

# NumPy/SciPy

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

[http://www.labri.fr/perso/nrougier/teaching\(numpy.100/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.

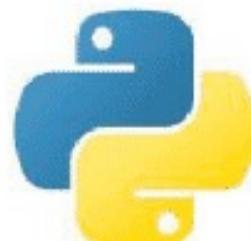
- ➊ Loved the expressive syntax of Python
- ➋ Loved the fact that slicing didn't make copies
- ➌ Loved the existing multiple data-types
- ➍ Loved how much more flexible it was to extend than MATLAB was
- ➎ Loved that I could read the source code and extend it

**General purpose?    2D?**



But....

Slowest things on earth:



## ■ 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

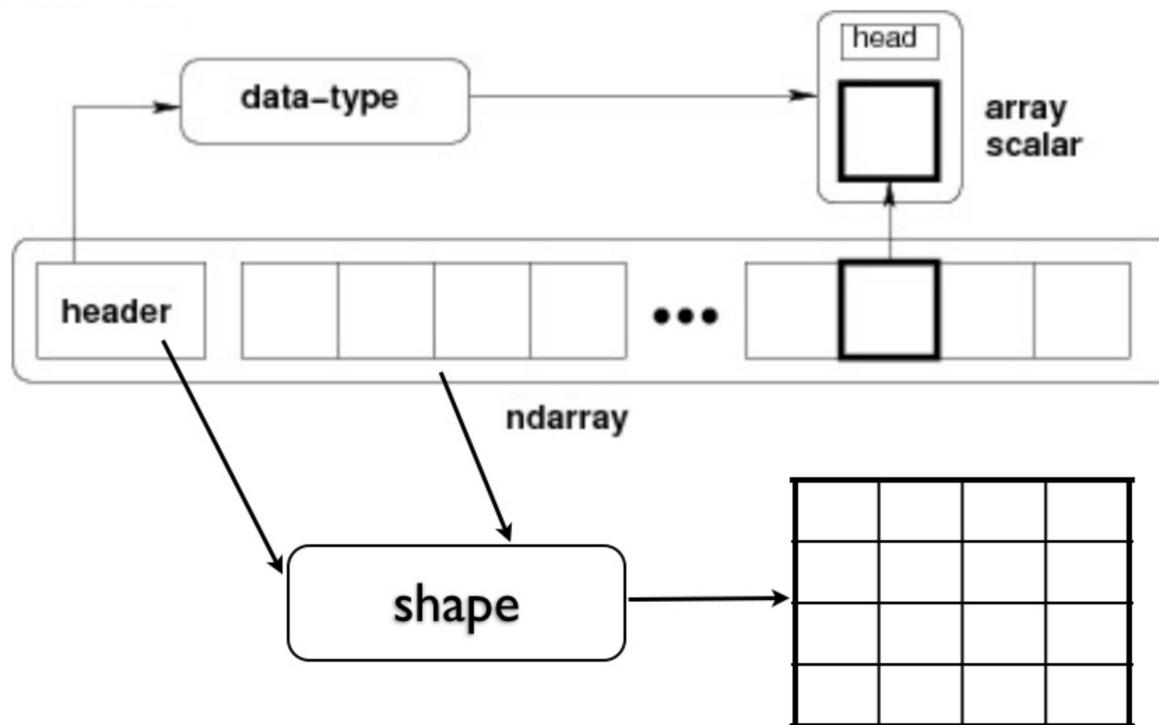
# NumPy

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

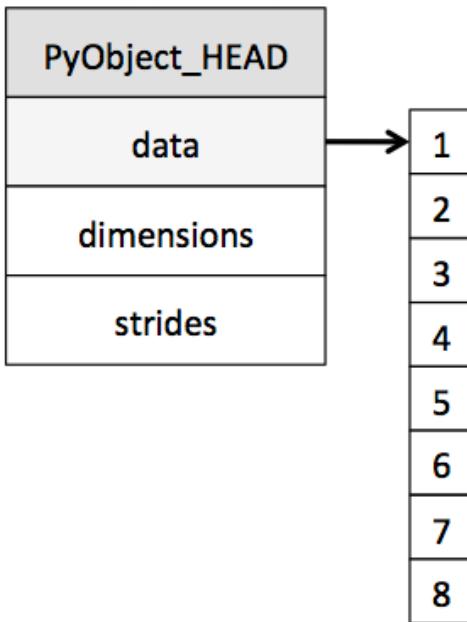
# ndarray

block of memory + indexing scheme + data type descriptor

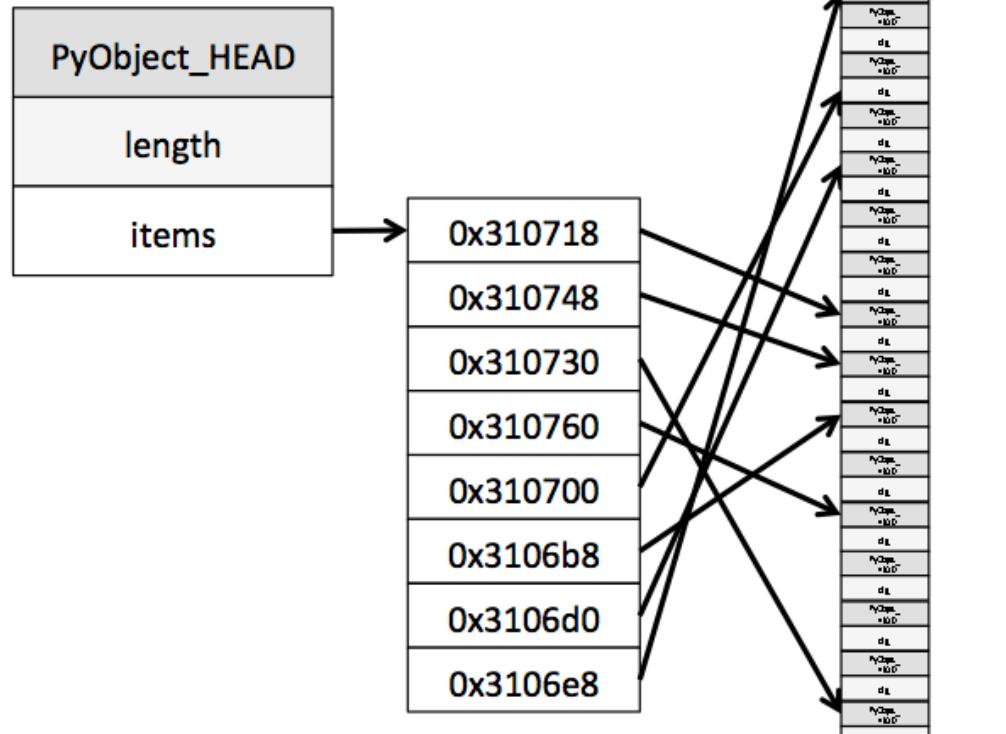
- raw data
- how to locate an element
- how to interpret an element



