DB HiTek

인공지능 활용 능력 개발 중급

[1차시]

AIBIZ
AIBIZ.Co.Ltd

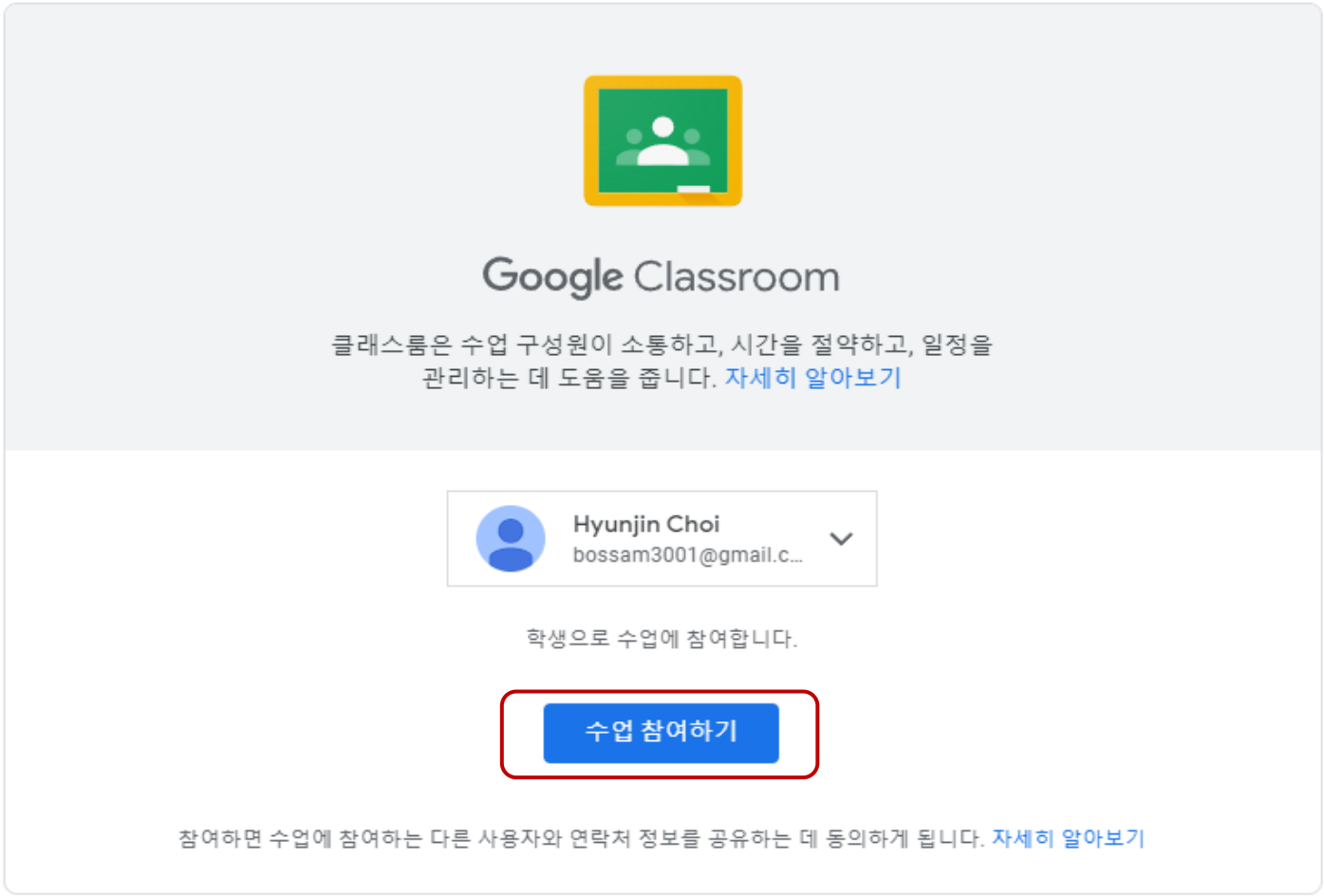
목 차

학습목표

- 인공지능 개발을 위한 기초 모듈인 Numpy 익히기
- 데이터 구조에 대한 이해
- 선형대수학을 통한 데이터를 보는 관점 바꾸기

구글 클래스 룸

https://classroom.google.com/c/NjE1NDg3NjA2ODI4?cjc=u2lk3cg



Numpy 소개

- Numerical Python
- 파이썬의 내장 타입인 리스트보다 데이터의 저장 및 처리에 있어 효율적인 NumPy 배열을 제공
- 선형 대수와 관련된 기능을 제공
- 파이썬을 기반으로 한 데이터 과학 도구의 핵심 패키지
- ※ 데이터 사이언스 영역의 대부분의 도구(Pandas, Scipy, scikit-learn 패키지 등)가 Numpy 기반

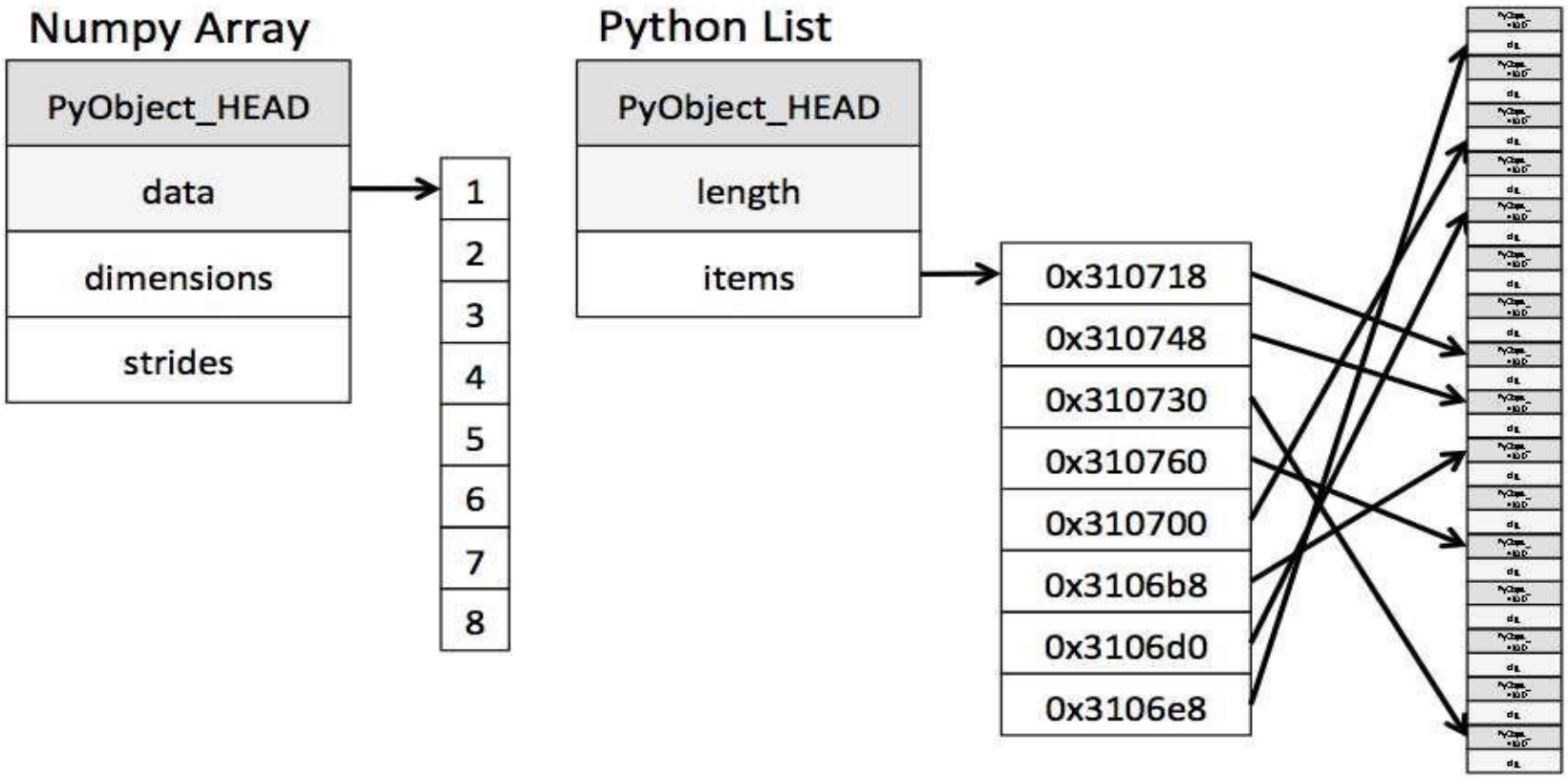
Why numpy ?

Why Numpy ?

- 굉장히 큰 matrix처리 및 다양한 연산 기능 제공
- 반복문 없이 데이터 배열에 대한 처리 지원
- 선형대수와 관련된 다양한 기능 제공
- python은 인터프리터 언어이므로 리스트로 처리 시 속도가 느린 단점이 있다, 넘파이는 이러한 단점을 보완해준다.
- 속도가 빠른 이유
  - numpy는 C언어로 구현되어 있음
  - numpy는 한 task를 subtask로 알아서 나눠 병렬적으로 처리한다
  - 메모리 접근 방식에 대한 차이로 인하여 속도가 빠르다

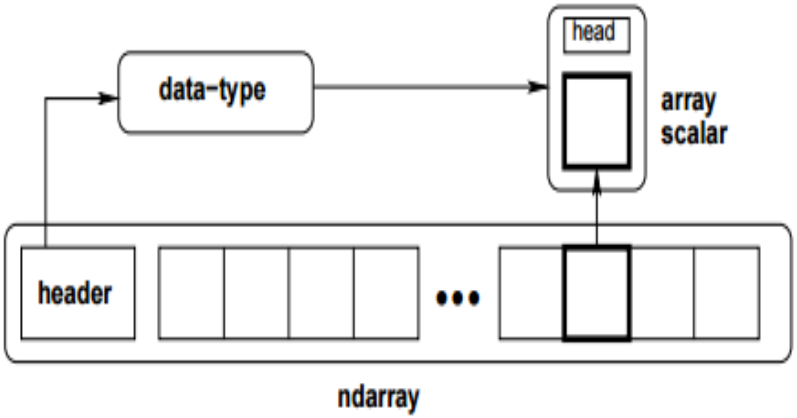
ndarray vs. list 구조

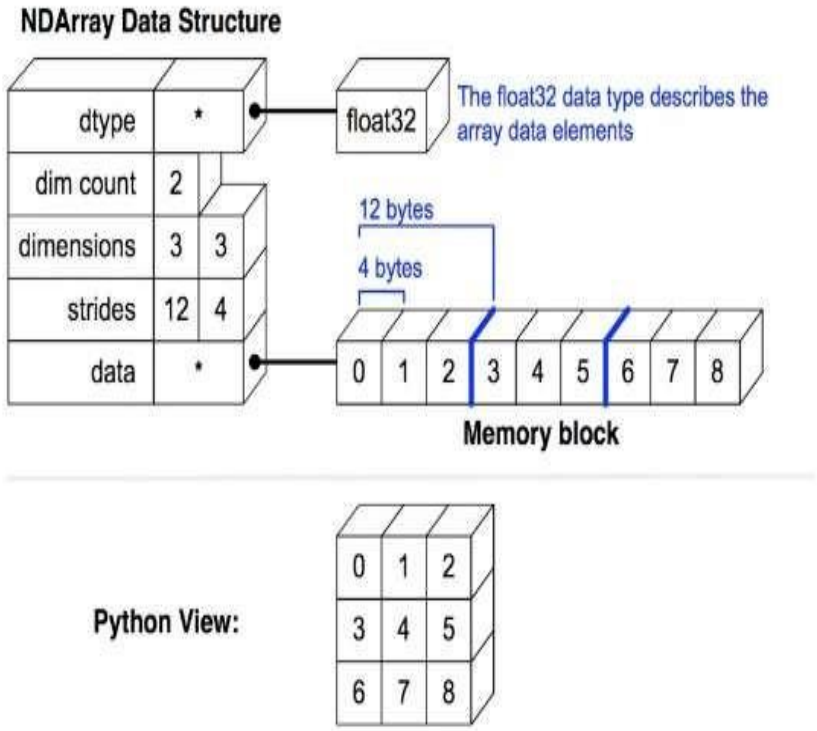
Ndarray와 list는 내부 구조부터 다르게 되어있어 ndarray가 더 처리가 빠르게 실행됨



ndarray 구조

Ndarray는 데이터를 관리하고 data-type은 실제 데이터들의 값을 관리하며, array scalar는 위치를 관리





ndarray 속성

Ndarray 생성시 shape, dtype,strides 인스턴스 속성이 생성됨

```
import numpy as np

help(np.ndarray)
```

```
Help on class ndarray in module numpy:

class ndarray(builtins.object)
 |  ndarray(shape, dtype=float, buffer=None, offset=0,
 |          strides=None, order=None)
 |
 |  An array object represents a multidimensional, homogeneous a
 |  of fixed-size items.  An associated data-type object describ
 |  format of each element in the array (its byte-order, how mar
 |  occupies in memory, whether it is an integer, a floating poi
 |  or something else, etc.)
 |
 |  Arrays should be constructed using `array`, `zeros` or `empt
 |  to the See Also section below).  The parameters given here r
 |  a low-level method (`ndarray(...)`) for instantiating an arr
 |
 |  For more information, refer to the `numpy` module and examir
```

```
import numpy as np

a = np.array([1, 2, 3])
print(a.shape)
print(a.dtype)
print(a.strides)
```

```
(3,)
int64
(8,)
```

