

■ NumPy

- Numerical Python

- 과학 계산 분야를 위한 라이브러리로써,

고성능의 다차원 배열 객체와 연산에 필요한 여러 유용한 기능을 제공

- 파이썬의 기본 List에 비해 빠르고, 메모리를 효율적으로 사용

- 별도의 Loop 를 사용하지 않아도 자체 함수를 통해 배열 내의 데이터 연산

- 참조 사이트

- Python Numpy Tutorial

- <http://cs231n.github.io/python-numpy-tutorial/#numpy>

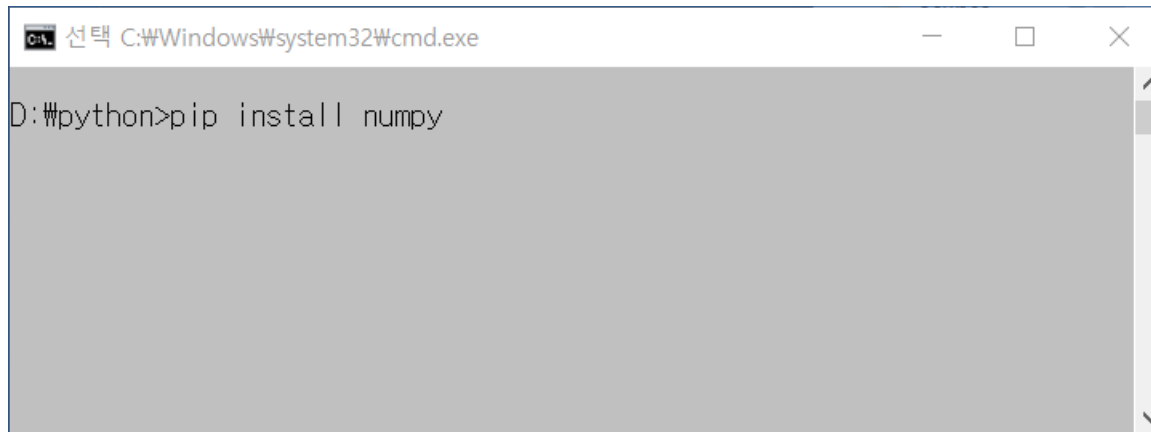
- 데이터 사이언스 스쿨

- <https://goo.gl/3hsjbS>

■ NumPy

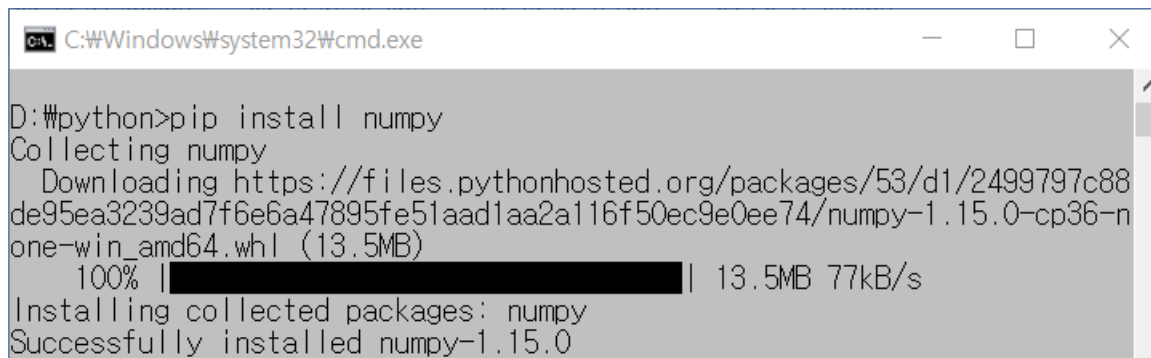
● 설치

- pandas 또는 jupyter notebook 등을 설치한 상태에서는 추가 설치 불필요
- `pip install numpy`



```
C:\Windows\system32\cmd.exe

D:\python>pip install numpy
```