## Numpy Array



## Python List



	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

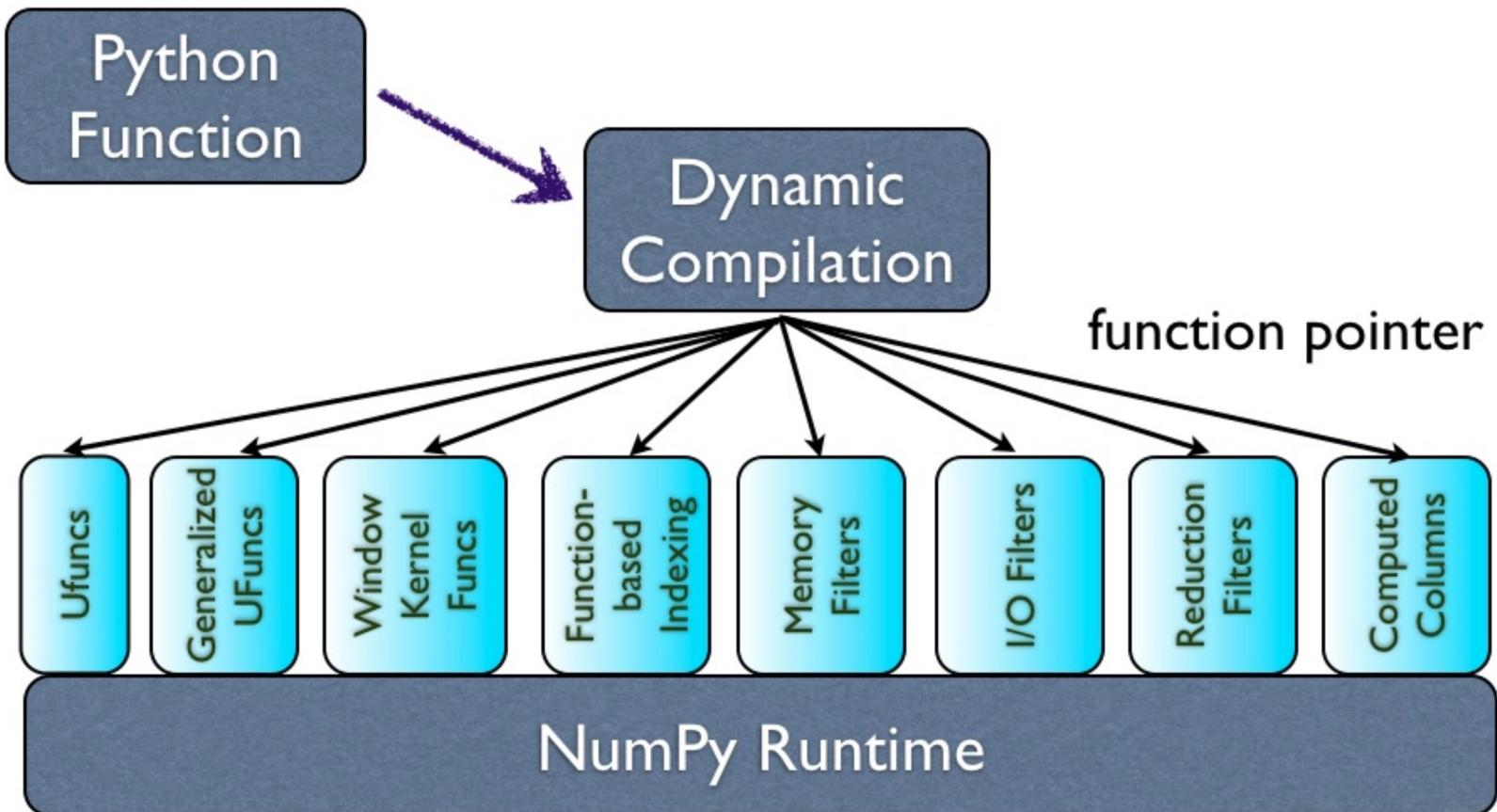
heterogeneous

list

포인터의 배열

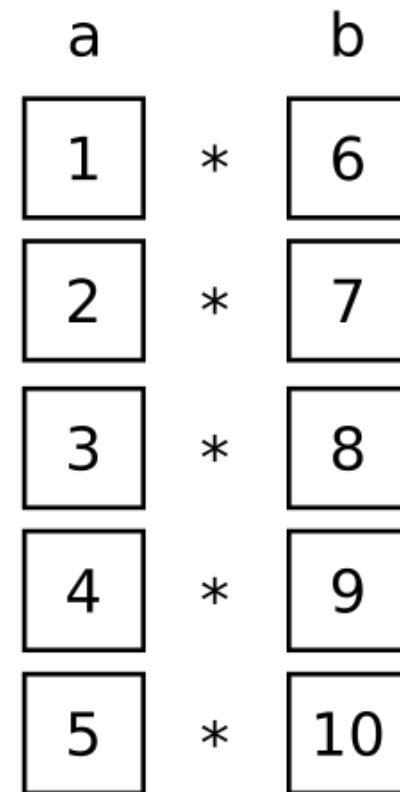
각각 객체가 메모리 여기저기 흩어져 있을 수 있음  
= 캐시 활용이 어려움

```
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;
```

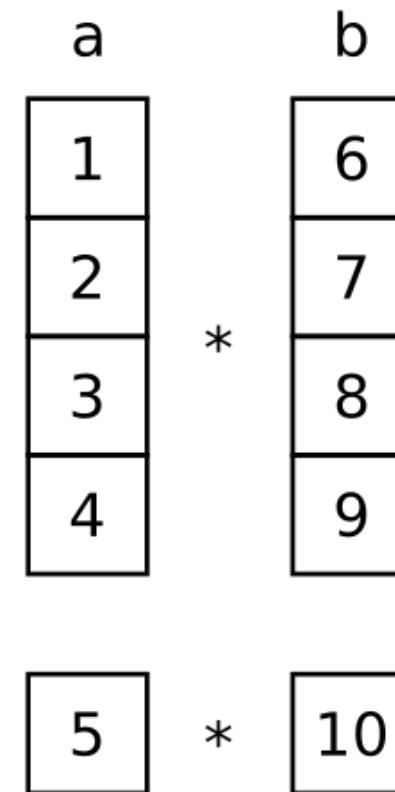


## ■ Vectorization

- 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 줄일뿐만 아니라, 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