데이터 타입 부여

ndarray는 각 원소별로 동일한 데이터 타입으로 처리

array([원소 , 원소 , 원소], dtype)

```
import numpy as np

l = [1, 2, 3, 4]
a = np.array(l, int)
print(a)

a = np.array(l, float)
print(a)

a = np.array(l, str)
print(a)
```

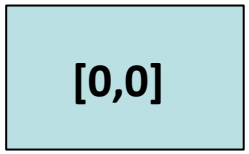
```
[1 2 3 4]
[1. 2. 3. 4.]
['1' '2' '3' '4']
```

0차원

numpy.array 생성시 단일값(scalar value)를 넣으면 arrary 타입이 아니 일반 타입을 만듬

Column: 열

Row : 행



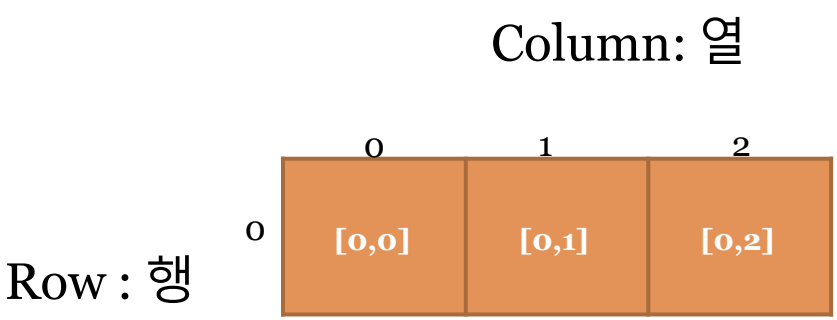

```
import numpy as np

a = np.array(10)
print(a)
print(a.ndim)
```

```
10
0
```

1차원

배열의 특징. 차원, 형태, 요소를 가지고 있음

생성시 데이터와 타입을 넣으면 ndim(차원)으로 확인




```
import numpy as np

l = [1, 2, 3, 4]
a = np.array(l)
print(a)
print(a.ndim)
```

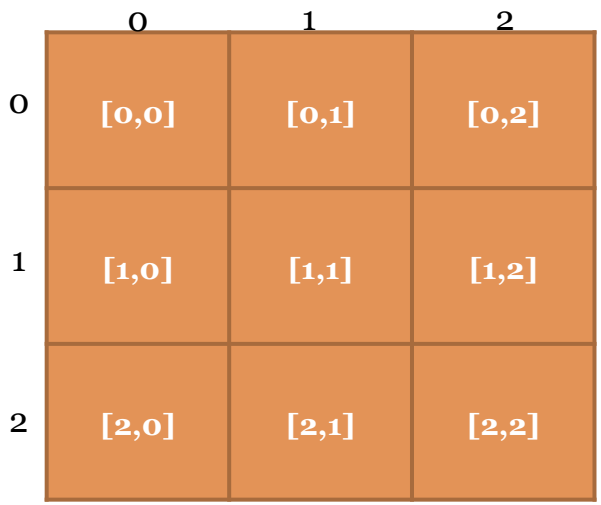
```
[1 2 3 4]
1
```

2차원 배열

3행, 3열의 배열을 기준으로 어떻게 내부를 행과 열로 처리하는 지를 이해

Column: 열

Row : 행



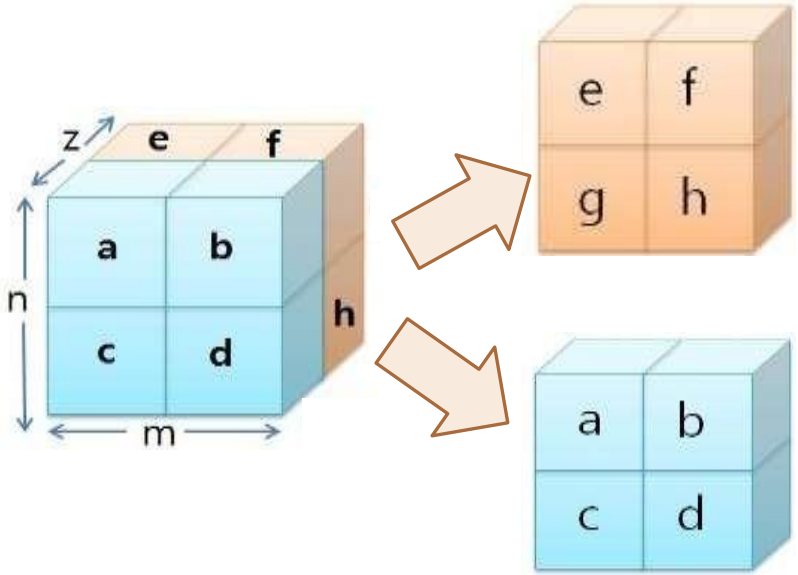

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
print(a)
print(a.ndim)
```

```
[[1 2]
 [3 4]]
2
```

Index 접근 표기법
배열명[행][열]
배열명[행, 열]
Slice 접근 표기법 배열명[슬라이스, 슬라이스]

3차원 배열

numpy.array 생성시 sequence 각 요소에 대해 접근변수와 타입을 정할 수 있음




```
import numpy as np

a = np.array([[[1, 2], [3, 4]], [[1, 2], [3, 4]]])
print(a)
print(a.ndim)
```

```
[[[1 2]
  [3 4]]

 [[1 2]
  [3 4]]]
3
```

할당은 참조만 전달

Ndarray 타입을 검색이나 슬라이싱은 참조만 할당하므로 변경을 반지하기 위해서는 새로운 ndarray로 만들어 사용. copy 메소드가 필요

```
import numpy as np

l = [1, 2, 3, 4]
a = np.array(l)
s = a[:2]
ss = a[:2].copy()
print(s.size)
s[0] = 99
print(a)
print(s)
print(ss)
```

```
2
[99  2  3  4]
[99  2]
[1 2]
```

벡터화 연산 : for문 미사용

F (화씨) = c(섭씨) * 9 / 5 + 32 이 공식을 기준으로 연속적인 배열을 loop 문 없이 계산

```
import numpy as np

cvalues = [25.3, 24.8, 26.9, 23.9]

# 섭씨 ndarray 생성
C = np.array(cvalues)
F = C * 9 / 5 + 32
print(type(F), F)

# 기존 방식, 리스트 컴프리헨션도 Loop 실행
F1 = [x * 9 / 5 + 32 for x in cvalues]
print(type(F1), F1)
```

```
<class 'numpy.ndarray'> [77.54 76.64 80.42 75.02]
<class 'list'> [77.54, 76.64, 80.42, 75.02]
```

ndarray 특징은 array 원소만큼 자동으로 순환 계산해서 ndarray로 반환

list와 ndarray 계산 성능

numpy.ndarray로 계산시 python list 타입에 비해 계산 속도가 빠름

```
import numpy as np
import time
SIZE_OF_VEC = 10000000

def pure_python_version():
  t1 = time.time()
  X = range(SIZE_OF_VEC)
  Y = range(SIZE_OF_VEC)
  Z = []
  for i in range(len(X)):
    Z.append(X[i] + Y[i])
  return time.time() - t1

def numpy_version():
  t2 = time.time()
  X = np.arange(SIZE_OF_VEC)
  Y = np.arange(SIZE_OF_VEC)
  Z = X + Y
  return time.time() - t2

t1 = pure_python_version()
t2 = numpy_version()
print(t1, t2)
print("numpy is in this example " + str(t1/t2) + " faster!")
```

```
2.4715118408203125 0.10322093963623047
numpy is in this example 23.943899847553933 faster!
```

배열을 c언어처럼 관리하므로 별도의 index를 구성하지 않으므로 계산 속도 빠름

2.4715118408203125 0.10322093963623047
numpy is in this example 23.943899847553933 faster!

Numpy array 생성

생성함수 : 1

Ndarray를 생성하는 함수

함수	설명
array	입력 데이터를 ndarray로 변환하며 dtype이 명시되지 않은 경우에는 자료형을 추론해 저장
asarray	입력 데이터를 ndarray로 변환하지만 입력 데이터가 ndarray일 경우 그대로 표시
arange	내장 range 함수와 유사하지만 리스트 대신 ndarray를 반환
ones	주어진 dtype과 주어진 모양을 가지는 배열을 생성하고 내용을 모두 1로 초기화
ones_like	주어진 배열과 동일한 모양과 dtype을 가지는 배열을 새로 생성하여 1로 초기화
zero	ones와 같지만 0으로 채운다

생성함수 : 2

Ndarray를 생성하는 함수

함수	설명
zeros_like	ones_like와 같지만 0dmfh codnsek
empty	메모리를 할당하지만 초기화가 없음
empty_like	메모리를 할당하지만 초기화가 없음
ndarray	메모리를 할당하지만 초기화가 없음
eye	n*n 단위행렬 생성하고 대각선으로 1을 표시하고 나머지는 0
identity	n*n 단위행렬 생성
linspace	시작과 종료 그리고 총갯수 생성을 주면 ndarray로 생성

배열 만들기

배열의 특징. 차원, 형태, 요소를 가지고 있음 생성시 데이터와 타입을 넣으면 ndim(차원), shape(형태), 타입(dtype)

```
import numpy as np

help(np.array)
```

```
Help on built-in function array in module numpy:

array(...)
    array(object, dtype=None, *, copy=True, order='K', subok=Fal
          like=None)

    Create an array.