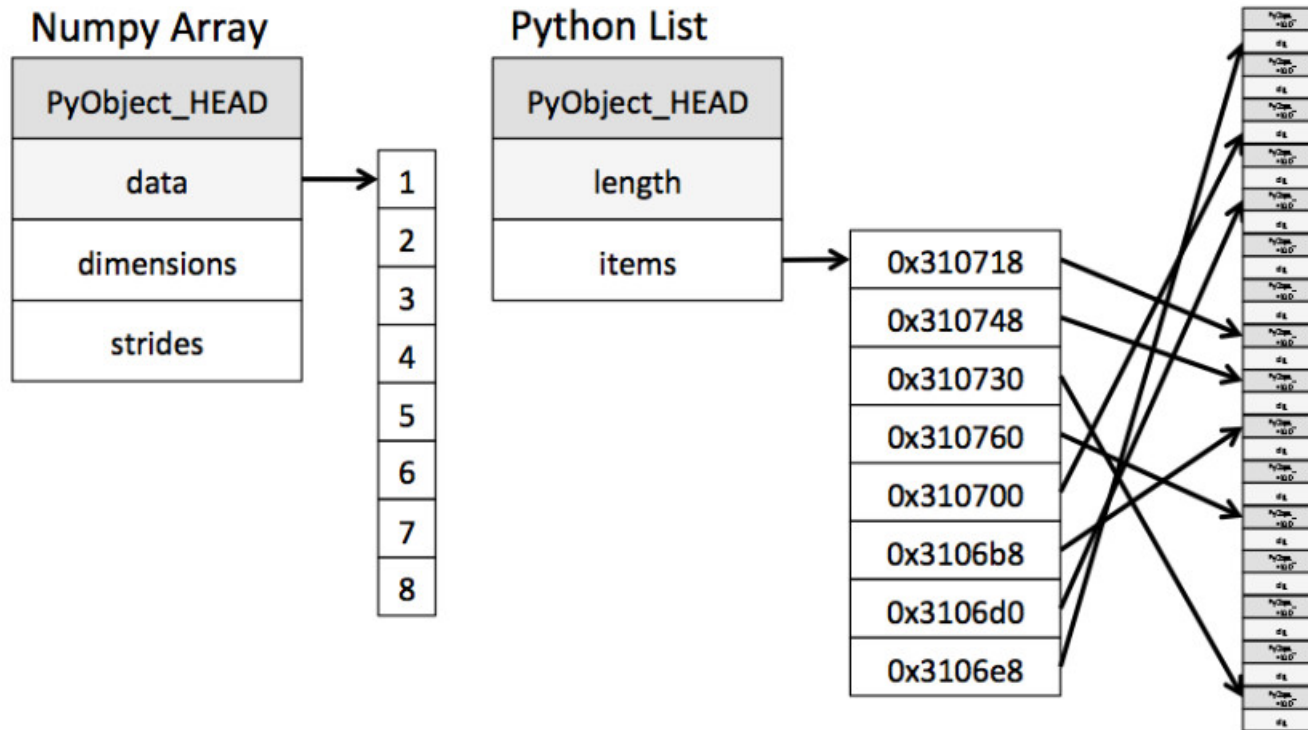


```
C:\Windows\system32\cmd.exe

D:\python>pip install numpy
Collecting numpy
  Downloading https://files.pythonhosted.org/packages/53/d1/2499797c88de95ea3239ad7f6e6a47895fe51aad1aa2a116f50ec9e0ee74/numpy-1.15.0-cp36-none-win_amd64.whl (13.5MB)
    100% |████████████████████████████████████████| 13.5MB 77kB/s
Installing collected packages: numpy
Successfully installed numpy-1.15.0
```

■ 배열 (NDArray : Numpy Dimensional Array)