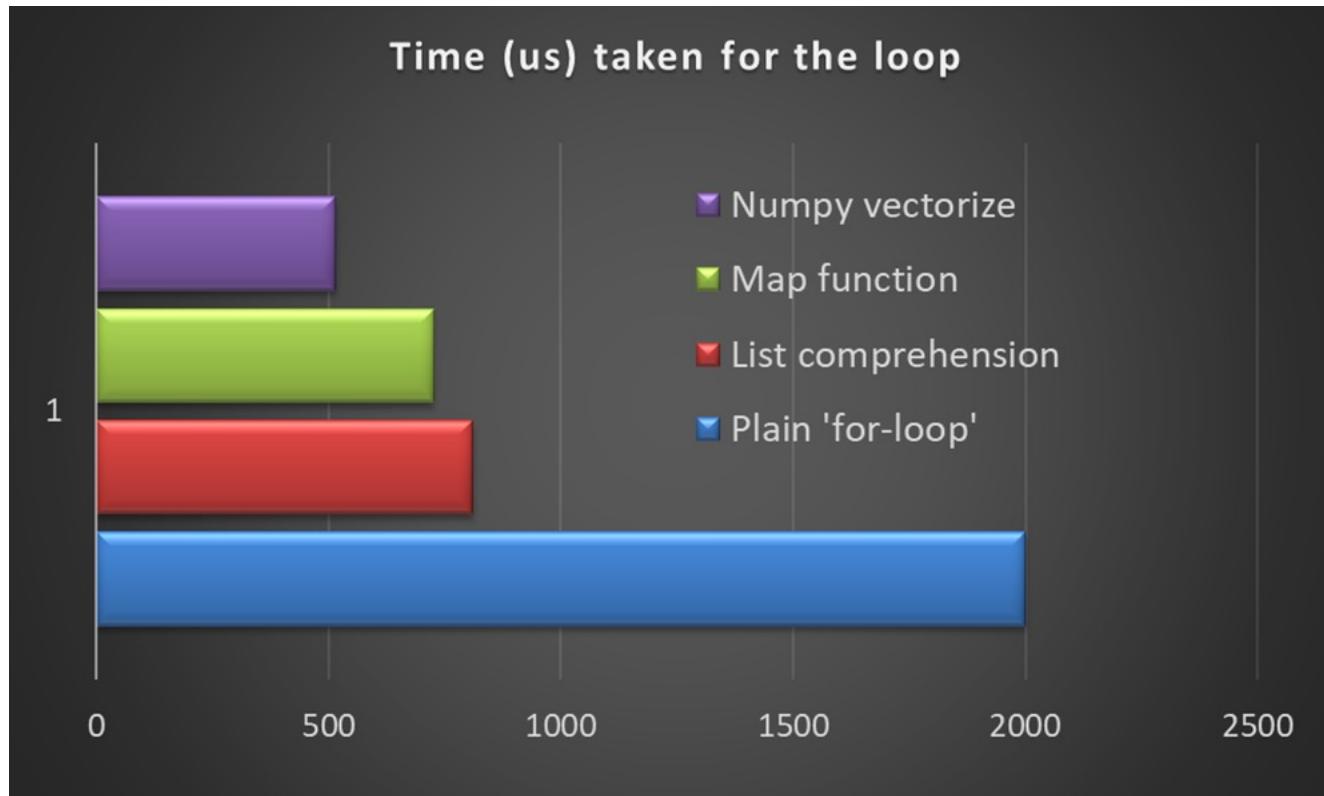
**vectorized**

2 operations

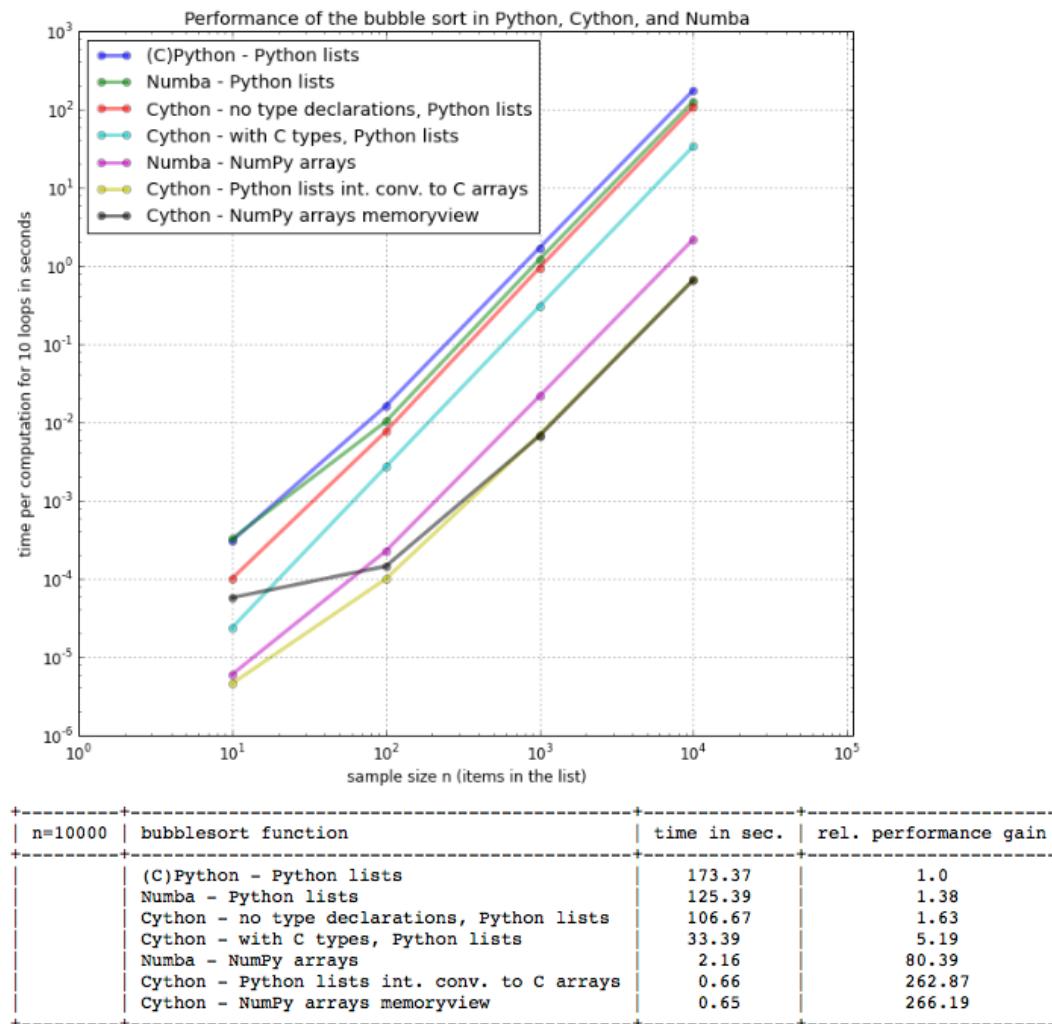
■ **stride**

■ **broadcasting**

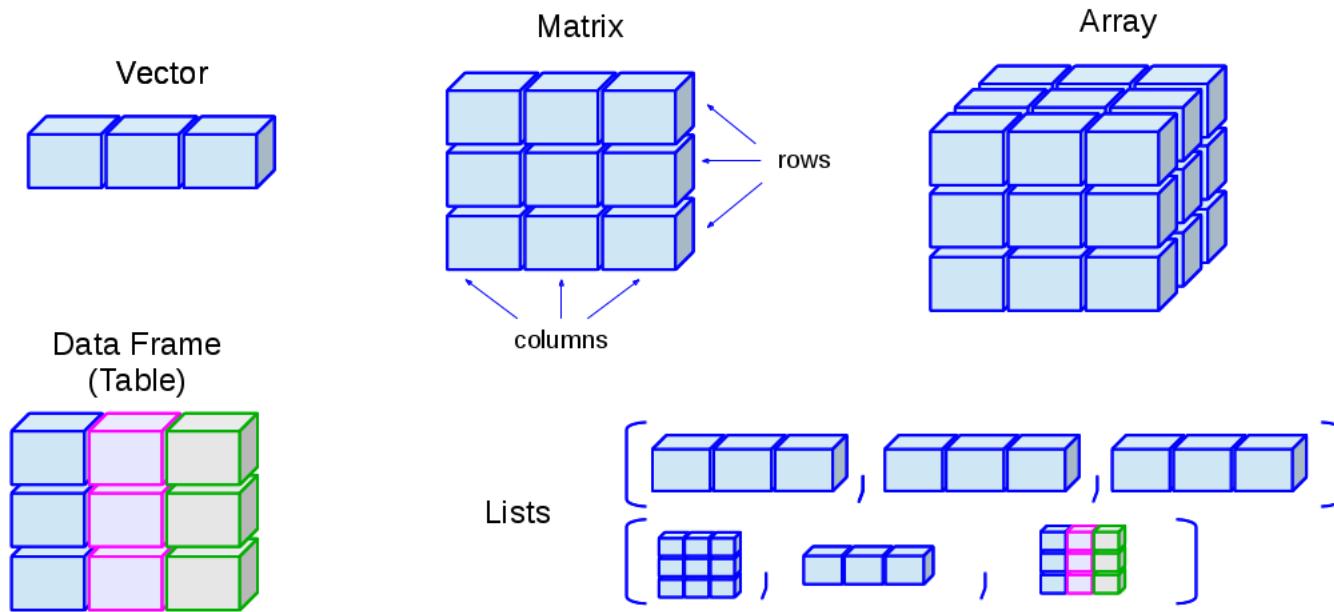
■ **ufunc**



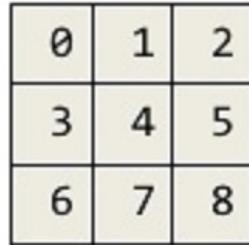
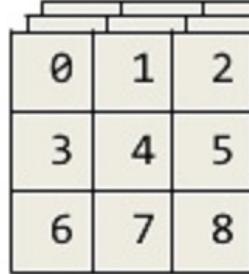
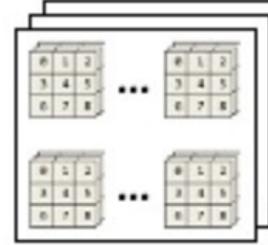
<https://www.codementor.io/tirthajyotsarkar/data-science-with-python-turn-your-conditional-loops-to-numpy-vectors-he1yo9265>



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



- "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 <sup>rd</sup> order Tensor)
N		ND Array

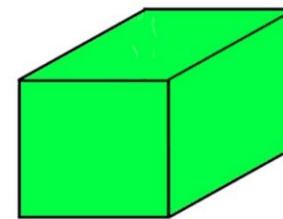
1 D TENSOR /  
VECTOR

5
7
4 5
1 2
- 6
3
2 2
1
6
3
- 9

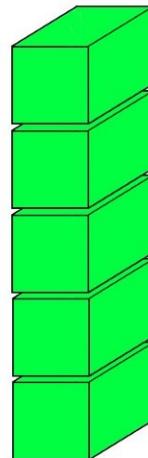
2 D TENSOR /  
MATRIX

- 9	4	2	5	7
3	0	1 2	8	6 1
1	2 3	- 6	4 5	2
2 2	3	- 1	7 2	6

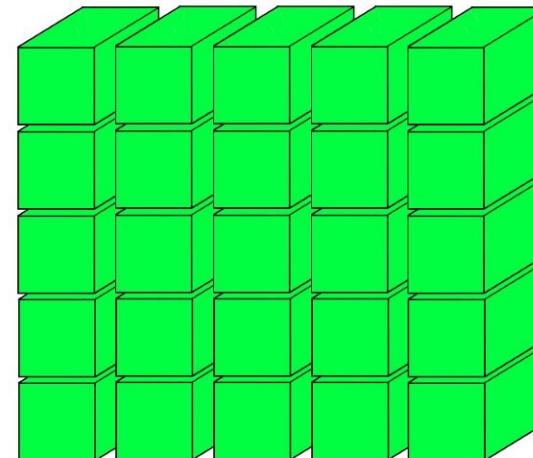
3 D TENSOR /  
CUBE



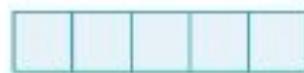
- 9	4	2	5	7
3	0	1 2	8	6 1
1	2 3	- 6	4 5	2
2 2	3	- 1	7 2	6



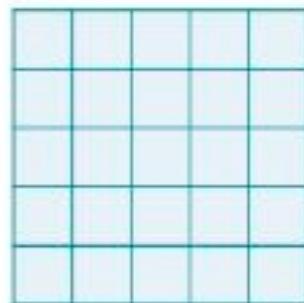
4 D TENSOR  
VECTOR OF CUBES



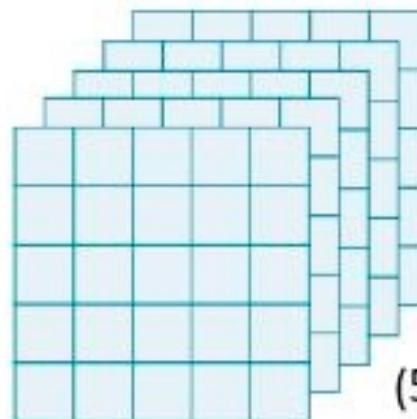
5 D TENSOR  
MATRIX OF CUBES



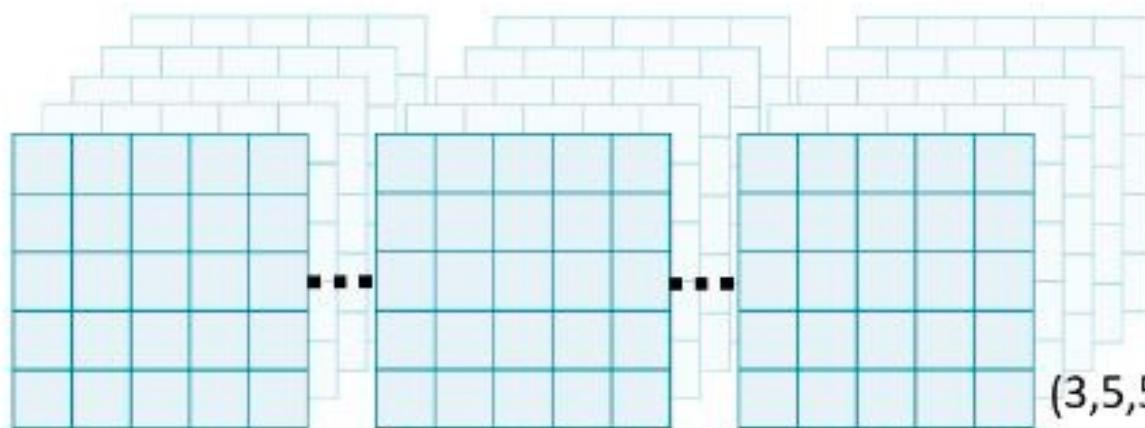
(5,)



(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*

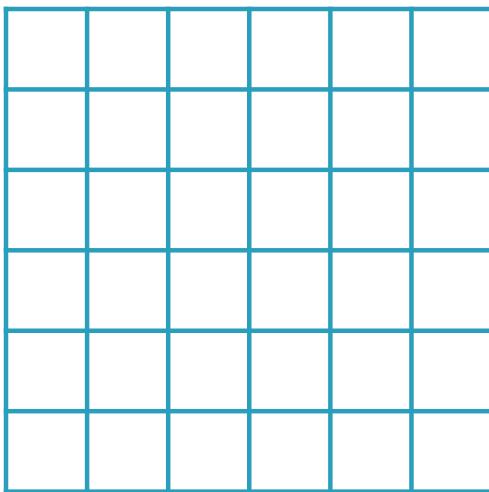
- 5D tensors of shape (samples, frames, height, width, channels) or (samples, frames, channels, height, width)

