
print x['bar']
print x['baz']
```

```
[(1, 2.0, 'Hello') (2, 3.0, 'World')]
(1, 2.0, 'Hello')
1
2.0
Hello
[1 2]
[ 2.  3.]
['Hello' 'World']
```

인덱스를 찾고
내부의 이름으로
검색

해당 이름에 해당되는 위치의
모든 값을
ndarray 타입으로 출력

```
import numpy as np
```

```
x = np.array([(1.5,2.5,(1.0,2.0)),(3.,4.,(4.0,5.)),(1.,3.,(2.,6.))], dtype=[('x', 'f4'), ('y', np.float32), ('value', 'f4'), (2,)]))
```

```
print x
print x[['x','y']]
```

```
print x[['x','value']]
```

```
[(1.5, 2.5, [[1.0, 2.0], [1.0, 2.0]]) (3.0, 4.0, [[4.0, 5.0], [4.0, 5.0]])
 (1.0, 3.0, [[2.0, 6.0], [2.0, 6.0]])]
[(1.5, 2.5) (3.0, 4.0) (1.0, 3.0)]
[(1.5, [[1.0, 2.0], [1.0, 2.0]]) (3.0, [[4.0, 5.0], [4.0, 5.0]])
 (1.0, [[2.0, 6.0], [2.0, 6.0]])]
```

```
import numpy as np
```

```
x = np.array([(1,2.,'Hello'), (2,3.,'World')],
             dtype=[('foo', 'i4'), ('bar', 'f4'), ('baz', 'S10')])
print x
```