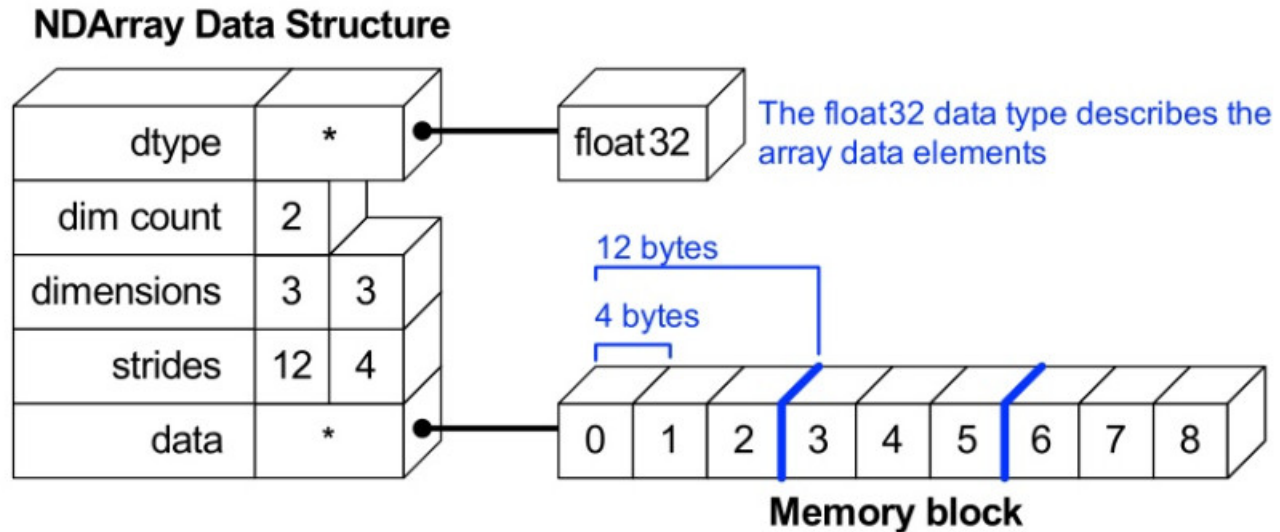
- 동일한 타입의 값을 가진다
- 내부적으로 C언어의 Array를 사용 (속도 빠름)



출처 : <http://jakevdp.github.io/blog/2014/05/09/why-python-is-slow>

■ 배열 (NDArray : Numpy Dimensional Array)

Array Data Structure



출처 : <https://www.slideshare.net/enthought/numpy-talk-at-siam>

■ 배열 (NDArray : Numpy Dimensional Array)

NumPy dtypes

Basic Type	Available NumPy types	Comments
Boolean	<code>bool</code>	Elements are 1 byte in size.
Integer	<code>int8, int16, int32, int64, int128, int</code>	<code>int</code> defaults to the size of <code>int</code> in C for the platform.
Unsigned Integer	<code>uint8, uint16, uint32, uint64, uint128, uint</code>	<code>uint</code> defaults to the size of unsigned <code>int</code> in C for the platform.
Float	<code>float32, float64, float, longfloat,</code>	<code>float</code> is always a double precision floating point value (64 bits). <code>longfloat</code> represents large precision floats. Its size is platform dependent.
Complex	<code>complex64, complex128, complex, longcomplex</code>	The real and complex elements of a <code>complex64</code> are each represented by a single precision (32 bit) value for a total size of 64 bits.
Strings	<code>str, unicode</code>	
Object	<code>Object</code>	Represent items in array as Python objects.
Records	<code>Void</code>	Used for arbitrary data structures.

출처 : <https://www.slideshare.net/enthought/numpy-talk-at-siam>

■ 배열 (NDArray : Numpy Dimensional Array)

dtype 접두사	설명	사용 예
b	불리언	b (True 또는 False)
i	정수	i8 (64비트)
u	부호없는 정수	u8 (64비트)
f	부동 소수점	f8 (64비트)
c	복소 부동소수점	c16 (128비트)
O	객체	O (객체에 대한 포인터)
S	바이트 문자열	S24 (24 글자)
U	유니코드 문자열	U24 (24 유니코드 글자)

■ 배열 (NDArray : Numpy Dimensional Array)