    Parameters
    ----------
    object : array_like
        An array, any object exposing the array interface, an ob
        __array__ method returns an array, or any (nested) seque
        If object is a scalar, a 0-dimensional array containing
        returned.
    dtype : data-type, optional
        The desired data-type for the array.  If not given, ther
        be determined as the minimum type required to hold the c
        sequence.
    copy : bool, optional
        If true (default), then the object is copied.  Otherwise
        only be made if __array__ returns a copy, if obj is a ne
        or if a copy is needed to satisfy any of the other requi
        (`dtype`, `order`, etc.).
```

```
import numpy as np

l2d = [[1, 2, 3], [4, 5, 6]]
np2d = np.array(l2d, int)
print(np2d)
print(np2d.ndim)
print(np2d.shape)
print(np2d.dtype)
```

```
[[1 2 3]
 [4 5 6]]
2
(2, 3)
int64
```

array, asarray 함수

np.array : ndarry 생성 함수로 가장 많이 사용됨
원소의 type을 추정하여 통일화시킴
np.asarray :List 등을 ndarray 타입으로 전환하는 함수

```
import numpy as np

x = np.array([(1, 2., '1'), (2, 3., "2")], dtype = int)
print(x)
x = np.array([(1, 2., '1'), (2, 3., "2")])
print(x)
```

```
[[1 2 1]
 [2 3 2]]
[['1' '2.0' '1']
 ['2' '3.0' '2']]
```

```
import numpy as np

l = [1, 2]
a = np.array(l)
print(a)
print(type(a))
b = np.asarray(l)
print(b)
print(type(b))
```

```
[1 2]
<class 'numpy.ndarray'>
[1 2]
<class 'numpy.ndarray'>
```

arange 함수

- 내장 range 함수와 유사하지만 리스트 대신 ndarray를 반환
- 값을 순차적으로 올리며 배열 생성

```
import numpy as np

z = np.arange(3, dtype = int)
print(z)

z1 = np.arange(3, 5, dtype = int)
print(z1)
print(type(z1))
```

```
[0 1 2]
[3 4]
<class 'numpy.ndarray'>
```

zeros, ones, full, empty 함수

ndarray 생성하면 내부 원소들은 zero, one, full 특정값으로 초기화되어 생성, emty는 잔여 메모리 값이 나타남

```
import numpy as np

a = np.zeros((3, 3))
b = np.ones((3, 3))
c = np.full((3, 3), 5)
d = np.empty((3, 3))

print(type(a), a.dtype)
print(type(b), b.dtype)
print(type(c), c.dtype)
print(type(d), d.dtype)
print(a)
print(b)
print(c)
print(d)
```

```
<class 'numpy.ndarray'> float64
<class 'numpy.ndarray'> float64
<class 'numpy.ndarray'> int64
<class 'numpy.ndarray'> float64
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[[5 5 5]
 [5 5 5]
 [5 5 5]]
[[2.46603658e-316 0.00000000e+000 1.35712977e+166]
 [2.31633955e-152 3.94355938e+180 1.89130905e+219]
 [3.68321739e+180 4.95913936e+173 4.74303020e-322]]
```

zeros, ones, empty_likes함수

생성된 ndarray와 shape이 같은 array를 생성하면서 내부 원소들은 zero, one, emty로 초기화하여 시킴

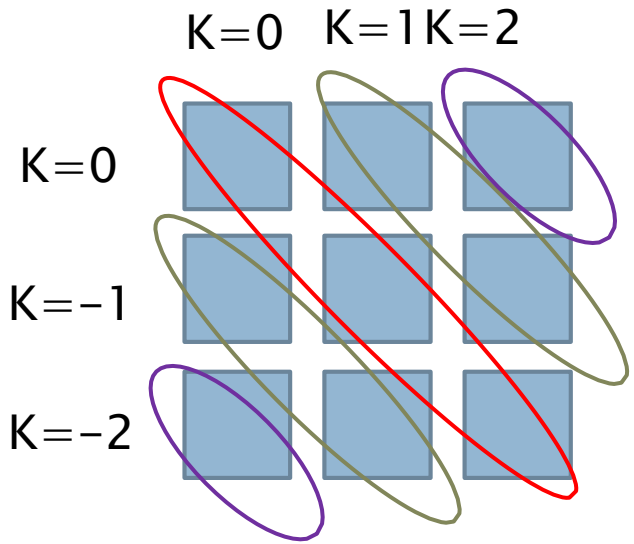
```
import numpy as np

a = np.array([[[1, 2, 3], [3, 4, 5], [6, 4, 7]],
              [[1, 2, 3], [3, 4, 5], [6, 4, 7]],
              [[1, 2, 3], [3, 4, 5], [6, 4, 7]]])
r, c, d = a.shape
a_1 = np.zeros((r, c, d))
a_2 = np.zeros_like(a, float)

print(a_1)
print(a_1.dtype)
print(a_2)
print(a_2.dtype)
```

```
[[[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]]
float64
[[[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]]
float64
```

eye 함수

Narray 생성시 K 값에 따라 1(1.0)이 위치가 대각선 방향으로 생김. N,M을 인자로 넘기면 n행M열 array 만들어짐

numpy.eye(N, M=None, k=0, dtype=<type 'float'>)




```
import numpy as np

np.eye(2, dtype = int)

print(np.eye(3))
print(np.eye(3, k = 1))
print(np.eye(3, k = -1))
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
[[0. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]]
```

numpy.identity 생성함수

numpy.identity 함수로 생성하면 실제 정방형 ndarray 타입이 생기고 대각선으로는 1이 정의됨

```
import numpy as np

print(np.identity(5))
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

Linspace 함수

시작과 종료 그리고 num(요소의 개수)를 지점해서 생성

linspace(start, stop, num=50, endpoint=True, retstep=False)

```
import numpy as np

a = np.linspace(2.0, 3.0, num = 5)
print(a)
```

```
[2.   2.25 2.5  2.75 3.  ]
```

```
a = np.linspace(2.0, 3.0, num = 5, endpoint = False)
print(a)
```

```
[2.  2.2 2.4 2.6 2.8]
```

```
a = np.linspace(2.0, 3.0, num = 5, retstep = True)
print(a)
print(a[0], type(a[0]))
print(a[1], type(a[1]))
```

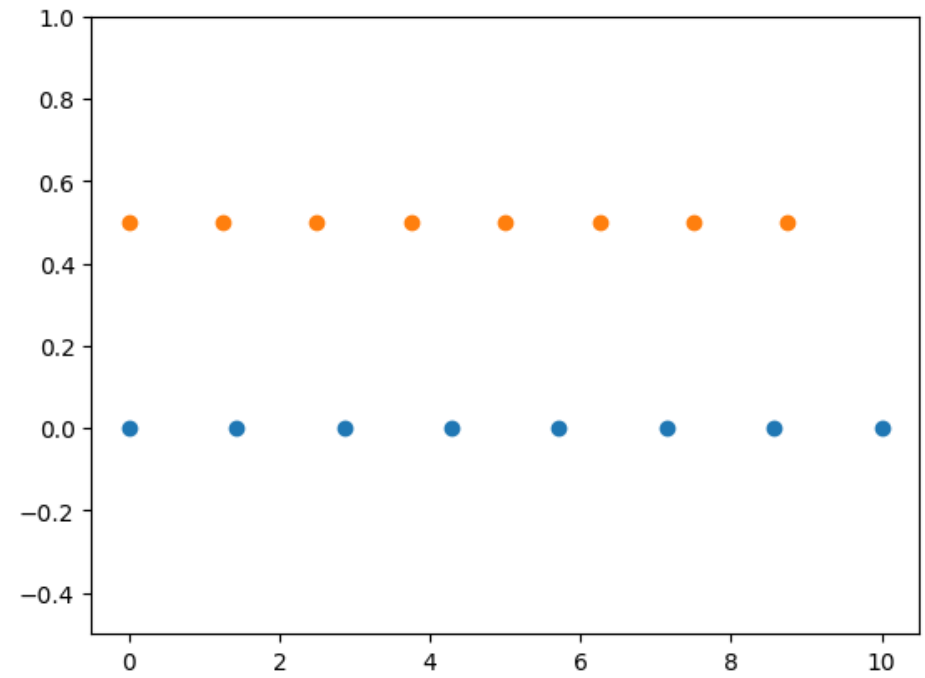
```
(array([2.  , 2.25, 2.5 , 2.75, 3.  ]), 0.25)
[2.   2.25 2.5  2.75 3.  ] <class 'numpy.ndarray'>
0.25 <class 'numpy.float64'>
```

linspace 함수로 생성 1

Linspace로 endpoint를 false로 하면 최종 값은 포함하지 않음

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

N = 8

y = np.zeros(N)
x1 = np.linspace(0, 10, N, endpoint = True)
print(x1)
x2 = np.linspace(0, 10, N, endpoint = False)
print(x2)

plt.plot(x1, y, 'o')
plt.plot(x2, y + 0.5, 'o')
plt.ylim([-0.5, 1])
plt.show()
```

```
[ 0.          1.42857143  2.85714286  4.28571429  5.71428571  7.14285714
  8.57142857 10.        ]
[0.   1.25 2.5  3.75 5.   6.25 7.5  8.75]
```



linspace 함수로 생성 2

1차원 ndarray를 생성하고 증가된 값 단위를 알고 싶으면 retstep 인자를 True하여 튜틀로 받아 확인하면 됨

```
import numpy as np

print(np.linspace(1, 10, 10))

samples, spacing = np.linspace(1, 10, 10, retstep = True)
print(spacing)
samples, spacing = np.linspace(1, 10, 20, endpoint = True, retstep = True)
print(samples)
print(spacing)
samples, spacing = np.linspace(1, 10, 20, endpoint = False, retstep = True)
print(spacing)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
1.0
[ 1.          1.47368421  1.94736842  2.42105263  2.89473684  3.36842105
  3.84210526  4.31578947  4.78947368  5.26315789  5.73684211  6.21052632
  6.68421053  7.15789474  7.63157895  8.10526316  8.57894737  9.05263158
  9.52631579 10.        ]
0.47368421052631576
0.45
```

연습문제

1. 모든 요소가 1인 4x4 크기의 NumPy 배열을 생성하십시오.
2. 요소가 모두 0인 5x5 크기의 NumPy 배열을 생성하십시오.
3. 1에서 20까지의 짝수를 가지는 NumPy 배열을 생성하십시오.

연습문제 코드

```
import numpy as np

# 1번 문제
arr = np.ones((4, 4))
print(arr)

# 2번 문제
arr = np.zeros((5, 5))
print(arr)

# 3번 문제
arr = np.arange(2, 21, 2)
print(arr)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
[ 2  4  6  8 10 12 14 16 18 20]
```

Numpy data type

Data type

numpy 내에 정의된 데이터 타입

구분		Type	Example
숫자형 (numeric)	bool형 (booleans)	bool	[True, True, False, False]
	정수형 (integers)	int8 (i1) int16 (i2) int32 (i4) int64 (i8)	[-2, -1, 0, 1, 2, 3]
	부호없는 (양수) 정수형 (unsigned integers)	uint8 (u1) uint16 (u2) uint32 (u4) uint64 (u8)	[2, 1, 0, 1, 2, 3]
	부동소수형 (floating points)	float16 (f2) float32 (f4) float64 (f8)	[-2.0, -1.3, 0.0, 1.9, 2.2, 3.6]
	복수수형(실수 + 허수) (complex)	complex64 (c8) complex128 (c16)	(1 + 2j)
문자형 (character)	문자형 (string)	string_ (s)	['Seoul', 'Busan', 'Incheon'] 고정길이 문자열형
	유니코드 (string)	unicode_ (u)	['Seoul', 'Busan', 'Incheon'] 고정길이 유니코드

Numpy data type 지정

ndarray를 생성에 필요한 데이터 타입을 정의하기 위한 클래스

```
# importing numpy module
import numpy as np

# making array with data type of float64: dtype = np.float64
x_float64 = np.array([1.4, 2.6, 3.0, 4.9, 5.32], dtype = np.float64)

# checking data type: dtype method
print(x_float64.dtype)
print(x_float64)

# making array with data type of float64: np.float64()
x_float64_2 = np.float64([1.4, 2.6, 3.0, 4.9, 5.32])

print(x_float64_2.dtype)
print(x_float64_2)
```

```
float64
[1.4  2.6  3.   4.9  5.32]
float64
[1.4  2.6  3.   4.9  5.32]
```

데이터 type 변환

float64의 소수점 부분이 int64로 변환 이후에는 잘림(truncated).