```
y = x['foo']
print y
y[:] = 2*y
print y
print x['foo']
```

```
[(1, 2.0, 'Hello') (2, 3.0, 'World')]
[1 2]
[2 4]
[2 4]
```

3. broadcasting

■ 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 | 차원 개수가 작은 배열은 왼쪽 차원을 1로 채움 | 각각 짝이 되는 차원의 크기가 같으면 연산 가능 | 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 | 브로드캐스팅과 쉐일 (shape) 변환을 활용하면 루프를 피할 수 있음 • 뒤의 최단거리 이웃 예제 참조 • 고차원에서 생각하라



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'

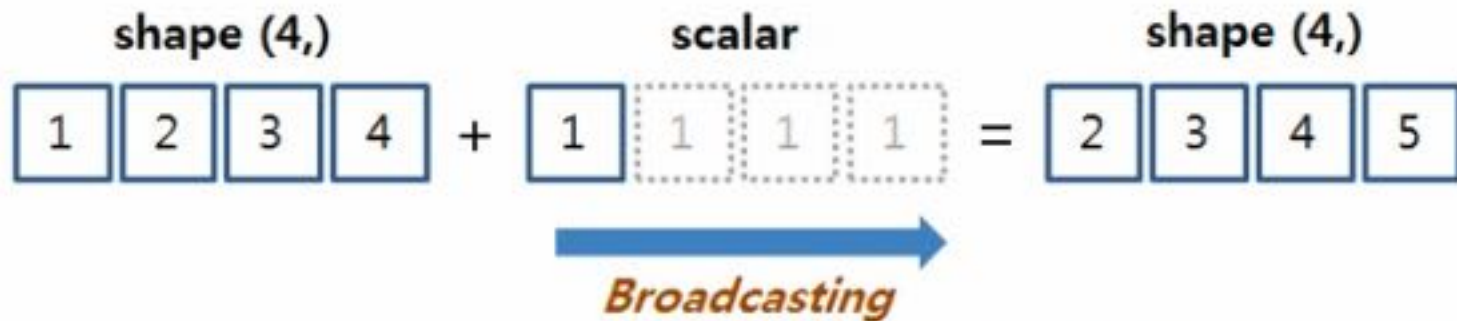
Operations
with **different-shape** arrays

broadcasting

<http://rfriend.tistory.com>

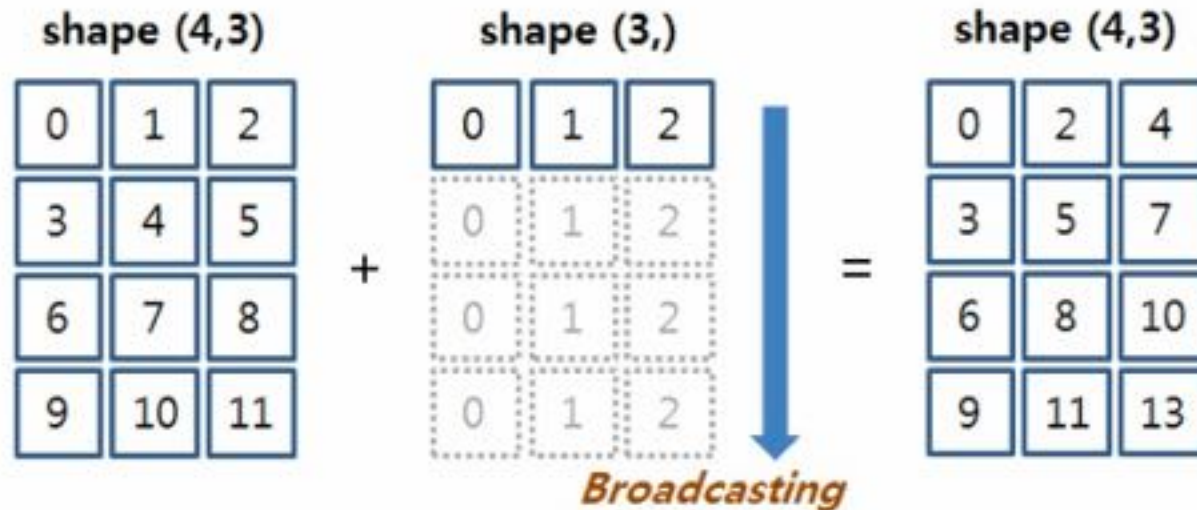


[Python NumPy]

Broadcasting over axis 1 with a Scalar<http://rfriend.tistory.com>



[Python NumPy]

Broadcasting over axis 0 with a 1D array

Arithmetic operation 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

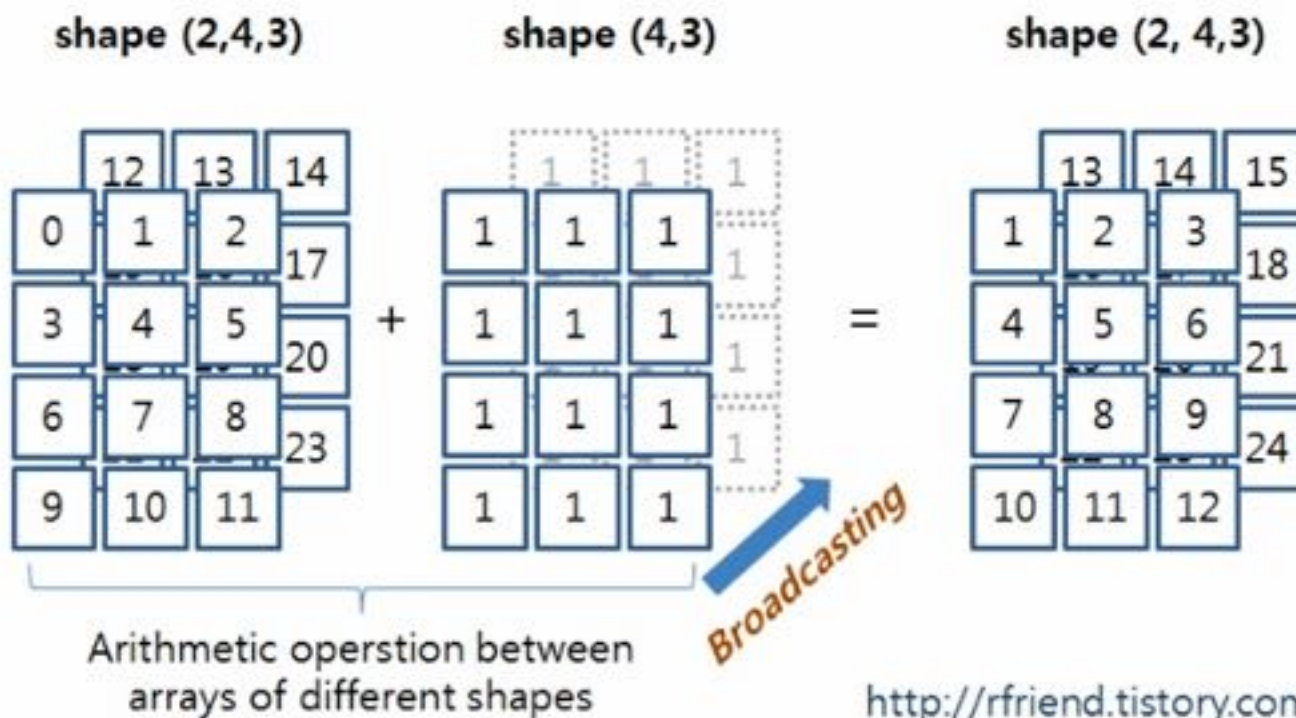
Arithmetic operation between
arrays of different shapes

<http://rfriend.tistory.com>



[Python NumPy]

Broadcasting over axis 0 with a 3D array



4. ufunc

■ 벡터화된 연산

- NumPy는 이미 컴파일된 코드로 만들어진 다양한 벡터화된 연산을 제공
- 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
- 모든 연산은 기본적으로 개별원소마다(elementwise)적용
- 파이썬에서 기본적으로 제공하는 함수와 섞어 쓰지 않을것

■ np.info()

■ 함수와 메소드를 동일하게 처리

- numpy 모듈에 함수, ndarray 내의 메소드가 이 중으로 지원하는 함수와 메소드가 많지만 용도에 맞춰사용 -> 메소드 사용을 권고