# Images as Matrices





RGB Images

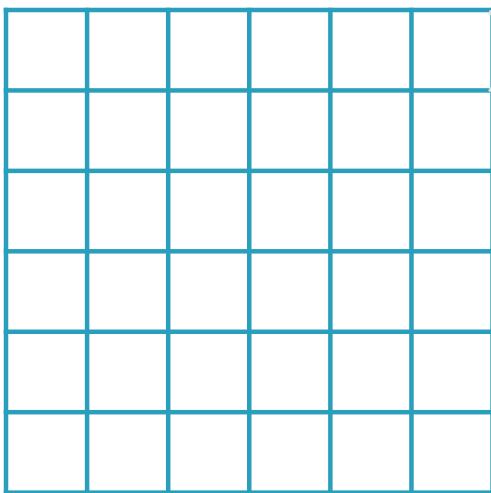


**RGB values are  
for color images**

**R, G, B: 0-255**



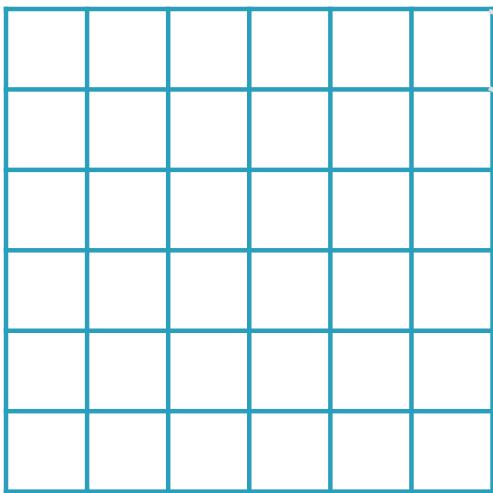
## RGB Images



**255, 0, 0**



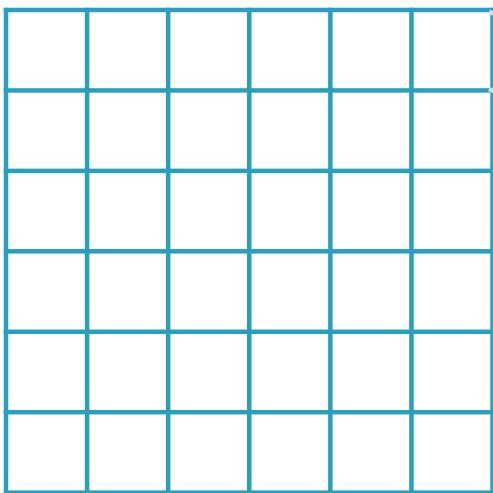
## RGB Images



0, 255, 0



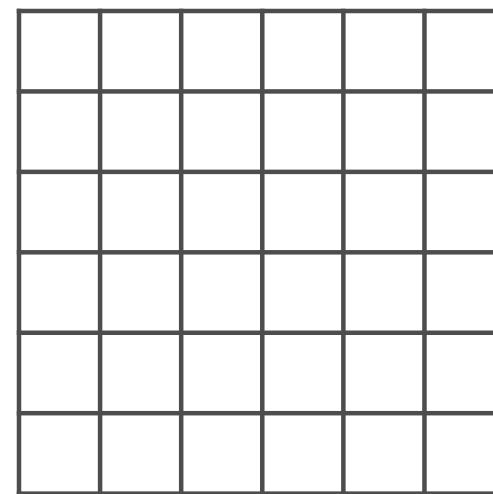
RGB Images



0, 0, 255

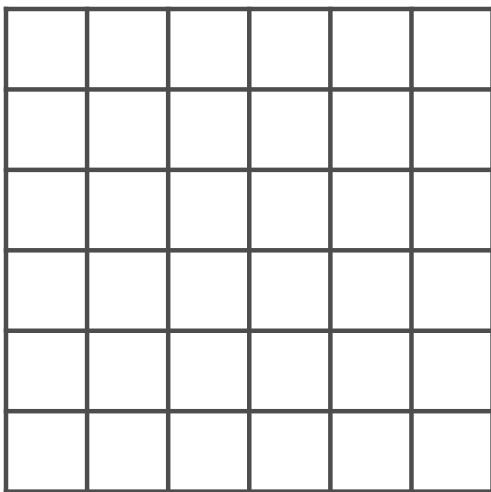
**3 values to represent  
color, 3 channels**

# Grayscale Images





## Grayscale Images

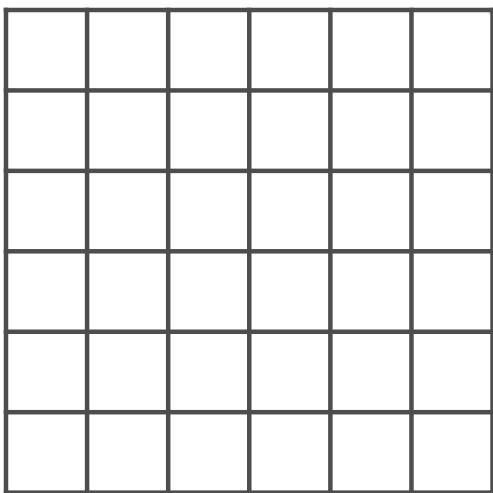


**Each pixel represents  
only intensity information**

0.0 - 1.0



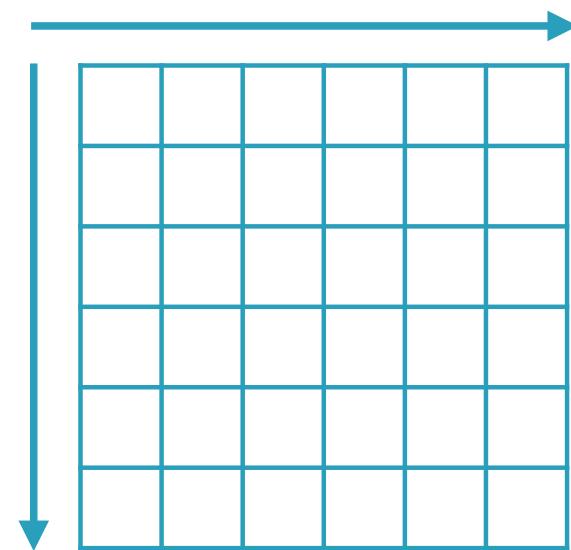
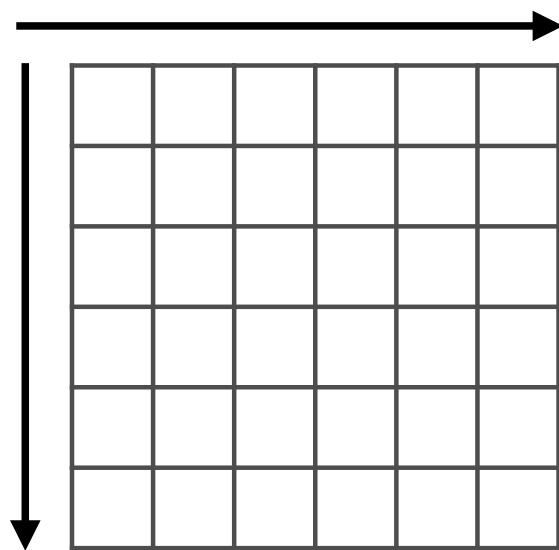
## Grayscale Images



0.5

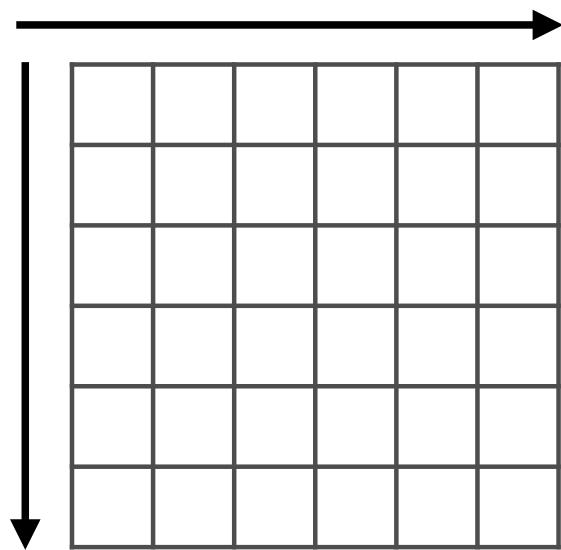
**1 value to represent  
intensity, 1 channel**

## Images as Matrices

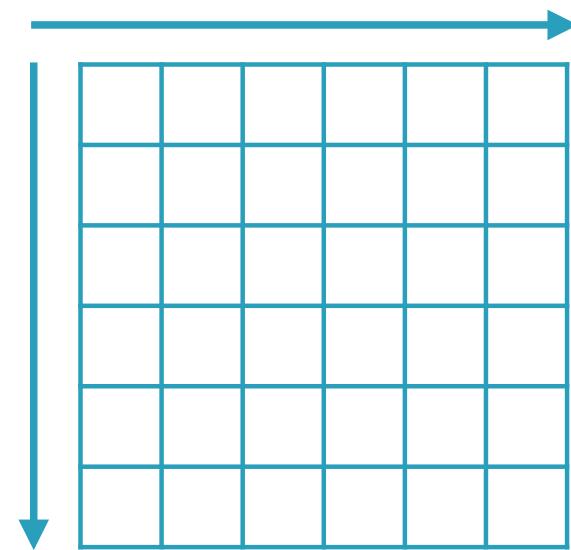
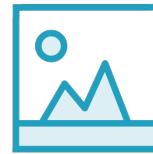


**Images can be represented by a 3-D matrix**

## Images as Tensors



**(6, 6, 1)**



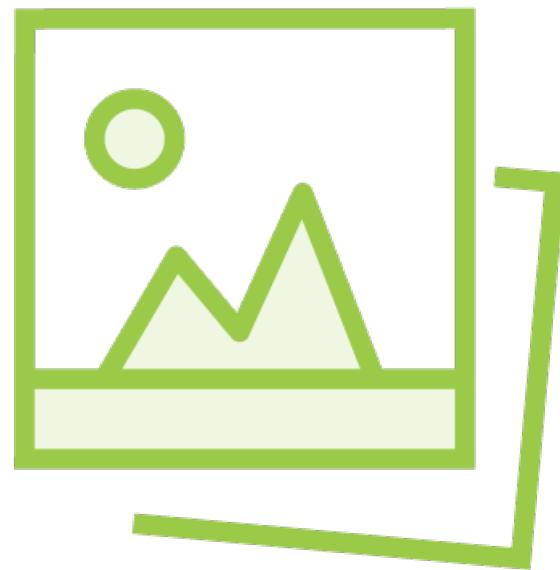
**(6, 6, 3)**

## List of Images



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

## List of Images



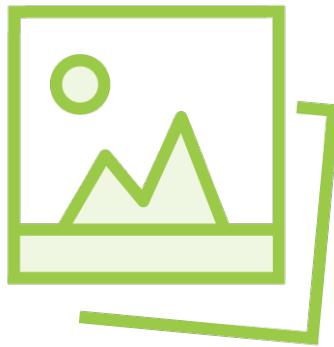
**The images should all be the same size**



List of Images

(10, 6, 6, **3**)

The number of channels



List of Images

(10, **6, 6,** 3)