```
import numpy as np

# original data
x_float64 = np.array([1.4, 2.6, 3.0, 4.9, 5.32], dtype = np.float64)
print(x_float64)

# object.astype(np.int64)
# the decimal parts are truncated
x_int64 = x_float64.astype(np.int64)

print(x_int64.dtype)
print(x_int64)
```

```
[1.4  2.6  3.   4.9  5.32]
int64
[1 2 3 4 5]
```

Python의 int와 NumPy의 int64

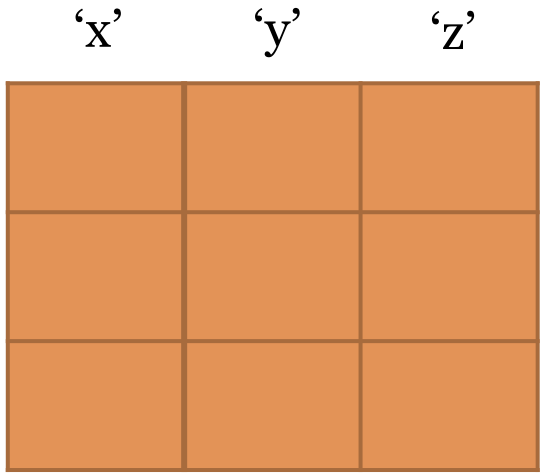
연산자는 큰 차이가 없지만 methods, attributes에서는 numpy에서 약 8배 많음 (numpy가 강력한 이유)

```
In [23]: x_py = 12345        --> 8개 methods,
In [24]: x_np = np.int64(12345)  --> 69개
```

# python's native int methods and attributes	# numpy's int64 methods and attributes
['bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']	**attributes** ['all', 'any', 'argmax','argmin', 'argsort','astype', 'base','byteswap', 'choose','clip', 'compress','conj', 'conjugate', 'copy', 'cumprod', 'cumsum', ...]

Ndarray 내부 원소에 이름 부여

칼럼별 처리를 위해 index 이외의 이름을 부여하여 직접 접근하여 처리



array[‘x’] 로 접근하면 ‘x’ 칼럼에 대해 전부 접근 가능

```
import numpy as np

xz = np.linspace(1, 10, 5, dtype = [('x', int)])
print(xz)
print(xz.ndim)
print(xz.shape)
print(xz.dtype)
print(xz['x'])
xz['x'].fill(10)
print(xz)
```

```
[( 1,) ( 3,) ( 5,) ( 7,) (10,)]
1
(5,)
[('x', '<i8')]
[ 1  3  5  7 10]
[(10,) (10,) (10,) (10,) (10,)]
```

dtype에 칼럼명 정의 : tuple

dtype 정의시 칼럼명, 칼럼값을 튜플로 정의하면 칼럼을 명으로 조회가 가능

np.dtype([(컬럼명, type, 자리수)])

```
import numpy as np

dt = np.dtype([('a', np.string_, 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'])
```

```
|S10
float64
[(b'aaa', 10000.)]
[b'aaa']
[10000.]
```

```
import numpy as np

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'])
```

```
|S10
float64
[(b'aaa', 10000.)]
[b'aaa']
[10000.]
```

dtype에 칼럼명 정의 : dict

dtype 내의 dict 내 에 names에 칼럼명 정의, formats에 타입정의 또는 칼럼명과 타입으로 정의해서 사용가능

```
import numpy as np

dict_type = {'names' : ['col1', 'col2', 'col3'], 'formats' : [np.int_, np.float_, np.float_]}

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.uint, 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.)]
[('col1', '<i8'), ('col2', '<f8'), ('col3', '<f8')]
[(10, 0., 0.) (22, 0., 0.) (44, 0., 0.)]
[(b'dahl', 20) (b'park', 30)]
[b'dahl' b'park']
```

dtype에 칼럼명 정의 : 배열처리

ndarray 생성하면 내부를 (int, float)를 원소로 한 2차원 배열이 생성됨

```
import numpy as np

xz = np.zeros((2, 2), dtype = [('x', int), ('y', float)])
print(xz)
print(xz.ndim)
print(xz.shape)
print(xz.dtype)
print(xz['x'])
print(xz['y'])

xz['x'].fill(10)
print(xz)
```

```
[[(0, 0.) (0, 0.)]
 [(0, 0.) (0, 0.)]]
2
(2, 2)
[('x', '<i8'), ('y', '<f8')]
[[0 0]
 [0 0]]
[[0. 0.]
 [0. 0.]]
[[(10, 0.) (10, 0.)]
 [(10, 0.) (10, 0.)]]
```

칼럼 필드명 변경

dtype 내의 names 변수로 칼럼 필드를 조회 및 갱신이 가능

```
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')
```

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

칼럼명 접근

numpy.array 생성시 sequence 각 요소에 대해 접근변수와 타입을 정할 수 있음

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

인덱스를 찾고 내부의 이름으로 검색

해당 이름에 해당되는 위치의 모든 값을 nda rray 타입으 로 출력

칼럼명 접근 : fancy

numpy.array를 여러 칼럼단위로 접근시는 실제 칼럼명을 내부에 리스트에 넣어서 검색

```
import numpy as np

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

print(x)
print(x[['x', 'y']])
print(x[['x', 'value']])
```

```
[(1.5, 2.5, [[1., 2.], [1., 2.]]) (3. , 4. , [[4., 5.], [4., 5.]])
 (1. , 3. , [[2., 6.], [2., 6.]])]
[(1.5, 2.5) (3. , 4. ) (1. , 3. )]
[(1.5, [[1., 2.], [1., 2.]]) (3. , [[4., 5.], [4., 5.]])
 (1. , [[2., 6.], [2., 6.]])]
```

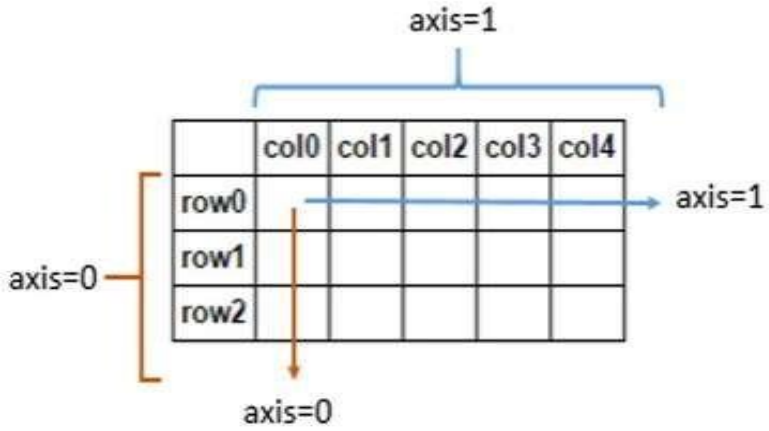
Indexing, slicing

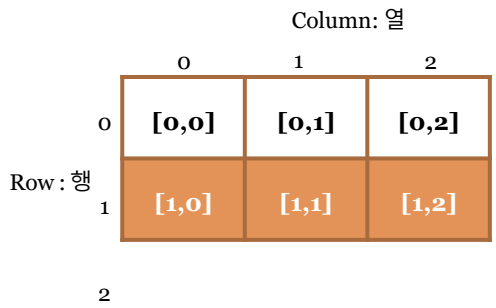
axis 이해하기 : 2차원

Axis는 배열의 축을 나타내며 0은 열이고, 1은 행을 표시

```
import numpy as np

f = np.arange(0, 6).reshape(2, 3)
print(f)

# column
print(np.mean(f, axis = 0))
# row
print(np.mean(f, axis = 1))
```

```
[[0 1 2]
 [3 4 5]]
[1.5 2.5 3.5]
[1. 4.]
```



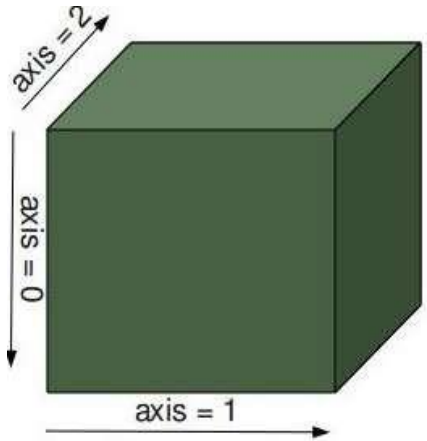


axis 이해하기 : 3차원

Axis는 배열의 축을 0은 두개의 행렬에서 각 열원소별로, 1은 행끼리 처리, 2은 내부 원소끼리 처리

```
import numpy as np

f = np.arange(0, 8).reshape(2, 2, 2)
print(f)

# [[0, 1] + [4, 5], [2, 3] + [6, 7]]
print(np.mean(f, axis = 0))
# [[0, 1] + [2, 3], [4, 5] + [6, 7]]
print(np.mean(f, axis = 1))
# [[0 + 1, 2 + 3], [4 + 5, 6 + 7]]
print(np.mean(f, axis = 2))
```

```
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
[[2. 3.]
 [4. 5.]]
[[1. 2.]
 [5. 6.]]
[[0.5 2.5]
 [4.5 6.5]]
```



Column: 열

Row: 행	0	1
0	0	1
1	2	3

Column: 열

Row: 행	0	1
0	4	5
1	6	7

axis 이해하기

Axis는 배열의 축을 나타내며 0은 열이고, 1은 행을 표시

```
import numpy as np

a = np.arange(6)
b = np.arange(6).reshape(2, 3)
a[5] = 100

print(a)
print(b)
print(a[np.argmax(a)])

print(np.argmax(b, axis = 0))    # 열 처리
print(np.argmax(b, axis = 1))
```

```
[  0   1   2   3   4 100]
[[0 1 2]
 [3 4 5]]
100
[1 1 1]
[2 2]
```

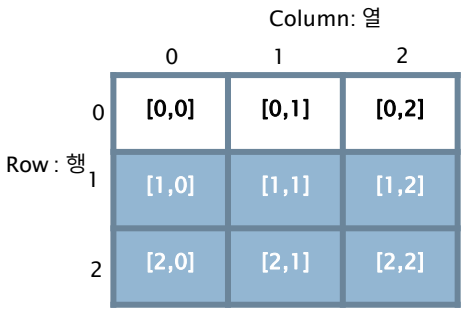
배열 접근하기 : 행과 열구분

배열명[행 범위, 열 범위] 행으로 접근, 열로 접근

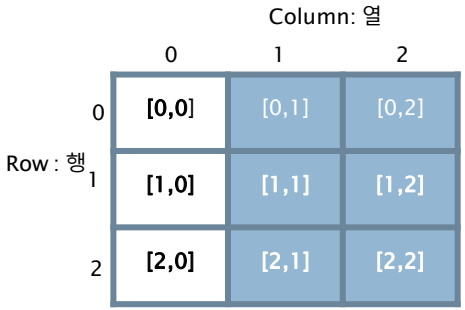
```
import numpy as np

l33 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
np33 = np.array(l33, dtype = int)

print(np33.shape)
print(np33.ndim)
print(np33)

print("(first row", np33[0])
print("first column", np33[:, 0])
```

```
(3, 3)
2
[[1 2 3]
 [4 5 6]
 [7 8 9]]
(first row [1 2 3]
first column [1 4 7]
```

첫번째 행 접근



첫번째 열 접근

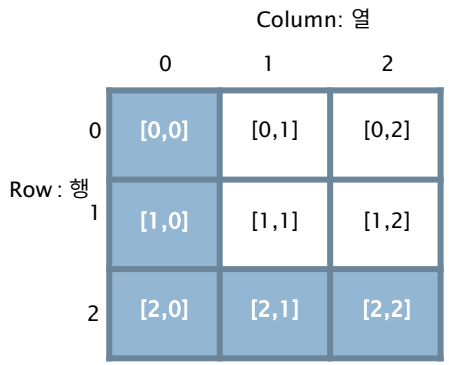


배열 접근하기 : 행렬로 구분

첫번째와 두번째 행과 두번째와 세번째 열로 접근

```
import numpy as np

l33 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
np33 = np.array(l33, int)
print(np33)
print(np33[:2, 1:])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[2 3]
 [5 6]]
```

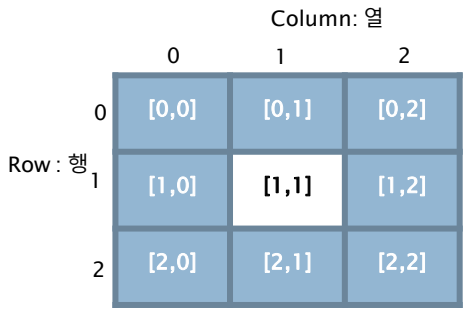


배열 접근하기 : 값

행과 열의 인덱스를 지정하면 실제 값에 접근해서 보여줌

```
import numpy as np

l33 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
np33 = np.array(l33, int)

print(np33)
print(np33[1, 1])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
5
```



행 검색

정수배열을 사용한 색인(양수,음수를 이용)이며 행에 대한 정보를 list로 제공해서 3번째와 1번째를 출력

정방향