● 모듈 호출

```
import numpy as np
```

● 배열 생성

```
np_arr1 = np.array([1, 2, 3]) # 기본 사용  
print(np_arr1)
```

```
[1 2 3]
```

```
np_arr2 = np.array([1, 2.0, '3']) # string 타입으로 자동 변환  
print(np_arr2)
```

```
['1' '2.0' '3']
```

```
np_arr3 = np.array([1, 2.0, '3'], float) # float 타입으로 강제 지정  
print(np_arr3)
```

```
[1. 2. 3.]
```

■ 배열 (NDArray : Numpy Dimensional Array)

● 자료형 확인 : dtype

```
print(np_arr1.dtype)  
print(np_arr2.dtype)  
print(np_arr3.dtype)
```

```
int32  
<U32  
float64
```

● Dimension 확인 : shape

```
print(np_arr1.shape)  
print(np_arr2.shape)  
print(np_arr3.shape)
```

```
(3,)  
(3,)  
(3,)
```


■ 배열 (NDArray : Numpy Dimensional Array)

● shape : vector / matrix / tensor

– vector : 1차원

```
np_arr1 = np.array([1, 2, 3]) # 기본 사용  
print(np_arr1)
```

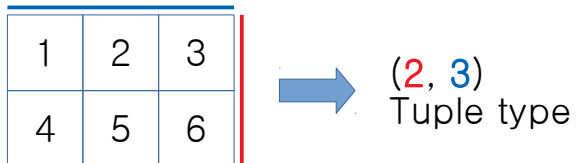
[1 2 3]



– matrix : 2차원

```
matrix = np.array([ [1, 2, 3], [4, 5, 6] ])  
print(matrix)
```

[[1 2 3]
 [4 5 6]]



■ 배열 (NDArray : Numpy Dimensional Array)

● shape : vector / matrix / tensor

– tensor : 3차원

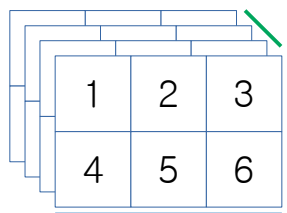
```
tensor = np.array([
    [ [1, 2, 3], [4, 5, 6] ],
    [ [1, 2, 3], [4, 5, 6] ],
    [ [1, 2, 3], [4, 5, 6] ],
    [ [1, 2, 3], [4, 5, 6] ]
])
print(tensor)
```

```
[[[1 2 3]
    [4 5 6]]
```

```
[[1 2 3]
 [4 5 6]]
```

```
[[1 2 3]
 [4 5 6]]
```

```
[[1 2 3]
 [4 5 6]]]
```



(4, 2, 3)
Tuple type

■ 배열 (NDArray : Numpy Dimensional Array)

● 배열 생성 시 주의사항

```
np_arr4 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]]) # shape이 같은 경우  
print(np_arr4)
```

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

```
np_arr5 = np.array([[1, 2, 3], [5, 6, 7, 8]]) # shape이 다른 경우  
print(np_arr5)
```

```
[list([1, 2, 3]) list([5, 6, 7, 8])] → (2, )
```

■ 배열 (NDArray : Numpy Dimensional Array)

● 차원(rank) 확인 : ndim (Number of Dimension)

```
print(np_arr1.ndim)
print(matrix.ndim)
print(tensor.ndim)
```

1
2
3

● 요소의 개수 확인 : size

```
print(np_arr1.size)
print(matrix.size)
print(tensor.size)
```

3
6
24

1	2	3
---	---	---



(3,)
Tuple type

1	2	3
4	5	6



(2, 3)
Tuple type

1	2	3
4	5	6



(4, 2, 3)
Tuple type

■ 배열 (NDArray : Numpy Dimensional Array)

● 자료형 지정 및 확인 : dtype (Data Type)

```
np.array([ [1, 2, 3], [4, 5, 6] ], dtype=int).dtype  
dtype('int32')
```

```
np.array([ [1, 2, 3], [4, 5, 6] ], dtype=np.int8).dtype  
dtype('int8')
```

```
np.array([ [1, 2, 3], [4, 5, 6] ], dtype=np.float32).dtype  
dtype('float32')
```

● 배열의 Memory 크기 확인 : nbytes

```
np.array([ [1, 2, 3], [4, 5, 6] ], dtype=int).nbytes
```