**The height and width of  
each image in the list**



List of Images

(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

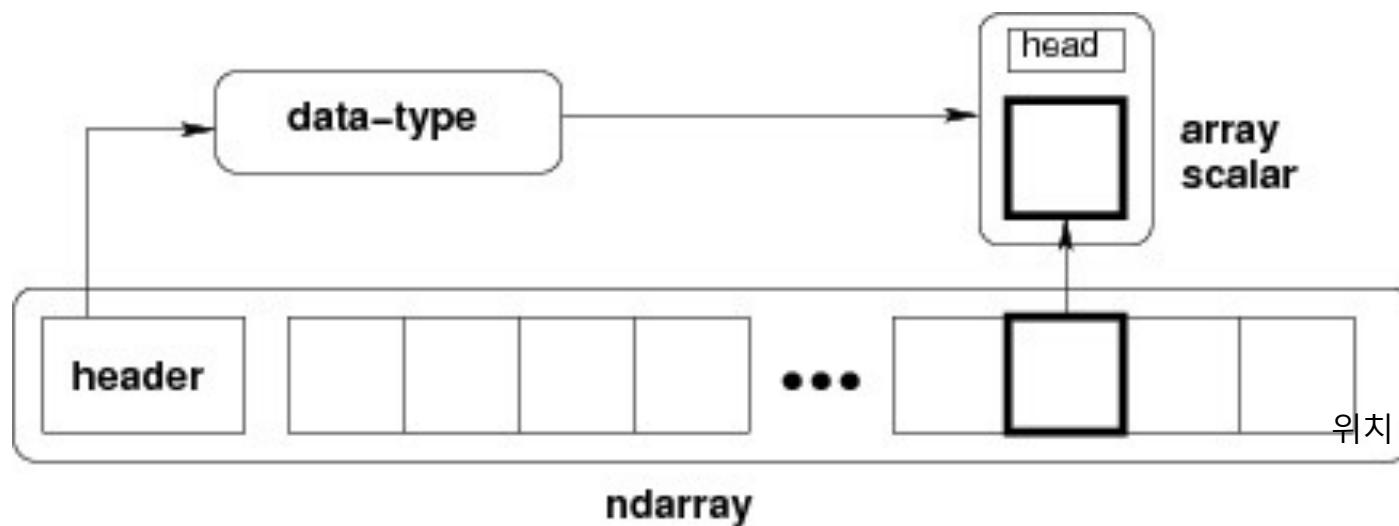
NDArray  
multi-dimensional  
array object

UFunc  
fast array  
math operations

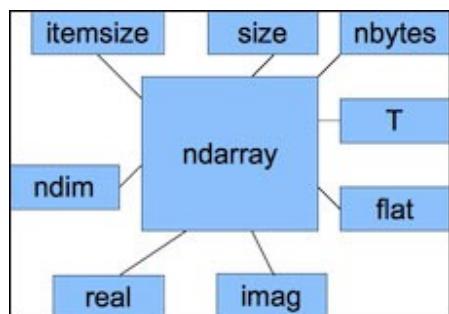
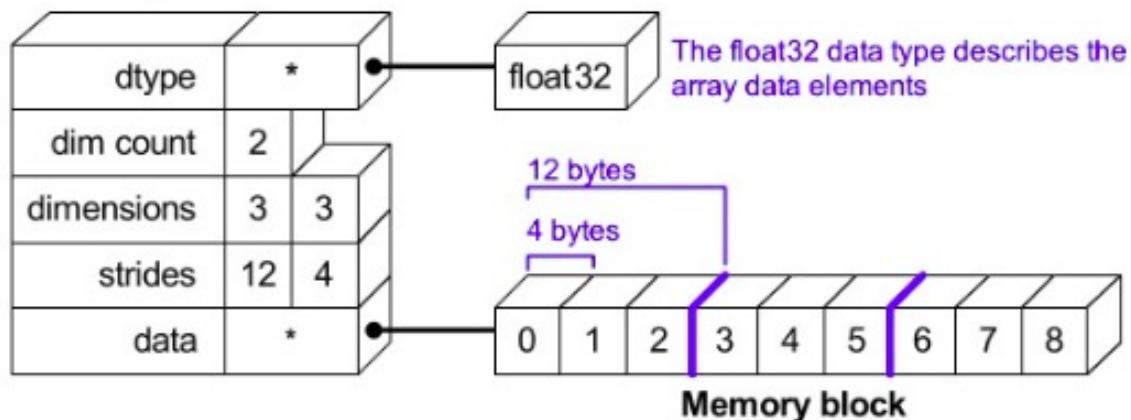
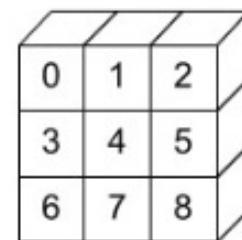
# ndarray

---

Function	Description
ndarray	constructor
array	Convert input data (list, tuple, array, or other sequence type) to an ndarray 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 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, casting, viewing, .astype(), .view()
- *strided indexing*: strides, C/F-order, slicing w/ integers, as\_strided, broadcasting, stride tricks, diag, CPU cache coherence

**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').

Common data type name	NumPy/pandas object	Pandas string name	Notes
Boolean	np.bool	<i>bool</i>	Stored as a single byte.
Integer	np.int	<i>int</i>	Defaulted to 64 bits. Unsigned ints are also available - np.uint.
Float	np.float	<i>float</i>	Defaulted to 64 bits.
Complex	np.complex	<i>complex</i>	Rarely seen in data analysis.
Object	np.object	<i>O, object</i>	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	<i>datetime64</i>	Specific moment in time with nanosecond precision.
Timedelta	np.timedelta64, pd.Timedelta	<i>timedelta64</i>	An amount of time, from days to nanoseconds.
Categorical	pd.Categorical	<i>category</i>	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 operations)