```
import numpy as np

f = np.arange(0, 12).reshape(3, 4)
print(f)

print(f[[2, 0]])
```

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

역방향

```
import numpy as np

f = np.arange(0, 12).reshape(3, 4)
print(f)

print(f[[-1, -3]])
```

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

배열 접근하기 : 값

1d-Slicing

행과 열의 인덱스를 지정하면 실제 값에 접근해서 보여줌

```
import numpy as np

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

print(arr[1:5])
print(arr[4:])
print(arr[:4])
print(arr[-3:-1])
print(arr[1:5:2])
print(arr[::2])
```

```
[2 3 4 5]
[5 6 7]
[1 2 3 4]
[5 6]
[2 4]
[1 3 5 7]
```

배열 접근하기 : 값

2d-Slicing

행과 열의 인덱스를 지정하면 실제 값에 접근해서 보여줌

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5],
                [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
print(arr[0:2, 2])
print(arr[0:2, 1:4])
```

```
[7 8 9]
[3 8]
[[2 3 4]
 [7 8 9]]
```

Fancy indexing : boolean

Ndarray 내부의 요소들을 ndarray 인덱싱으로 접근해서 추출 하는 방식

```
import numpy as np

B = np.array([[142, 56, 189, 65],
              [299, 288, 10, 12],
              [55, 142, 17, 18]])
print(B>=82)

d = B[B>82]
print(d)
```

```
[[ True False  True False]
 [ True  True False False]
 [False  True False False]]
[142 189 299 288 142]
```

Fancy indexing : 숫자 1행

행축과 열축을 조합해서 처리하므로 e0는 첫번째 행만 e1은 열에 대해 처리

```
import numpy as np

B = np.array([[142, 56, 189, 65],
              [299, 288, 10, 12],
              [55, 142, 17, 18]])
# 첫번째 행에 대한 위치 조정 후 출력
e0 = np.array([0, 0, 0, 0])
e1 = np.array([0, 3, 2, 1])

f = B[(e0, e1)]
print(f)
```

```
[142  65 189  56]
```

Fancy indexing : 숫자 여러 행

행축과 열축을 조합해서 처리하므로 e0는 첫번째 첫번째 행과 두번째 행만 e1은 열에 대해 처리

```
import numpy as np

B = np.array([[142, 56, 189, 65],
              [299, 288, 10, 12],
              [55, 142, 17, 18]])
# 첫번째 행에 대한 위치 조정 후 출력
e0 = np.array([0, 0, 1, 1])
e1 = np.array([0, 3, 2, 1])

f = B[(e0, e1)]
print(f)
```

```
[142  65  10 288]
```

Fancy indexing :expression

ndarray 내의 값이 3으로 나눠지지 않은 요소 검색을 위해 인덱스에 로직 처리 후 추출하기

```
import numpy as np

A = np.array([3, 4, 6, 10, 24, 89, 45, 43, 46, 99, 100])
# 3으로 나눠지지 않는 수 추출
print(A%3 != 0)
g = A[A%3 != 0]
print(g)
```

```
[False  True False  True False  True False  True  True False  True]
[  4  10  89  43  46 100]
```

Fancy indexing : nonzero

Nonzero 메소드를 이용해서 zero 값이 아닌 행 인덱스와 열 인덱스를 ndarray로 전환해서 처리를 확인해서 추출하기

```
import numpy as np

Z = np.array([[142, 56, 0, 65],
              [0, 288, 10, 12],
              [55, 142, 0, 18]])

z = Z[Z.nonzero()]
print(Z.nonzero())
print(z)

print(np.count_nonzero(Z))
print(np.flatnonzero(Z))
```

```
(array([0, 0, 0, 1, 1, 1, 2, 2, 2]), array([0, 1, 3, 1, 2, 3, 0, 1, 3]))
[142  56  65 288  10  12  55 142  18]
9
[ 0  1  3  5  6  7  8  9 11]
```

연습문제

1. 1부터 9까지의 수를 원소로 가지는 3x3 크기의 2차원 배열을 생성하고, 이 배열에서 마지막 행을 제외한 모든 행을 선택하여 출력하세요.
2. 무작위 값을 가지는 8x8 크기의 2차원 배열을 생성하고, 이 배열에서 첫 번째 열과 마지막 열, 첫 번째 행과 마지막 행을 제외한 부분을 선택하여 출력하세요..

연습문제 코드

```
import numpy as np

# 1번 문제
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr[:-1])

# 2번 문제
arr = np.random.rand(8, 8)
print(arr[1:-1, 1:-1])
```

```
[[1 2 3]
 [4 5 6]]
[[0.65780892 0.20106182 0.82536207 0.57803154 0.98170178 0.03534729]
 [0.14740965 0.70157403 0.62387107 0.12839445 0.42026322 0.23275107]
 [0.08843885 0.88632035 0.02055575 0.43310826 0.57602892 0.35894404]
 [0.05098333 0.45614459 0.29046027 0.99358119 0.9264235  0.15146649]
 [0.13016794 0.55418438 0.30501768 0.52991863 0.28638581 0.19103727]
 [0.31634945 0.36246807 0.40031942 0.80432218 0.15324918 0.02531952]]
```

연산

수치계산

Ndarray에 대한 수치 계산

+ : 배열간 덧셈

- : 배열간 뺄셈

* : 배열간 곱셈

/ : 배열간 나눗셈

** : 배열간 제곱

% : 배열간 나머지

```
import numpy as np

x = np.array([1, 5, 2])
y = np.array([7, 4, 1])
print(x + y)
print(x * y)
print(x - y)
print(x / y)
print(x % y)
```

```
[8 9 3]
[ 7 20  2]
[-6  1  1]
[0.14285714 1.25       2.        ]
[1 1 0]
```

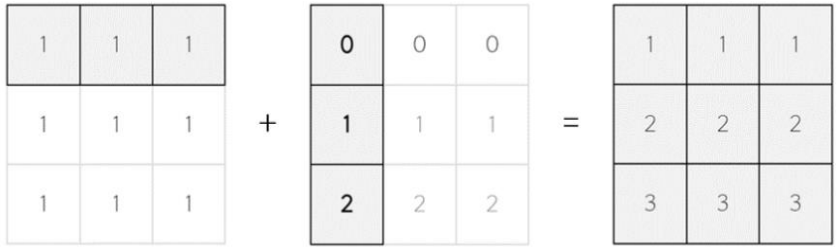
Broadcasting

Ndarry간 크기가 맞지 않을때, 자동으로 크기를 전파하여 연산을 수행

1. 원소가 하나인 배열은 어떤 배열이나 브로드캐스팅이 가능
2. 하나의 배열이 1차원 배열인 경우, 브로드캐스팅이 가능
3. 차원의 짝이 맞을때 브로드캐스팅이 가능


```
import numpy as np

arr1 = np.array([[0, 0, 0],
                 [1, 1, 1],
                 [2, 2, 2]])
arr2 = np.array([5, 6, 7])
print(arr1 + arr2)
```

```
[[5 6 7]
 [6 7 8]
 [7 8 9]]
```




```
import numpy as np

arr3 = np.array([1, 1, 1])
arr4 = np.array([[0], [1], [2]])
print(np.add(arr3, arr4))
```

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

Dot 연산 :ndarray

배열과 배열의 요소들을 곱하고 전체를 덧셈을 처리하고 ndarray로 리턴

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

```
import numpy as np

bb = np.array([1, 2, 3])
cc = np.array([-7, 8, 9])
print(np.dot(bb, cc))
```

```
36
```

Dot 연산 : ndarray

행렬(2행 2열)과 행렬(2행 2열)을 곱하면 결과는 2행 2열의 행렬로 처리하고 ndarray타입으로 리턴

```
import numpy as np

xs = np.array(((2, 3), (3, 5)))
ys = np.array(((1, 2), (5, -1)))
print(np.dot(xs, ys), type(np.dot(xs, ys)))
```

```
[[17  1]
 [28  1]] <class 'numpy.ndarray'>
```

Dot 연산 : ndarray (4,3)*(3,1)

ndarray 를 행렬 연산하고 (4,1) 배열로 만들고 100으로 나눠서 결과값 출력

```
import numpy as np

persons = np.array([[100, 175, 210], [90, 160, 150], [200, 50, 100], [120, 0, 310]])
print(" persons shape ", persons.shape)
Price_per_100_g = np.array([2.98, 3.90, 1.99])
print(Price_per_100_g.shape)
Price_in_Cent = np.dot(persons, Price_per_100_g)
print(Price_in_Cent, type(Price_in_Cent, ))
Price_in_Euro = Price_in_Cent / np.array([100, 100, 100, 100])
print(Price_in_Euro)
```

```
 persons shape  (4, 3)
(3,)
[1398.4 1190.7  990.   974.5] <class 'numpy.ndarray'>
[13.984 11.907  9.9    9.745]
```

cross 연산

벡터곱 연산을 np.cross 함수를 이용하여 처리

```
import numpy as np

x = np.array([0, 0, 1])
y = np.array([0, 1, 0])

print(np.cross(x, y))
print(np.cross(y, x))
```

```
[-1  0  0]
[1 0 0]
```

ndarray 와 비교연산 처리

ndarray와 ndarray간의 비교연산. Scala 값은 broadcasting하므로 ndarray 동일 모형의 동일값으로 인지해서 처리된 후 bool값을 가지는 ndarray 가 생성됨

ndarray = ndarray 비교연산 (>, <, ==) ndarray

```
import numpy as np

A = np.array([4, 7, 3, 4, 2, 8])
C = (A == 4)
print(C, type(C))

LT = (A < 5)
print(LT, type(LT))
```

```
[ True False False  True False False] <class 'numpy.ndarray'>
[ True False  True  True  True False] <class 'numpy.ndarray'>
```

```
import numpy as np

B = np.array([[142, 56, 189, 65],
              [299, 288, 10, 12],
              [55, 142, 17, 18]])
print(B>=82)
```

```
[[ True False  True False]
 [ True  True False False]
 [False  True False False]]
```

bool-> int로 전환

비교연산 결과가 bool 타입을 astype 메소드로 값을 전환

```
import numpy as np

B = np.array([[142, 56, 189, 65],
              [299, 288, 10, 12],
              [55, 142, 17, 18]])
print(B>=82)

b = B>82
c = b.astype(int)
print(c)
```

```
[[ True False  True False]
 [ True  True False False]
 [False  True False False]]
[[1 0 1 0]
 [1 1 0 0]
 [0 1 0 0]]
```

연습문제

1. 0부터 50까지 2의 배수로 채워진 NumPy 배열을 생성하고, 배열의 요소 중 10보다 크고 30보다 작은 모든 요소의 합계를 구하세요.