```
# 함수 정의
def add(self,x,y) :
    return x+y
# 클래스정의 및 메소드에 외부 함수 정의
class A(object) :
    add= add

a = A()
print a.add(5,5)
print type(a.add)
print add(a,5,5)
print type(add)
```

```
10
<type 'instancemethod'>
10
<type 'function'>
```



[Python NumPy] Universal Functions (or ufunc) : Fast Element-wise Array Functions

1개의 배열에 대한 ufunc 함수
: Unary universal functions

input `x = np.array([-2.1, 0, 1.5, 3.7])`

ufunc

`np.abs(x)`

output `array([2.1, 0., 1.5, 3.7])`

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.isnan(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)`

2개의 배열 간 ufunc 함수
: Binary universal functions

`m = np.array([0, 1, 2, 3])`

`n = np.array([1, 1, 2, 2])`

`np.add(m, n)`

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

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 ufuncs*)

```
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)
```


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 $\text{arr}^{**0.5}$
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 the dtype
modf	Return fractional and integral parts of array as separate array
isnan	Return boolean array indicating whether each value is NaN (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, arsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of not x element-wise. Equivalent to $\sim \text{arr}$.

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 argument
greater, greater_equal, less, less_equal, equal, not_equal	Perform element-wise comparison, yielding boolean array. Equivalent to infix operators $>$, $>=$, $<$, $<=$, $=$, $!=$
logical_and, logical_or, logical_xor	Compute element-wise truth value of logical operation. Equivalent to infix operators $\&$, $ $, $^$

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)
```

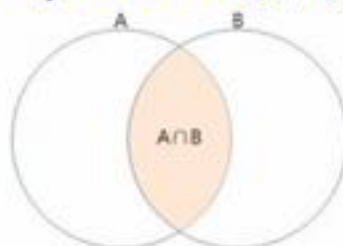
```
[0, 0, 0, 0, 0, 0, 1, 2, 3, 4]
```

<http://rfriend.tistory.com>

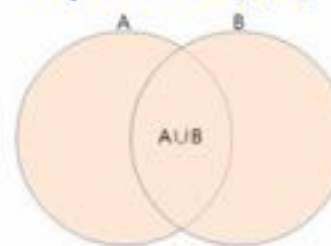


[Python NumPy] 집합 함수 (set functions)

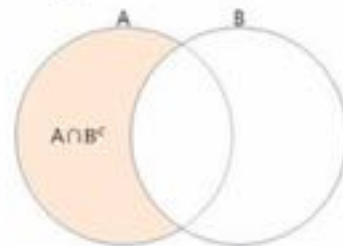
교집합 (intersect)
`np.intersect1d(A, B)`



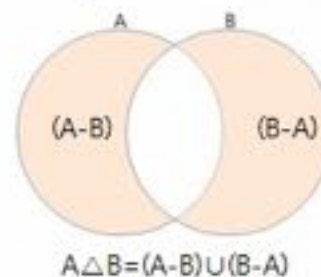
합집합 (union)
`np.union1d(A, B)`



차집합 (relative complement)
`np.setdiff1d(A, B)`



대칭차집합 (symmetric difference)
`np.setxor1d(A, B)`



<http://rfriend.tistory.com>

Method	Description
<code>unique(x)</code>	Compute the sorted, unique elements in <code>x</code>
<code>intersect1d(x, y)</code>	Compute the sorted, common elements in <code>x</code> and <code>y</code>
<code>union1d(x, y)</code>	Compute the sorted union of elements
<code>in1d(x, y)</code>	Compute a boolean array indicating whether each element of <code>x</code> is contained in <code>y</code>
<code>setdiff1d(x, y)</code>	Set difference, elements in <code>x</code> that are not in <code>y</code>
<code>setxor1d(x, y)</code>	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 콘텐츠에 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]

배열 분할하기 (split an array into sub-arrays)

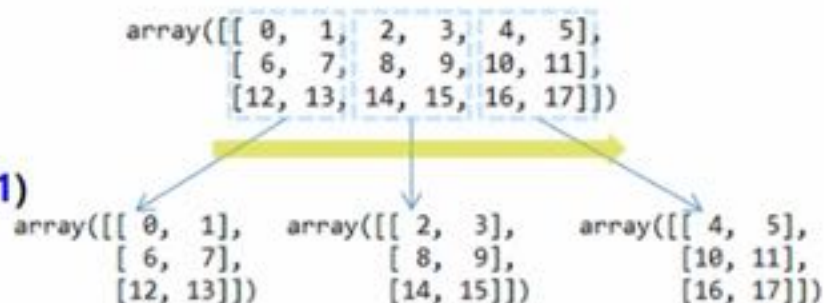
1 수평 축으로 배열 분할하기 (split array horizontally)