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

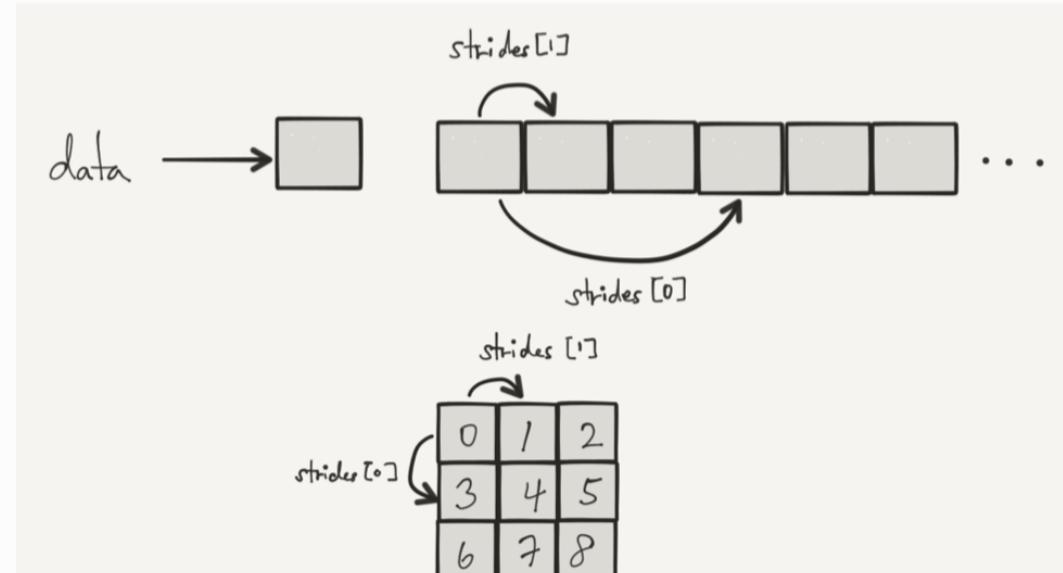
|: not-relevant

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
```

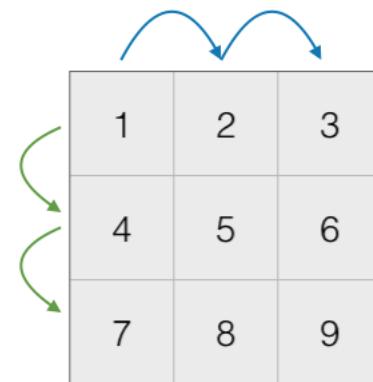


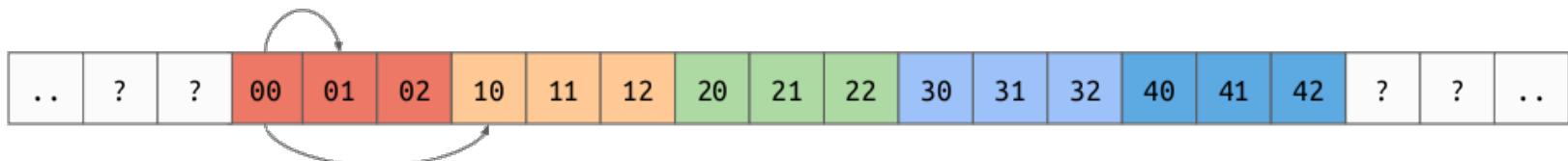
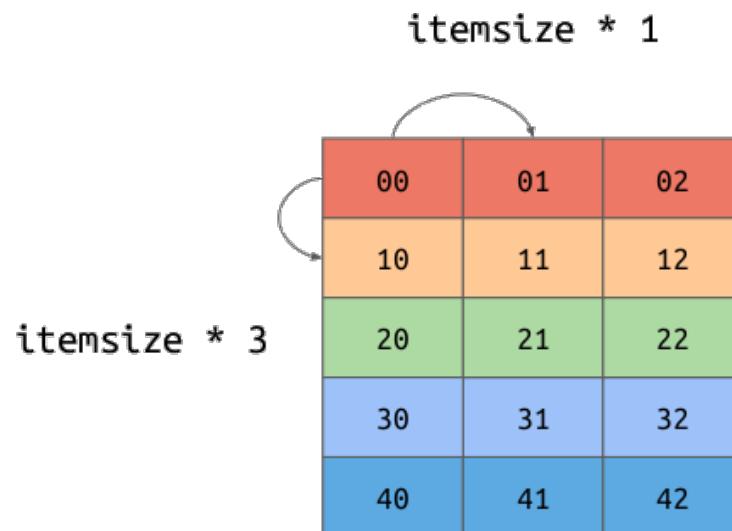
stride (8,)

np.int64 = 8 bytes



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

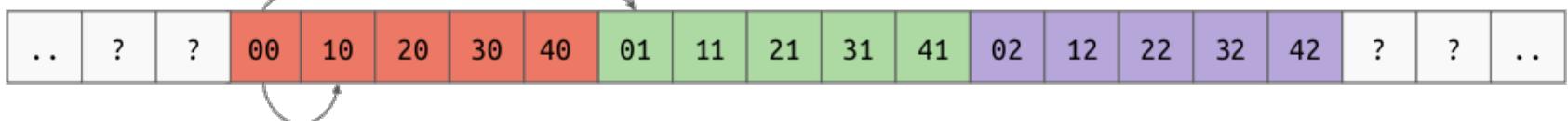




**itemsize \* 5**

00	01	02
10	11	12
20	21	22
30	31	32
40	41	42

**itemsize \* 1**



How the array is represented in Numpy

Row Major  
Order (C)  
(default in NumPy)

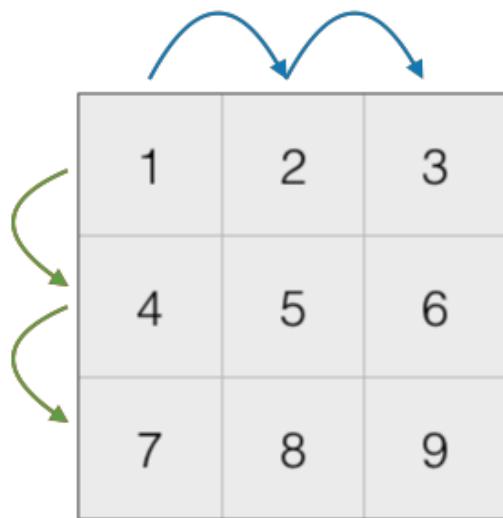


How the array is stored in memory



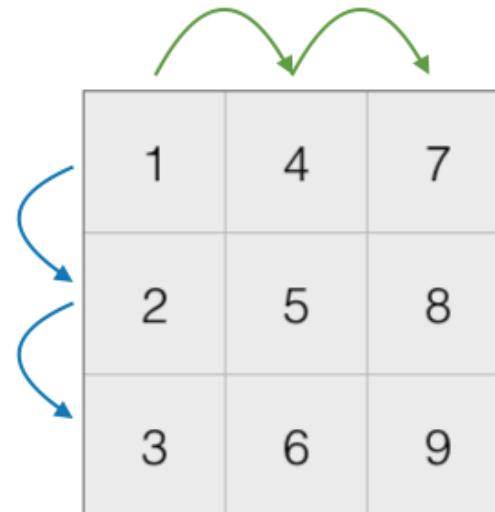
Column Major  
Order (Fortran)





stride (12, 4)

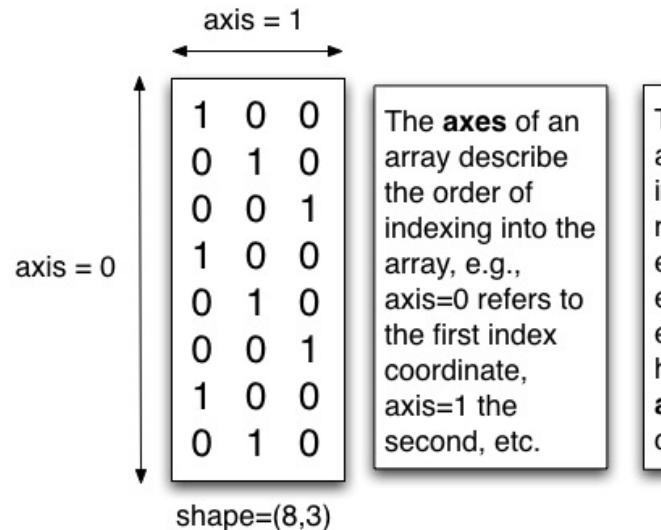
C



stride (4, 12)

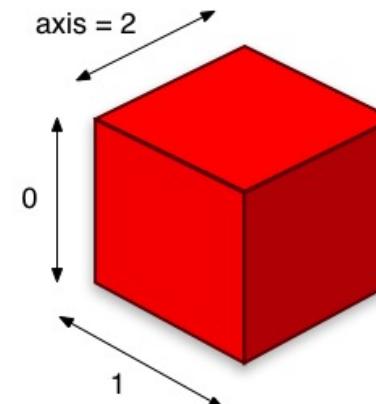
fortran

## Anatomy of an array



The **axes** of an array describe the order of indexing into the array, e.g., axis=0 refers to the first index coordinate, axis=1 the second, etc.

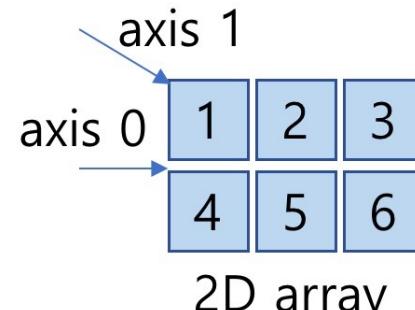
The **shape** of an array is a tuple indicating the number of elements along each axis. An existing array **a** has an attribute **a.shape** which contains this tuple.



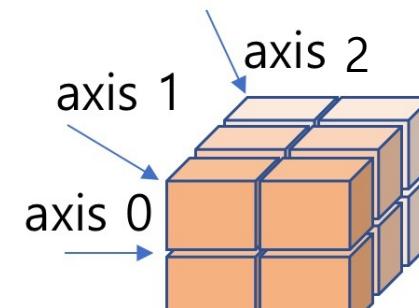
- all elements must be of the same dtype (datatype)
- the default dtype is float
- arrays constructed from list of mixed dtype will be upcast to the "greatest" common type



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],  
       [12, 13, 14],  
       [15, 16, 17],  
       [18, 19, 20],  
       [21, 22, 23],  
       [24, 25, 26],  
       [27, 28, 29]])
```

axis=1

(10,3)  
axis=1

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

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

axis  
0=axis

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
```

(10,2,5)  
axis=1

```
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]])
```

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.]])

(10,2,5)  
axis=2

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!!!*

### 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  
Represen-  
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'])

<http://rfriend.tistory.com>



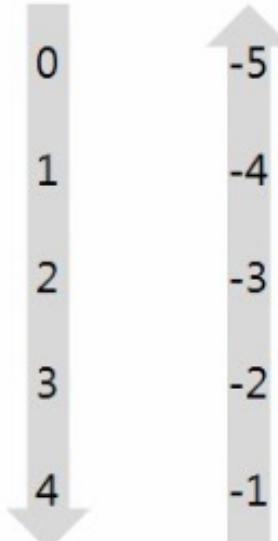
[ Python NumPy ]

## Fancy Indexing by using integer arrays

copy

### indexer

From the first      From the end



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



[ Python NumPy ]

## Fancy Indexing by using integer arrays

	axis 1 0	1	2
axis 0 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]])
```

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

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[0,3:5]  
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

<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	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	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

+

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

=



## ■ 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 짹이 되는 차원의 크기가 같으면 연산 가능 | 차원의 크기가 1이면 연산 가능 • 잡아 당겨서 같은 크기가 되도록 변환

■ 잡아당기는 연산은 명시적인 메모리 복사가 일어나지 않음 • 속도/메모리 이득 • 브로드캐스팅은 좋은 아이디어다 – 가능하면 사용하자 | 브로드캐스팅과 꼬 (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

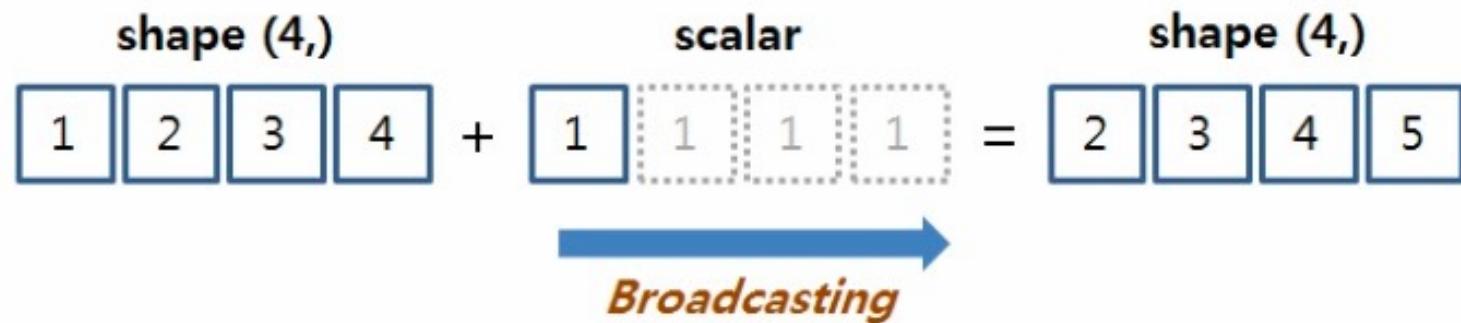
*broadcasting*

<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

shape (4,3)

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

shape (3,)

0	1	2
0	1	2
0	1	2
0	1	2

+

shape (4,3)

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

Broadcasting

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**

shape (2,4,3)

	12	13	14
0	1	2	17
3	4	5	20
6	7	8	23
9	10	11	

shape (4,3)

	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

+

shape (2, 4, 3)

	13	14	15
1	2	3	18
4	5	6	21
7	8	9	24
10	11	12	

**Broadcasting**Arithmetic operation 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 \***



## [ 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.acos(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)`



## [ 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 (*difference 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]))`