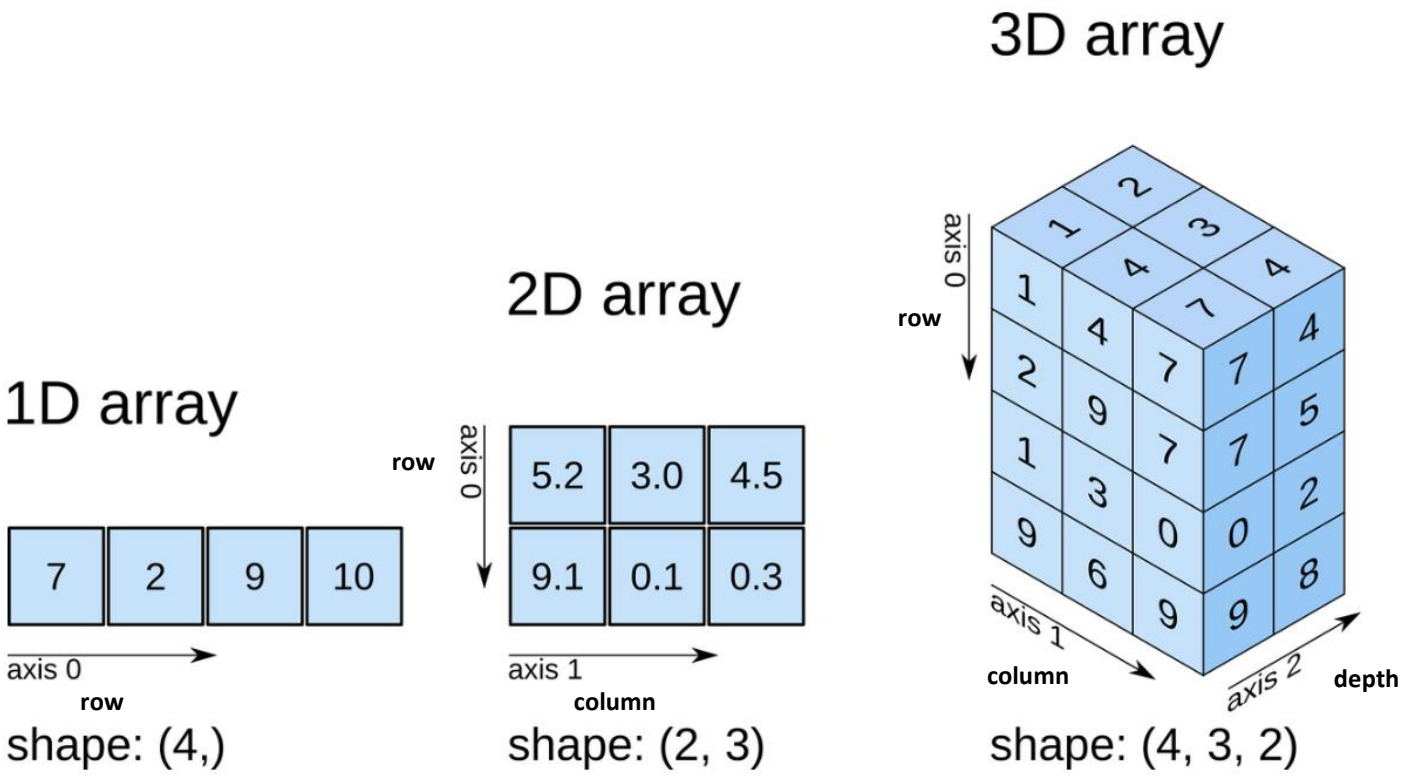
연습문제 코드

```
import numpy as np

# 1번 문제

# 0부터 50까지 2의 배수로 채워진 numpy 배열 생성
arr = np.arange(0, 51, 2)

# 배열의 요소 중 10보다 크고 30보다 작은 요소의 합계
result = np.sum(arr[(arr > 10) & (arr < 30)])
print(result)
```

```
180
```

Numpy 기본 함수

Data axis 개념


1D array
7
2
9
10
axis 0
row
shape: (4,)
2D array
row
axis 0
5.2
3.0
4.5
9.1
0.1
0.3
axis 1
column
shape: (2, 3)
3D array
axis 0
row
axis 1
column
axis 2
depth
shape: (4, 3, 2)

reshape 함수

Reshape(a, newshape, order='C') 배열의 크기를 변경 하는 함수

마지막 자리는 "-1"를 채워 자동 계산 가능

원소의 수 = #(R) x #(C) x #(d)

```
import numpy as np

x = np.array(range(12))
print(np.shape(x))
y = x.reshape((3, 4, 1))
z = x.reshape((3, 4, -1))
print(np.shape(y))
r, c, d = np.shape(z)
a = r * c * d
print(a)

if y.all() == z.all():
  print("same")
```

```
(12,)
(3, 4, 1)
12
same
```

np.newaxis 변수

np.newaxis 변수는 numpy array의 차원 증가
→ 1차원 배열을 vector화 하기 위해 많이 사용

```
import numpy as np

x = np.array([2, 5, 18, 14, 4])
print(np.shape(x))
print(x)
y = x[:, np.newaxis]
print(np.shape(y))
print(y)
```

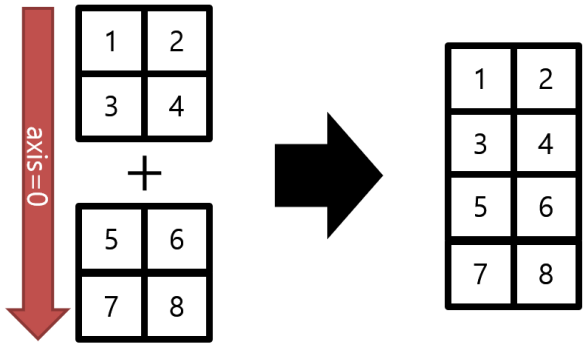
```
(5,)
[ 2  5 18 14  4]
(5, 1)
[[ 2]
 [ 5]
 [18]
 [14]
 [ 4]]
```

```
import numpy as np

x = np.arange(6)
y = x.reshape(2, 3)
z = y[np.newaxis, :]
print(y)
print(np.shape(y))
print(z)
print(np.shape(z))
```

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

concatenate 함수

concatenate((a1, a2, ...), axis=0)로 array를 연결




```
import numpy as np

x = np.array([11, 22])
y = np.array([18, 7, 6])
z = np.array([1, 3, 5])
c = np.concatenate((x, y, z))
print(c)
```

```
[11 22 18  7  6  1  3  5]
```

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
c = np.concatenate((a, b), axis = 0)
print(np.shape(c))
print(c)
c = np.concatenate((a, b), axis = 1)
print(np.shape(c))
print(c)
```

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

stack 함수

Concat. 함수와 유사하지만 새로운 축에 의해 결합하는 함수

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
c = np.concatenate((a, b), axis = 0)
print(c)
d = np.stack((a, b), axis = 0)     # ValueError
print(d)
```

```
[[1 2]
 [3 4]
 [5 6]]
---------------------------------------------------------------------
ValueError                                Traceback
(most recent call last)
<ipython-input-8-f4a20e03025e> in <cell line: 7>()
      5 c = np.concatenate((a, b), axis = 0)
      6 print(c)
----> 7 d = np.stack((a, b), axis = 0)
      8 print(d)
```

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
print(np.shape(a))
print(np.shape(b))
c = np.concatenate((a, b), axis = 0)
d = np.stack((a, b), axis = 0)
print(d)
print(np.shape(d))
```

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

 [[5 6]
  [7 8]]]
(2, 2, 2)
```

stack 함수 : Quiz

Axis에 따른 배열의 Shape 은 ??

```
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 2
a3_2 = np.stack((a1, a2), axis=2)
a3_2.shape:
```

9	21
10	22
11	23
12	24

5	17
6	18
7	19
8	20

1	13
2	14
3	15
4	16

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

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

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

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

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

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

9	10	11	12
21	22	23	24

5	6	7	8
17	18	19	20

1	2	3	4
13	14	15	16

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

stack 함수 : Quiz

Axis에 따른 배열의 Shape 은 ??

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

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

1	13	5	17
2	14	6	18
3	15	7	19
4	16	8	20

9	21
10	22
11	23
12	24

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

# retrieve a1
a3_0[0]
a3_0[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

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

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

1	2	3	4
13	14	15	16

5	6	7	8
17	18	19	20

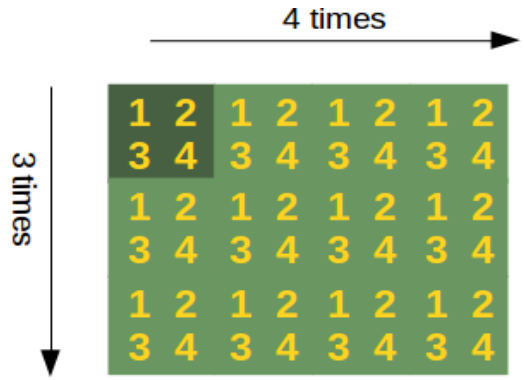
9	10	11	12
21	22	23	24

tile

A 배열에 대한 Reps는 axis 축에 따른 반복을 표시

numpy.tile(A, reps)

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



1. Reps가 스칼라 값은 배수만큼 증가
2. Reps가 벡터값 일 경우 행과 열에 따라 추가

```
import numpy as np

# List에서 ndarray
a = [1, 2, 3]
b = np.tile(a, 2)
print(b)
b = np.tile(a, (2, 1))
print(b)
b = np.tile(a, (2, 2))
print(b)
```

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

Flatten 함수

ndarray에 대한 shape를 1차 ndarray로 전환
메모리에 새로 할당

```
import numpy as np

A = np.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]]])
Flattened_X = A.flatten()
print(Flattened_X)
print(A.flatten(order = "C"))
print(A.flatten(order = "F"))
print(A.flatten(order = "A"))
```

'F': fortran 타입은
칼럼 순으로 flat 처리함

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 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]
[ 0  8 16  2 10 18  4 12 20  6 14 22  1  9 17  3 11 19  5 13 21  7 15 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]
```

ravel 함수

ndarray에 대한 shape를 1차 ndarray로 전환
기존 메모리 그대로 사용

```
import numpy as np

A = np.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]]])
print(A.ravel())
print(A.ravel(order = "C"))
print(A.ravel(order = "F"))
print(A.ravel(order = "A"))
print(A.ravel(order = "K"))
```

'F': fortran 타입은
칼럼 순으로 flat 처리함
'K': 메모리에 있는 그대로 처리

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 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]
[ 0  8 16  2 10 18  4 12 20  6 14 22  1  9 17  3 11 19  5 13 21  7 15 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]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

save/load

생성된 ndarray를 파일에 저장(확장자: npy)했다가 다시 load해서 처리가 가능

```
import numpy as np

nparr = np.array([1, 2, 3.0])
print(nparr)
s = open('arraystore.npy', 'wb')
np.save(s, nparr)
%ls
s = open('arraystore.npy', 'rb')
ndx = np.load(s)
print(ndx, type(ndx))
s.seek(0)
print(s.read())
```

```
[1. 2. 3.]
arraystore.npy  sample_data/
[1. 2. 3.] <class 'numpy.ndarray'>
b"₩x93NUMPY₩x01₩x00v₩x00{'descr': '<f8', 'fortran_order': False, 'shap
```

주요 난수 생성 함수

rand(d0, d1, ..., dn)	주어진 모양에 대해 0에서 1사이의 균등 분포 생성
randn(d0, d1, ..., dn)	주어진 모양에 대해 정규분포 값을 생성
randint(low[, high, size])	Return random integers from *low* (inclusive) to *high* (exclusive).
random_sample([size])	Return random floats in the half-open interval [0.0, 1.0).
random([size])	Return random floats in the half-open interval [0.0, 1.0).
ranf([size])	Return random floats in the half-open interval [0.0, 1.0).
sample([size])	Return random floats in the half-open interval [0.0, 1.0).
choice(a[, size, replace, p])	Generates a random sample from a given 1-D array ..
bytes(length)	Return random bytes.

rand : uniform distribution

rand(균등분포)에 따라 ndarray 를 생성 모양이 없을 경우는 scalar 값을 생성

```
import numpy as np

help(np.random.rand)
```

```
Help on built-in function rand:

rand(...) method of numpy.random.mtrand.RandomState instan
    rand(d0, d1, ..., dn)

    Random values in a given shape.

    .. note::
        This is a convenience function for users porting c
        and wraps `random_sample`. That function takes a
        tuple to specify the size of the output, which is
        other NumPy functions like `numpy.zeros` and `nump

    Create an array of the given shape and populate it wit
    random samples from a uniform distribution
    over ``[0, 1)``.