`np.hsplit(x, 3)`

`np.hsplit(x, (2, 4))`

`np.split(x, 3, axis=1)`

`np.split(x, (2, 4), axis=1)`



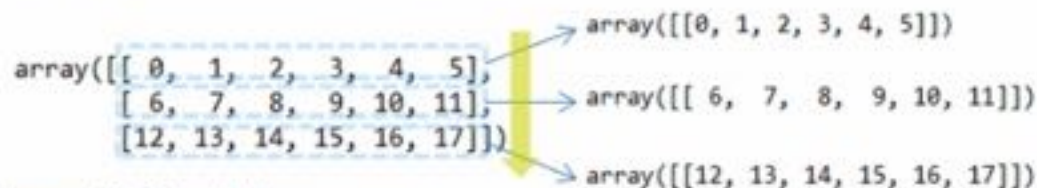
2 수직 축으로 배열 분할하기 (split array vertically)

`np.vsplit(x, 3)`

`np.vsplit(x, (1,2))`

`np.split(x, 3, axis=0)`

`np.split(x, (1, 2), axis=0)`



<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>



[Python NumPy]

배열을 옆으로, 위 아래로 붙이기

(concatenating array along the first/second axis,)

`np.r_[a, b]`

`np.hstack([a, b])`

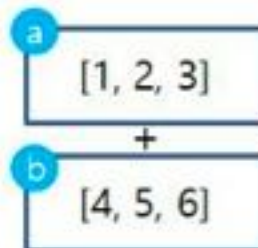
`np.concatenate((c, d),
axis=0)`



`np.r_[[a], [b]]`

`np.vstack([a, b])`

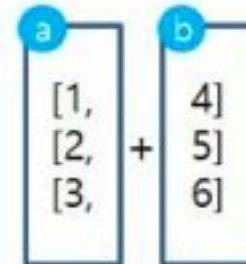
`np.concatenate(
(c, d), axis=1)`



`np.c_[a, b]`

`np.column_stack([a, b])`

`np.concatenate(
(c.T, d.T), axis=1)`



R, Python 분석과 프로그래밍의 친구 <http://rfriend.tistory.com>

`=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>



[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)

1

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)

2

np.sort(x2, axis=0)

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

아래에서 위로 거꾸로 정렬
(from bottom to top, reverse)

3

np.sort(x2, axis=0)[::-1]

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

<http://rfriend.tistory.com>

- 1. `np.append(a, b)` 또는 `np.concatenate`는 `a`와 `b`를 합친것을 `a`에 덮어씌우지않고 새로 합쳐진 결과를 반환한다.
- `a= np.concatenate((np.array([1,2,3,4]), np.array([5,6,7,8])), axis=0)`
- `print(np.append(a,[1,2,3,4]))`
- `a= np.append(a,[1,2,3,4])`
`print(a)`
-
- `[1 2 3 4 5 6 7 8]`
- `[1 2 3 4 5 6 7 8 1 2 3 4]`
- 2. `np.concatenate`는 `axis` 필요. 기능적으로 `append`와 구분
- 3. `np.append(a,b)` 형식으로 사용하지 `a.append(b)`는 불가능 (리스트와는 다르다)
- 4. `np.append(list, list) -> list` , `np.append(list, numpy) -> numpy`



[Python NumPy]

배열을 파일로 저장(save array), 불러오기(load)

【 NumPy Array 】

✓ *single array*

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

np.save()

np.load()

✓ *several arrays*

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

```
y = np.array([5, 6, 7, 8, 9])
```

np.savez()

np.load()

np.savez_compressed()

np.load()

np.savetxt()

np.loadtxt()

【 File Format 】

Binary file in
NumPy format
(.npy)

a single file in
uncompressed
format (.npz)

a single file in
compressed
format (.npz)

Text file

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

<http://friendtistory.com>

- `reshape(-1, 2)` 혹은 `reshape(3, -1)` 처럼 `reshape()` 메소드 안에 '-1'이 들어가 있는 경우가 있습니다. 이때 **`reshape()`의 '-1'이 의미하는 바는, 변경된 배열의 '-1' 위치의 차원은 "원래 배열의 길이와 남은 차원으로 부터 추정"이 된다는 뜻입니다. (One shape dimension can be -1. In this case, the value is inferred from the length of the array and remaining dimensions.)**
- Reshape = view

■ 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.