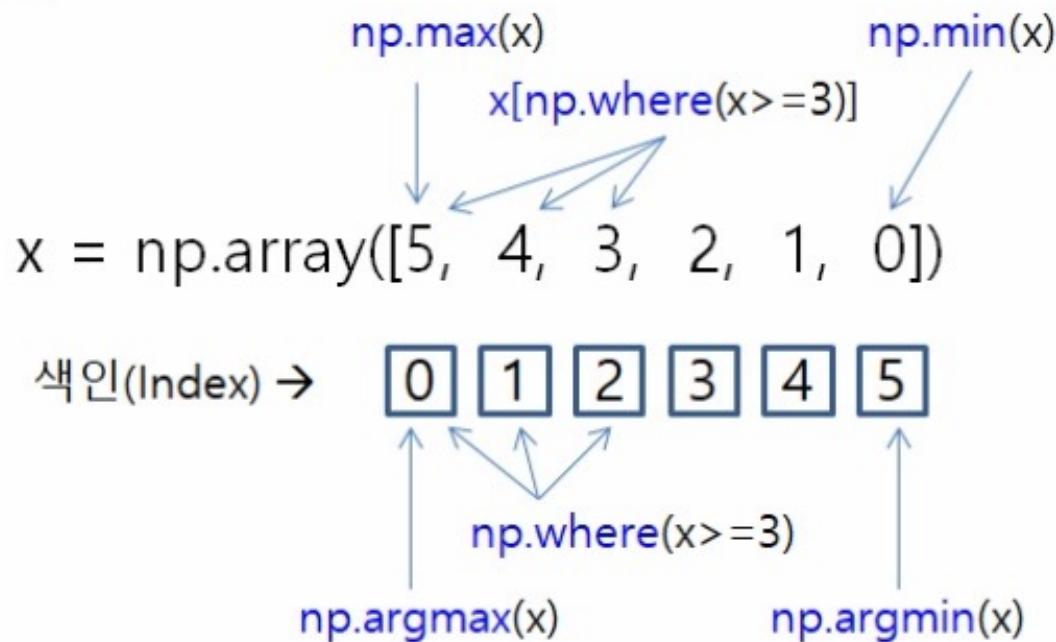
# 배열 원소 간 기울기 계산 범용 함수 (*gradient ufuncs*)  
`np.gradient(g), np.gradient(g, 2)`  
`np.gradient(g, edge_order=2)`  
`np.gradient(h, axis=0), np.gradient(h, axis=1)`

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



## [ Python NumPy ] 최소값, 최대값, 조건에 해당하는 값, 색인



<http://rfriend.tistory.com>

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)
```

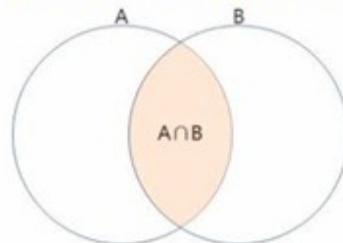
```
[0, 0, 0, 0, 0, 0, 1, 2, 3, 4]
```



## [ Python NumPy ] 집합 함수 (set functions)

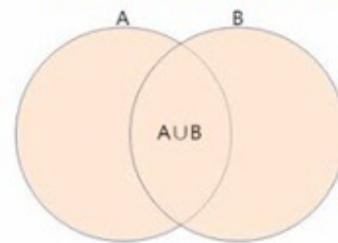
교집합 (intersect)

`np.intersect1d(A, B)`



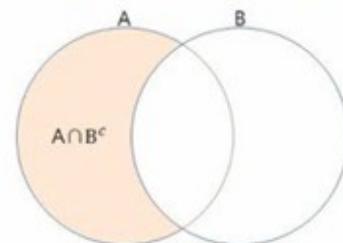
합집합 (union)

`np.union1d(A, B)`



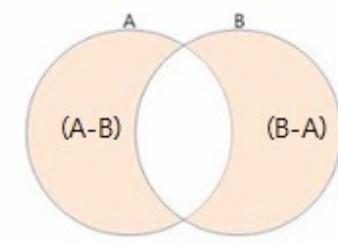
차집합 (relative complement)

`np.setdiff1d(A, B)`



대칭차집합 (symmetric difference)

`np.setxor1d(A, B)`



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

## reshape & ravel

```
a1 = np.arange(1, 13)
```

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

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

```
a1.reshape(3, 4)
a1.reshape(-1, 4)
a1.reshape(3, -1)
    .ravel() # back to 1D
```

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

```
a1.reshape(3, -1, order='F')
    .ravel(order='F') # back to 1D
```

## stack

```
a1 = np.arange(1, 13)
```

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

```
a2 = np.arange(13, 25)
```

13	14	15	16	17	18	19	20	21	22	23	24
----	----	----	----	----	----	----	----	----	----	----	----

```
np.stack((a1, a2))
```

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24

```
np.hstack((a1, a2))
```

1	2	3	4	5	...	20	21	22	23	24
---	---	---	---	---	-----	----	----	----	----	----

## 3D array from 2D arrays

```
a1 = np.arange(1, 13).reshape(3, 4)
a2 = np.arange(13, 25).reshape(3, -1)
```

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

13	14	15	16
17	18	19	20
21	22	23	24

```
# stack along axis 0
a3_0 = np.stack((a1, a2))
a3_0.shape: (2, 3, 4)
```

13	14	15	16
17	18	19	20

```
# retrieve a1
a3_0[0]
a3_0[0, :, :]
```

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

```
# stack along axis 2
a3_2 = np.stack((a1, a2), axis=2)
a3_2.shape: (3, 4, 2)
```

9	21
10	22
11	23
12	24
5	17
6	18

```
# retrieve a1
a3_2[:, :, 0]
```

1	13	7	19
2	14	8	20
3	15		
4	16		

```
# stack along axis 1
a3_1 = np.stack((a1, a2), axis=1)
a3_1.shape: (3, 2, 4)
```

9	10	11	12
21	22	23	24
5	6	7	8
21	22	23	24

```
# retrieve a1
a3_1[:, 0, :]
```

## flatten 3D array

13	14	15	16
17	18	19	20
21	22	23	24
5	6	7	8

```
# flatten/ravel
a3_0.ravel()
```

1	2	3	4	5	...	20	21	22	23	24
---	---	---	---	---	-----	----	----	----	----	----

```
# flatten/ravel
a3_0.ravel(order='F')
```

1	13	5	17	9	...	16	8	20	12	24
---	----	---	----	---	-----	----	---	----	----	----

## reshape 3D array

```
# reshape from (2, 3, 4) to (4, 2, 3)
a3_0.reshape(4, 2, 3)
```



19	20	21
22	23	24
13	14	15

7	8	9
10	11	12
16	17	18

1	2	3
4	5	6



[ 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]])
```



[ 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])`

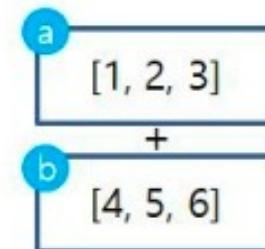
`np.concatenate((c, d),  
axis=0)`



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

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

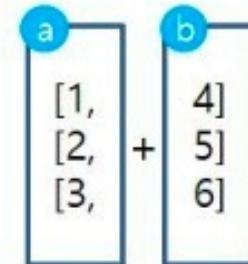
`np.concatenate(  
(c, d), axis=1)`



`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.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()

```
x2=np.array([[2, 1, 6],  
            [0, 7, 4],  
            [5, 3, 2]])
```

위에서  
아래로 정렬  
(from top  
to bottom)

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

<http://rfriend.tistory.com>

좌에서 우로 정렬  
(from left to right)

① **np.sort(x2, axis=1)**

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

② **np.sort(x2, axis=0)**

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

③ **np.sort(x2, axis=0)[::-1]**

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



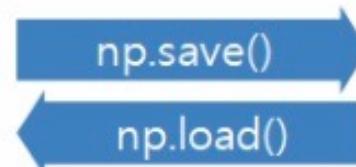
## [ Python NumPy ]

### 배열을 파일로 저장(save array), 불러오기(load)

#### 【 NumPy Array 】

✓ *single array*

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



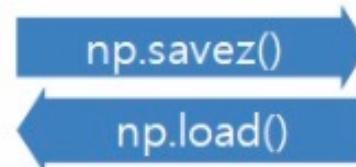
#### 【 File Format 】

**Binary file in  
NumPy format  
(.npy)**

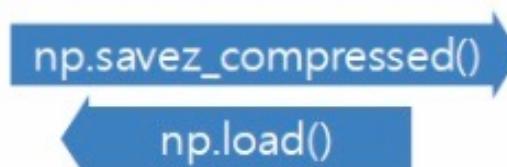
✓ *several arrays*

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

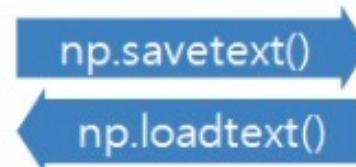
```
y = np.array([5, 6, 7, 8, 9])
```



**a single file in  
uncompressed  
format (.npz)**



**a single file in  
compressed  
format (.npz)**



**Text file**

`close()` 함수로 연 파일을 더 이상 사용할 일  
있으면 메모리 효율 관리를 위해  
<http://rriend.tistory.com>  
`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?

```
1 # 1: Imports.  
2 import numpy as np  
  
4 # 2: Invocation. Requires format string + any # of input args.  
5 arg0 = np.random.normal(...)  
6 arg1 = np.random.normal(...)  
7 ...  
8 argn = np.random.normal(...)  
9  
10 # ...  
11  
12 dst = np.einsum("████████", arg0, arg1, ..., argn)
```

```
14 # 3: Format string. Incomplete example with 3 input args.  
15 dst = np.einsum("█,█,█->█", arg0, arg1, arg2)
```



```
17 # 4: Format string. Incomplete example with 3 input args.  
18 dst = np.einsum("uu,uuu,uu->uuuu", arg0, arg1, arg2)  
19  
20  
21  
22  
23 assert arg0.ndim == len("uu")                                # (Order-2 Tensor)  
24 assert arg1.ndim == len("uuu")                               # (Order-3 Tensor)  
25 assert arg2.ndim == len("uu")                                # (Order-2 Tensor)  
26 assert dst .ndim == len("uuuu")                            # (Order-4 Tensor)
```

```
28 # 5: Format string. Complete examples with 1 and 2 args.
29 s = np.einsum("a->",      v    )
30 T = np.einsum("ij->ji",   M    )
31 C = np.einsum("mn,np->mp", A, B)
32
33
34 assert v.ndim == len("a")
35 assert s.ndim == len("")
36
37 assert M.ndim == len("ij")
38 assert T.ndim == len("ji")
39
40 assert A.ndim == len("mn")
41 assert B.ndim == len("np")
42 assert C.ndim == len("mp")
```

```

44 # 6: Elaborated Example. Matrix multiplication.
45 #      C      =      A      *      B
46 #      Ni x Nj  =  Ni x Nk  *  Nk x Nj
47 #          |           ^-----^
48 #          |           Match
49 #
50 #
51 #          V
52 #      Ni x Nj  <----- Ni x Nj
53
54 C = np.empty((Ni,Nj))
55 for i in range(Ni):
56     for j in range(Nj):
57         total = 0
58
59         for k in range(Nk):
60             total += A[i,k]*B[k,j]
61
62         C[i,j] = total
63
64
65
66
67 C = np.einsum("ik,kj->ij", A, B)