    Parameters
    ----------
    d0, d1, ..., dn : int, optional
        The dimensions of the returned array, must be non-
        If no argument is given a single Python float is r
```

```
import numpy as np

a = np.random.rand(3, 2)
print(a)

b = np.random.rand(3, 3, 3)
print(b)
```

```
[[0.56045562 0.13598121]
 [0.85159731 0.9717135 ]
 [0.56406381 0.87235469]]
[[[0.41728969 0.66366119 0.67256595]
  [0.21230009 0.22611234 0.21839137]
  [0.34989298 0.02142837 0.52179488]]

 [[0.57597557 0.37060067 0.35684035]
  [0.48881287 0.29262743 0.96119483]
  [0.10520446 0.70033453 0.59161889]]

 [[0.72312173 0.26213526 0.59094355]
  [0.91590714 0.65001887 0.75414864]
  [0.69330161 0.06734861 0.74122397]]]
```

randn : "standard normal"distribution

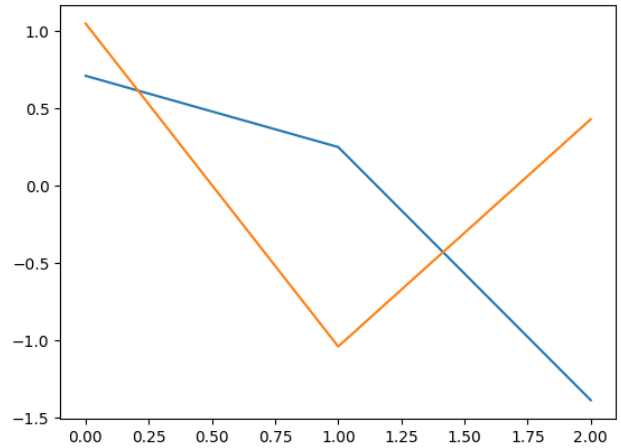
randn(정규분포)에 따라 ndarray 를 생성 모양이 없을 경우는 scalar 값을 생성

```
import numpy as np

help(np.random.randn)
```

```
Help on built-in function randn:

randn(...) method of numpy.random.mtrand.RandomState insta
    randn(d0, d1, ..., dn)

    Return a sample (or samples) from the "standard normal

    .. note::
        This is a convenience function for users porting c
        and wraps `standard_normal`. That function takes a
        tuple to specify the size of the output, which is
        other NumPy functions like `numpy.zeros` and `nump

    .. note::
        New code should use the ``standard_normal`` method
        instance instead; please see the :ref:`random-quic

    If positive int_like arguments are provided, `randn` g
    of shape ``(d0, d1, ..., dn)``, filled
    with random floats sampled from a univariate "normal"
    distribution of mean 0 and variance 1. A single float
    from the distribution is returned if no argument is pr
```

```
import numpy as np
import matplotlib.pyplot as plt

a = np.random.randn(3, 2)
print(a)

b = np.random.randn(3, 3, 3)
print(b)

plt.plot(a)
plt.show()
```

```
[[ 0.71247635  1.05033096]
 [ 0.25153842 -1.03850471]
 [-1.38558222  0.43173528]]
[[[ 0.24470292  0.71898136 -1.85296038]
  [ 1.699092    0.28926248  0.08137626]
  [ 0.34363602 -1.17506949  0.25325196]]

 [[ 1.08628745 -0.76852261 -1.5623215 ]
  [-2.22542224 -0.32620751 -2.74931884]
  [-0.86365995  0.96106425  0.53288764]]

 [[-0.52708001  0.20321533 -0.75334295]
  [ 0.34138826  0.26188398 -0.92599567]
  [-1.2160026  -0.61929712  0.30672198]]]
```



randint

randint(low, high=None, size=None)는 최저값, 최고값-1, 총 길이 인자를 넣어 ndarray로 리턴

Size에 tuple로 선언시 다차원 생성

```
import numpy as np

help(np.random.randint)
```

```
Help on built-in function randint:

randint(...) method of numpy.random.mtrand.RandomState
    randint(low, high=None, size=None, dtype=int)

    Return random integers from `low` (inclusive) to `

    Return random integers from the "discrete uniform"
    the specified dtype in the "half-open" interval [`
    `high` is None (the default), then results are fro

    .. note::
        New code should use the ``integers`` method of
        instance instead; please see the :ref:`random-

    Parameters
    ----------
    low : int or array-like of ints
        Lowest (signed) integers to be drawn from the
        ``high=None``, in which case this parameter is
        *highest* such integer).
```

```
import numpy as np

outcome = np.random.randint(1, 7, size = 10)
print(outcome)
print(type(outcome))
print(len(outcome))

print(np.random.randint(2, size = 10))

print(np.random.randint(1, size = 10))
print(np.random.randint(5, size = (2, 4)))
```

```
[2 6 1 1 6 6 5 4 6 5]
<class 'numpy.ndarray'>
10
[0 1 0 0 0 0 1 0 1 1]
[0 0 0 0 0 0 0 0 0 0]
[[2 1 2 1]
 [0 4 4 0]]
```

random_sample

random_sample(size=None)에 size가 없을 경우는 하나의 값만 생성하고 size를 주면 ndarray를 생

```
import numpy as np

help(np.random.random_sample)
```

```
Help on built-in function random_sample:

random_sample(...) method of numpy.random.mtrand.RandomSt
    random_sample(size=None)

    Return random floats in the half-open interval [0.0,

    Results are from the "continuous uniform" distributi
    stated interval.  To sample :math:`Unif[a, b), b > a`
    the output of `random_sample` by `(b-a)` and add `a`

      (b - a) * random_sample() + a

    .. note::
        New code should use the ``random`` method of a ``
        instance instead; please see the :ref:`random-qu

    Parameters
    ----------
    size : int or tuple of ints, optional
        Output shape.  If the given shape is, e.g., ``(m.
```

```
import numpy as np

print(np.random.random_sample(5))

x = np.random.random_sample((3, 4))
print(x)
a = -3.4
b = 5.9
A = (b - a) * np.random.random_sample((3, 4)) + a
print(A)
```

```
[0.5796268  0.86735489 0.80649691 0.56621571 0.76842581]
[[0.78469306 0.0995265  0.70317086 0.65862406]
 [0.0527525  0.818297   0.70123835 0.00140744]
 [0.2575395  0.19382862 0.28793171 0.86577305]]
[[-2.32375236 -1.02703924  5.71362914  0.61347297]
 [ 4.83079399  3.37289734  3.65584258 -1.18802723]
 [ 5.4028105  -1.73165811  1.57010921 -3.29704796]]
```

ranf

ranf(size=None)에 size가 없을 경우는 하나의 값만 생성하고 size를 주면 ndarray를 생성

```
import numpy as np

print(np.random.ranf(5))
print(np.random.random_sample(5))
print(np.random.ranf((2, 2)))
```

```
[0.71180353 0.556609   0.68523597 0.78713119 0.2171693 ]
[0.85196398 0.27879624 0.66192991 0.04477933 0.92865317]
[[0.68447836 0.30761533]
 [0.09036449 0.8163114 ]]
```

random

random(size=None)에 size가 없을 경우는 하나의 값만 생성하고 size를 주면 ndarray를 생성

```
import numpy as np

print(np.random.random(5))
print(np.random.random_sample(5))
print(np.random.random((2, 2)))
```

```
[0.09417228 0.83604248 0.0106523  0.7776963  0.91021598]
[0.71888825 0.45604324 0.8536176  0.36229309 0.5497288 ]
[[0.18890232 0.0787348 ]
 [0.9991561  0.8924328 ]]
```

seed

seed는 반복적인 random을 동일한 범주에서 처리하기 위한 방식

```
import numpy as np

help(np.random.seed)
```

```
Help on built-in function seed:

seed(...) method of numpy.random.mtrand.RandomState insta
    seed(self, seed=None)

    Reseed a legacy MT19937 BitGenerator

    Notes
    -----
    This is a convenience, legacy function.

    The best practice is to **not** reseed a BitGenerator
    recreate a new one. This method is here for legacy re
    This example demonstrates best practice.

    >>> from numpy.random import MT19937
    >>> from numpy.random import RandomState, SeedSequenc
    >>> rs = RandomState(MT19937(SeedSequence(123456789)))
    # Later, you want to restart the stream
    >>> rs = RandomState(MT19937(SeedSequence(987654321)))
```

```
import numpy as np

for i in range(5):
  arr = np.arange(5)    # [0, 1, 2, 3, 4]
  np.random.seed(1)     # Reset random state
  np.random.shuffle(arr)
  print(arr)
```

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

choice

choice(a, size=None, replace=True, p=None)

A값을 int, size는 모형, p는 나오는 원소에 대한 확률을 정의

```
import numpy as np

help(np.random.choice)
```

```
Help on built-in function choice:

choice(...) method of numpy.random.mtrand.RandomState ins
    choice(a, size=None, replace=True, p=None)

    Generates a random sample from a given 1-D array

    .. versionadded:: 1.7.0

    .. note::
        New code should use the ``choice`` method of a ``
        instance instead; please see the :ref:`random-qu

    Parameters
    ----------
    a : 1-D array-like or int
        If an ndarray, a random sample is generated from
        If an int, the random sample is generated as if
    size : int or tuple of ints, optional
        Output shape.  If the given shape is, e.g., ``(m,
        ``m * n * k`` samples are drawn.  Default is None
```

```
import numpy as np

# This is equivalent to np.random.randint(0, 5, 3)
print(np.random.choice(5, 3))
# p값은 선택되는 확률
# 0, 1, 2, 3, 4의 확률
print(np.random.choice(5, 3, p = [0.1, 0, 0.3, 0.6, 0]))

print(np.random.choice(5, (3, 3)))
```

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

Permutation

선택된 배열의 원소를 섞기

```
import numpy as np

help(np.random.permutation)
```

```
Help on built-in function permutation:

permutation(...) method of numpy.random.mtrand.RandomState insta
    permutation(x)

    Randomly permute a sequence, or return a permuted range.

    If `x` is a multi-dimensional array, it is only shuffled alo
    first index.

    .. note::
        New code should use the ``permutation`` method of a ``de
        instance instead; please see the :ref:`random-quick-star

    Parameters
    ----------
    x : int or array_like
        If `x` is an integer, randomly permute ``np.arange(x)``.
        If `x` is an array, make a copy and shuffle the elements
        randomly.
```

```
import numpy as np

arr = np.random.permutation(10)
print(arr)

arr1 = np.random.permutation([1, 4, 9, 12, 15])
print(arr1)

arr2 = np.arange(9).reshape((3, 3))
print(arr2)
arr3 = np.random.permutation(arr2)
print(arr3)
```

```
[8 3 5 9 0 6 1 7 4 2]
[ 9  1 15 12  4]
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[0 1 2]
 [6 7 8]
 [3 4 5]]
```

shuffle

선택된 배열의 원소를 섞기

```
import numpy as np

help(np.random.shuffle)
```

```
Help on built-in function shuffle:

shuffle(...) method of numpy.random.mtrand.RandomState instance
    shuffle(x)

    Modify a sequence in-place by shuffling its contents.

    This function only shuffles the array along the first axis o
    multi-dimensional array. The order of sub-arrays is changed
    their contents remains the same.

    .. note::
        New code should use the ``shuffle`` method of a ``defaul
        instance instead; please see the :ref:`random-quick-star

    Parameters
    ----------
    x : ndarray or MutableSequence
        The array, list or mutable sequence to be shuffled.

    Returns
```

```
import numpy as np

arr = np.arange(10)
print(arr)
np.random.shuffle(arr)
print(arr)

arr2 = np.arange(9).reshape((3, 3))
print(arr2)
np.random.shuffle(arr2)
print(arr2)
```

```
[0 1 2 3 4 5 6 7 8 9]
[6 8 2 4 3 5 7 9 1 0]
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[3 4 5]
 [6 7 8]
 [0 1 2]]
```

RandomState : 생성

size를 argument로 취하는데 기본값은 None. 만약 size가 None이라면, 하나의 값이 생성되고 반환. 만약 size가 정수라면, 1-D 행렬이 랜덤변수들로 채워져 반환된다.