24 6개 * 4byte = 24byte

```
np.array([ [1, 2, 3], [4, 5, 6] ], dtype=np.int8).nbytes
```

6 6개 * 1byte = 6byte

```
np.array([ [1, 2, 3], [4, 5, 6] ], dtype=np.float32).nbytes
```

24 6개 * 4byte = 24byte

■ 배열 인덱싱

● 기본 List의 인덱싱과 거의 같음

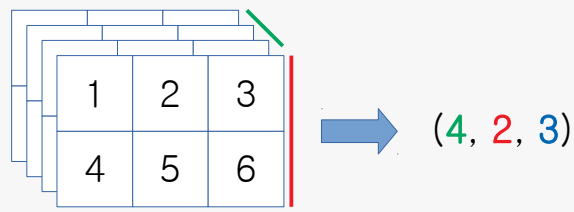
- `array[col]`
- `array[row][col]`
- `array[rank][row][col]`

● 다차원 배열인 경우에는 대괄호([]) 대신 콤마(,) 사용 가능

- `array[row, col]`
- `array[rank, row, col]`

■ 배열 인덱싱

```
tensor2 = np.array([
    [ [1, 2, 3], [4, 5, 6] ],
    [ [11, 12, 13], [14, 15, 16] ],
    [ [21, 22, 23], [24, 25, 26] ],
    [ [31, 32, 33], [34, 35, 36] ]
])
print(tensor2)
```



```
[[[ 1  2  3]
   [ 4  5  6]]
```



tensor2[0, 1, 2]

```
[[11 12 13]
 [14 15 16]]
```



tensor2[1, 0, 2]

```
[[21 22 23]
 [24 25 26]]
```



tensor2[2, 1, 1]

```
[[31 32 33]
 [34 35 36]]
```



tensor2[3, 0, 0]

■ 배열 인덱싱 (fancy indexing)

- 기본 데이터

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

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

- Boolean 배열 방식

```
idx = [True, False, True, True, False]  
arr[idx]
```

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

- index (정수) 배열 방식

```
idx = [0, 2, 3]  
arr[idx]
```

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


■ 배열 인덱싱 (fancy indexing)

- 연산식 사용 - 1

```
arr % 2 == 0
```

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

```
arr[arr % 2 == 0]
```

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

- 연산식 사용 - 2

```
(arr % 2 == 0) | (arr == 3)
```

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

```
arr[(arr % 2 == 0) | (arr == 3)]
```

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

■ 배열 슬라이싱

- 행과 열 부분을 나눠서 슬라이싱 가능
- 다차원 배열의 부분(범위) 데이터를 추출할 때 유용
 - 파이썬의 기본 슬라이싱 구문과 콤마(,)를 함께 사용

```
matrix2 = np.array([ [1, 2, 3, 4, 5],  
                     [6, 7, 8, 9, 10],  
                     [11, 12, 13, 14, 15],  
                     [16, 17, 18, 19, 20]])
```

matrix2

```
array([[ 1,  2,  3,  4,  5],  
       [ 6,  7,  8,  9, 10],  
       [11, 12, 13, 14, 15],  
       [16, 17, 18, 19, 20]])
```

■ matrix2[0, 1:-1]

■ matrix2[:, -2:]

■ matrix2[-2:, -3:]

■ matrix2[0:, 0]

■ matrix2[1:-1, 1:3]

■ 배열 슬라이싱

```
tensor2 = np.array([
    [ [1, 2, 3], [4, 5, 6] ],
    [ [11, 12, 13], [14, 15, 16] ],
    [ [21, 22, 23], [24, 25, 26] ],
    [ [31, 32, 33], [34, 35, 36] ]
])
print(tensor2)
```

[[[1 2 3]
 [4 5 6]]

[[11 12 13]
 [14 15 16]]

[[21 22 23]
 [24 25 26]]

[[31 32 33]
 [34 35 36]]]

31	32	33
34	35	36
21	22	23
24	25	26
11	12	13
14	15	16
1	2	3
4	5	6

■ tensor2[:, 0, -1]

■ tensor2[1:, -1:, :-1]