```

} Free Indices:  $i, j$

} Summation Indices:  
all non-free indices ( $k$ )

```
69 # 7. Free Indices
70 C = np.empty((Ni,Nj))
71 for i in range(Ni):
72     for j in range(Nj):
73         total = 0
74
75     for k in range(Nk):
76         total += A[i,k]*B[k,j]
77
78     C[i,j] = total
79
80
81
82
83
84 C = np.einsum("ik,kj->ij", A, B)
```

```
86 # 8. Summation Indices
87 C = np.empty((Ni,Nj))
88 for i in range(Ni):
89     for j in range(Nj):
90         total = 0
91
92         for k in range(Nk):
93             total += A[i,k]*B[k,j]
94
95         C[i,j] = total
96
97
98
99
100
101 C = np.einsum("ik,kj->ij", A, B)
```

```
103 # 9. Elaborated Example. Matrix diagonal extraction.  
104 d = np.empty((Ni))  
105 for i in xrange(Ni):  
106     total = 0  
107  
108     total += M[i,i]  
109     d[i] = total  
110  
111  
112  
113  
114  
115  
116 d = np.einsum("ii->i", M)
```

The diagram illustrates the execution flow of the code. It shows the flow of data from the matrix  $M$  through the summation and assignment operations to the resulting array  $d$ . Red arrows indicate the flow from  $M[i,i]$  to  $total$  and from  $total$  to  $d[i]$ . Blue arrows indicate the flow from  $M[i,i]$  directly to  $d[i]$ . A dotted red arrow points from the start of the loop back to the beginning of the loop. A blue bracket on the right side groups the assignment to  $d[i]$  and the assignment to  $total$  under the comment 'Summation Indices: None'.

```

118 # 10. Elaborated Example. Matrix trace.
119 Tr = 0      # Scalar! Has dimension 0 and no indices
120
121
122 total = 0
123
124 for i in xrange(Ni):
125     total += M[i,i]                                } Summation Indices: i
126
127 Tr = total
128
129
130
131
132 Tr = np.einsum("ii->", M)

```

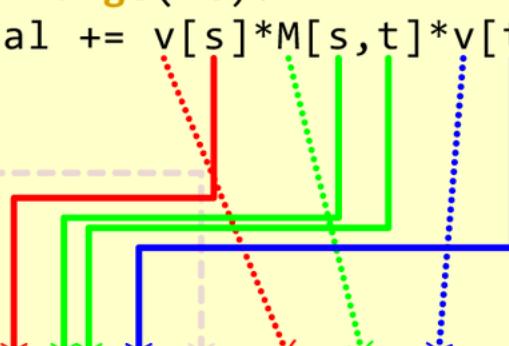
```

134 # 11. Elaborated Example. Quadratic Form.
135 x = 0
136
137
138 total = 0
139
140 for s in xrange(Ns):
141     for t in xrange(Nt):
142         total += v[s]*M[s,t]*v[t]
143
144 x = total
145
146
147
148
149 x = np.einsum("s,st,t->", v, M, v)

```

} Free Indices: None

} Summation Indices: s, t



```

151 # 12. Elaborated example. Batched outer product.
152
153 R = np.empty((NB,ni,nj))
154 for B in xrange(NB):
155     for i in xrange(Ni):
156         for j in xrange(Nj):
157             total = 0
158
159             total
160
161             R[B,i,j] = total
162
163             += P[B,i]*Q[B,j]
164
165
166
167 R = np.einsum("Bi,Bj->Bij", P, Q)

```

The diagram illustrates the mapping of loop indices from the provided Python code to the summation indices used in the einsum function. Red arrows point from the nested loops `B`, `i`, and `j` in the code to the summation indices `Bi` and `Bj` in the einsum notation. A green bracket groups the indices `Bi` and `Bj`. A blue arrow points from the `total` variable in the code to the summation index `Bij` in the einsum notation. A dotted arrow points from the `+` operator in the code to the `">"` operator in the einsum notation, indicating that the summation is over the indices `Bi` and `Bj`.

```

169 # 13. Natural consequences of einsum definition.
170 # Requirements on size of individual axes.
171 C = np.einsum("ik,kj->ij", A, B)
172
173
174
175 assert A.shape == (Ni, Nk)
176 assert B.shape == (Nk, Nj)
177 assert C.shape == (Ni, Nj)
178
179 # Requirement for identical size of certain axes due to
180 # shared index label.
181 # Example 1: Matrix Multiplication
182 C = np.einsum("ik,kj->ij", A, B)
183
184
185
186 assert A.shape[1] == B.shape[0] # == Nk
187 assert A.shape[0] == C.shape[0] # == Ni
188 assert B.shape[1] == C.shape[1] # == Nj
189
190 # Example 2: Matrix Diagonal Extraction
191 d = np.einsum("ii->i", D)
192
193
194
195 assert D.shape[0] == D.shape[1] # == Ni
196 assert D.shape[1] == d.shape[0] # == Ni

```

```
198 # 14: Format Strings. Rules.
199
200 #     Bad. Number of input index groups doesn't match number of
201 #             arguments.
202 np.einsum("ab,bc->ac", A)
203
204 #     Bad. Indexes must be ASCII upper/lowercase letters.
205 np.einsum("012,1^%->:;?", A, B)
206
207 #     Bad. Argument 0 has 3 dimensions but only 2 indices are
208 #             given.
209 A = np.random.normal(size = (2,3,4))
210 B = np.random.normal(size = (4,5,6))
211 np.einsum("ab,bcd->a", A, B)
212
213 #     Bad. One of the output indices isn't in the set of all
214 #             input indices.
215 np.einsum("ab,bc->acz", A, B)
216
217 #     Bad. Output has a repeated index.
218 np.einsum("ab,bc->baa", A, B)
219
220 #     Bad. Mismatches in the sizes of input argument axes
221 #             that are labelled with the same index.
222 A = np.random.normal(size = (2,3,4))
223 B = np.random.normal(size = (3,4,5))
224 np.einsum("ckj,cqq->c", A, B)
225
226 assert      A.shape[0] == B.shape[0]          # ERROR: 2 != 3
227 assert      B.shape[1] == B.shape[2]          # ERROR: 4 != 5
```

```
229 # 15: MLP Backprop done easily (stochastic version).
230 #     h = sigmoid(Wx + b)
231 #     y = softmax(Vh + c)
232 Ni = 784
233 Nh = 500
234 No = 10
235
236 W = np.random.normal(size = (Nh,Ni)) # Nh x Ni
237 b = np.random.normal(size = (Nh,)) # Nh
238 V = np.random.normal(size = (No,Nh)) # No x Nh
239 c = np.random.normal(size = (No,)) # No
240
241 # Load x and t...
242 x, t = train_set[k]
243
244 # With a judicious, consistent choice of index labels, we can
245 # express fprop() and bprop() extremely tersely; No thought
246 # needs to be given about the details of shoehorning matrices
247 # into np.dot(), such as the exact argument order and the
248 # required transpositions.
249 #
250 # Let
251 #
252 #     'i' be the input dimension label.
253 #     'h' be the hidden dimension label.
254 #     'o' be the output dimension label.
255 #
256 # Then
257
258 # Fprop
259 ha = np.einsum("hi, i -> h", W, x) + b
260 h = sigmoid(ha)
261 ya = np.einsum("oh, h -> o", V, h) + c
262 y = softmax(ya)
263
264 # Bprop
265 dLdy = y - t
266 dLdV = np.einsum("h , o -> oh", h, dLdy)
267 dLdc = dLdy
268 dLdh = np.einsum("oh, o -> h ", V, dLdy)
269 dLdha = dLdh * sigmoidgrad(ha)
270 dLdW = np.einsum("i, h -> hi", x, dLdha)
271 dLdb = dLdha
```

```
273 # 16: MLP Backprop done easily (batch version).
274 #      But we want to exploit hardware with a batch version!
275 #      This is trivially implemented with simple additions
276 #      to np.einsum's format string, in addition to the usual
277 #      averaging logic required when handling batches. We
278 #      implement even that logic with einsum for demonstration
279 #      and elegance purposes.
280 batch_size = 128
281
282 # Let
283 #      'B' be the batch dimension label.
284 #      'i' be the input dimension label.
285 #      'h' be the hidden dimension label.
286 #      'o' be the output dimension label.
287 #
288 # Then
289
290 # Fprop
291 ha    = np.einsum("hi, Bi -> Bh", W, x) + b
292 h     = sigmoid(ha)
293 ya    = np.einsum("oh, Bh -> Bo", V, h) + c
294 y     = softmax(ya)
295
296 # Bprop
297 dLdy = y - t
298 dLdV = np.einsum("Bh, Bo -> oh", h, dLdy) / batch_size
299 dLdc = np.einsum("Bo      -> o ", dLdy) / batch_size
300 dLdh = np.einsum("oh, Bo -> Bh", V, dLdy)
301 dLdha = dLdh * sigmoidgrad(ha)
302 dLdW = np.einsum("Bi, Bh -> hi", x, dLdha) / batch_size
303 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.