```
import numpy as np

rng = np.random.RandomState(10)
z = np.asarray(rng.uniform(size = (2, 5)))
print(z)
z1 = np.asarray(rng.standard_normal(size = (2, 5)))
print(z1)
```

```
[[0.77132064 0.02075195 0.63364823 0.74880388 0.49850701]
 [0.22479665 0.19806286 0.76053071 0.16911084 0.08833981]]
[[ 0.26551159  0.10854853  0.00429143 -0.17460021  0.43302619]
 [ 1.20303737 -0.96506567  1.02827408  0.22863013  0.44513761]]
```

RandomState :seed/get_state

Seed는 반복 가능한 것을 처리할 때 사용하면 get-state()로 처리하면 현재 상태가 출력

```
import numpy as np

np.random.seed(1234)
print(np.random.uniform(0, 10, 5))

r = np.random.RandomState(1234)
print(r.uniform(0, 10, 5))
print(r.get_state())
```

```
[1.9151945  6.22108771 4.37727739 7.85358584 7.79975808]
[1.9151945  6.22108771 4.37727739 7.85358584 7.79975808]
('MT19937', array([2260313690,  348938374, 3392255680, 29090337(
       1016917445, 4051655600,  976942074, 1628339371,  9329899
        417988570, 3106230116, 3847402493, 2846838083, 18540650
       2365406610,  631390710, 3006558680, 1855109059,  2300643
        758538135, 1999313224, 2345696623, 4174662269,  28056111
       1706268812, 4182435209, 1014638053,  610687375, 23315256
       3432349290, 1302213857, 2461808965, 1211193860, 31200042
        159403718,  785407708, 1103582039, 2181742160, 40034748(
       3333684546, 2164025542, 3329631014, 3331897623,   448415(
       2124190575, 4103716897, 1985760015, 3231349092, 25792233(
       2045506447, 1684183393, 4105280043,  264622240, 3457386
```

Binomial : 이항분포

n은 trial , p는 구간 [0,1]에 성공 P는 확률
이항 분포에서 작성한 것임

$$P(N) = \binom{n}{N} p^N (1-p)^{n-N},$$

```
import numpy as np

# number of trials, probability of each trial
n, p = 10, .5
# result of flipping a coin 10 times, tested 1000 times
s = np.random.binomial(n, p, 100)
print(s)
a = sum(np.random.binomial(9, 0.1, 20000) == 0) / 20000.
# answer = 0.38885, or 38%
print(a)
```

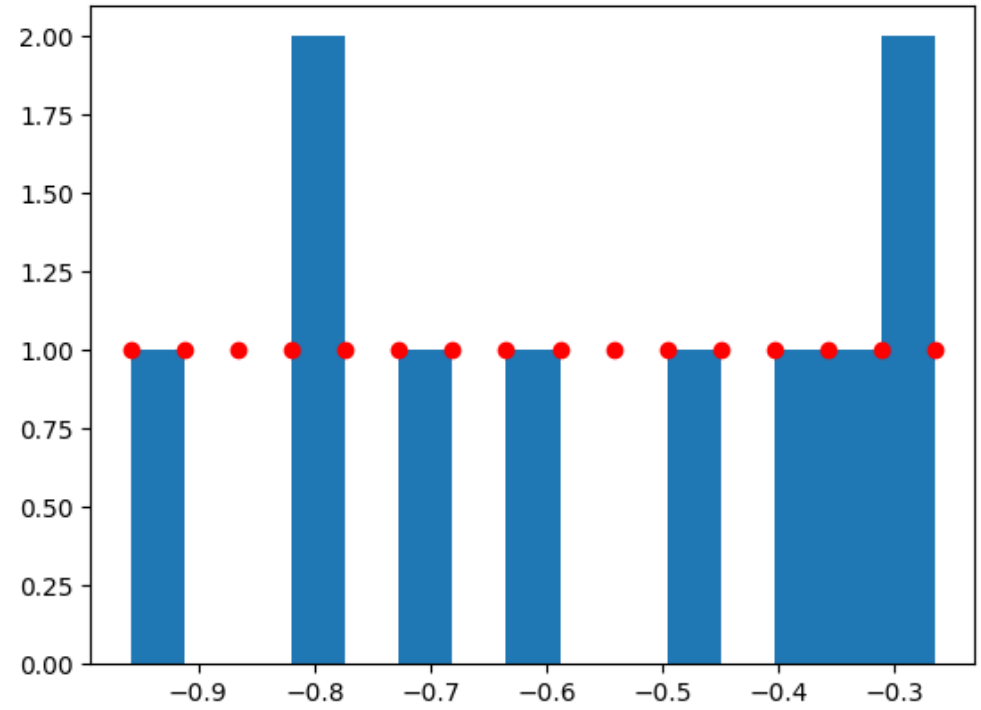
```
[4 4 6 8 7 4 5 6 6 4 5 5 2 6 7 4 5 3 4 7 6 5 6 4 5 7 5 6 3 6 6 4 7 5 7 3 4
 2 6 5 5 2 5 4 5 3 5 5 1 5 7 6 9 8 6 4 6 5 4 5 2 5 8 3 3 6 5 5 3 4 7 5 5 1
 4 5 5 7 6 6 3 6 7 6 5 3 5 5 8 5 5 5 6 3 6 6 6 6 5 8]
0.38475
```

Uniform

[low, high)는 low를 포함하지만 high를 포함하지 않는 정규분로를 표시

```
import numpy as np
import matplotlib.pyplot as plt

s = np.random.uniform(-1, 0, 10)
print(s)
print(np.all(s >= -1))
print(np.all(s < 0))

count, bins, ignored = plt.hist(s, bins = 15)
plt.plot(bins, np.ones_like(bins), 'o', color = 'r')
plt.show()
```

```
[-0.46948537 -0.3270837  -0.95798444 -0.69004014 -0.81211721 -0.37556301
 -0.78725304 -0.28333982 -0.26528262 -0.61012888]
True
True
```

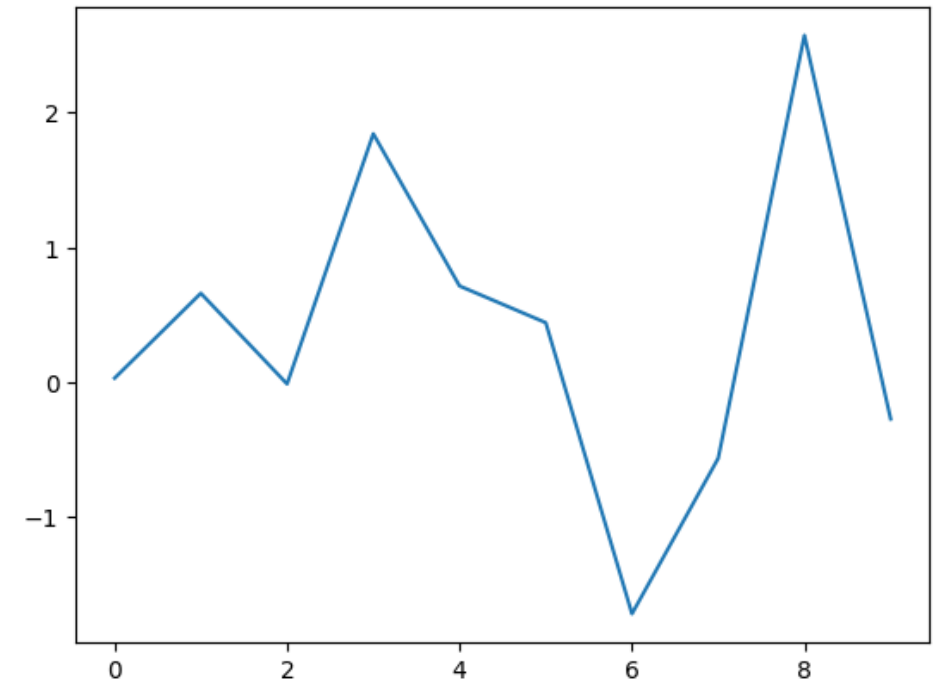


standard_normal

a standard Normal distribution (mean=0, stdev=1) 표시

```
import numpy as np
import matplotlib.pyplot as plt

s = np.random.standard_normal(10)
print(s)
print(s.shape)

plt.plot(s)
plt.show()

s1 = np.random.standard_normal(size = (3, 4, 2))
print(s1.shape)
```

```
[ 0.0301284   0.66014156 -0.01283008  1.84430754  0.71452268  0.4420287
 -1.71929974 -0.56479674  2.57507172 -0.27284241]
(10,)
```




```
(3, 4, 2)
```

Distributions 1

beta(a, b[, size])	Draw samples from a Beta distribution.
binomial(n, p[, size])	Draw samples from a binomial distribution.
chisquare(df[, size])	Draw samples from a chi-square distribution.
dirichlet(alpha[, size])	Draw samples from the Dirichlet distribution.
exponential([scale, size])	Draw samples from an exponential distribution.
f(dfnum, dfden[, size])	Draw samples from an F distribution.
gamma(shape[, scale, size])	Draw samples from a Gamma distribution.
geometric(p[, size])	Draw samples from the geometric distribution.
gumbel([loc, scale, size])	Draw samples from a Gumbel distribution.
hypergeometric(ngood, nbad, nsample[, size])	Draw samples from a Hypergeometric distribution.
laplace([loc, scale, size])	Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay).

Distributions 2

logistic([loc, scale, size])	Draw samples from a logistic distribution.
lognormal([mean, sigma, size])	Draw samples from a log-normal distribution.
logseries(p[, size])	Draw samples from a logarithmic series distribution.
multinomial(n, pvals[, size])	Draw samples from a multinomial distribution.
multivariate_normal(mean, cov[, size])	Draw random samples from a multivariate normal dist r ibution.
negative_binomial(n, p[, size])	Draw samples from a negative binomial distribution.
noncentral_chisquare(df, nonc[, size])	Draw samples from a noncentral chi-square distributi o n.
noncentral_f(dfnum, dfden, nonc[, size])	Draw samples from the noncentral F distribution.
normal([loc, scale, size])	Draw random samples from a normal (Gaussian) distri bution.
pareto(a[, size])	Draw samples from a Pareto II or Lomax distribution w ith specified shape.
poisson([lam, size])	Draw samples from a Poisson distribution.

Distributions 3

power(a[, size])	Draws samples in [0, 1] from a power distribution with positive e x ponent a - 1.
rayleigh([scale, size])	Draw samples from a Rayleigh distribution.
standard_cauchy([size])	Draw samples from a standard Cauchy distribution with mode = 0 .
standard_exponential([size])	Draw samples from the standard exponential distribution.
standard_gamma(shape[, size])	Draw samples from a standard Gamma distribution.
standard_normal([size])	Draw samples from a standard Normal distribution (mean=0, std e v=1).
standard_t(df[, size])	Draw samples from a standard Student's t distribution with *df* de g rees of freedom.
triangular(left, mode, right[, size])	Draw samples from the triangular distribution.
uniform([low, high, size])	Draw samples from a uniform distribution.
vonmises(mu, kappa[, size])	Draw samples from a von Mises distribution.
wald(mean, scale[, size])	Draw samples from a Wald, or inverse Gaussian, distribution.
weibull(a[, size])	Draw samples from a Weibull distribution.
zipf(a[, size])	Draw samples from a Zipf distribution.

hypot 함수

sqrt 함수를 ndarray 모듈도 처리할 수 있는 함 수

```
import numpy as np

print(np.hypot(np.array([3]), np.array([4])))
print(np.hypot(3, 4))

np.sqrt(3**2 + 4**2)

np.hypot(3*np.ones((3, 3)), [4])
```

```
[5.]
5.0
array([[5., 5., 5.],
       [5., 5., 5.],
       [5., 5., 5.]])
```

Nonzero 확인 함수

count_nonzero 함수를 이용해서 갯수확인 및 flatnonzero 함수를 이용해서 인덱스를 식별

```
import numpy as np

Z = np.array([[142, 56, 0, 65],
              [0, 288, 10, 12],
              [55, 142, 0, 18]])

z = Z[Z.nonzero()]
print(Z.nonzero())
print(z)

print(np.count_nonzero(Z))
print(np.flatnonzero(Z))
```

```
(array([0, 0, 0, 1, 1, 1, 2, 2, 2]), array([0, 1, 3, 1, 2, 3, 0, 1, 3]))
[142  56  65 288  10  12  55 142  18]
9
[ 0  1  3  5  6  7  8  9 11]
```

Sum 함수

Axis에 대한 인자가 없을 경우 전체를 합산하고 axis가 0이면 칼럼 합을 구하고 axis가 1이면 행에 대한 합을 계산

```
import numpy as np

b = np.arange(12).reshape(3, 4)
print(b)

print(np.sum(b))
# sum of each column
print(np.sum(b, axis = 0))
# sum of each row
print(np.sum(b, axis = 1))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
66
[12 15 18 21]
[ 6 22 38]
```

cumsum 함수

모형을 유지하면 행과 열로 누적된 값을 계산하는 함수

```
import numpy as np

b = np.arange(12).reshape(3, 4)
print(b)

print(np.sum(b))
# cumulative sum along each row
print(np.cumsum(b, axis = 1))
# cumulative sum along each column
print(np.cumsum(b, axis = 0))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
66
[[ 0  1  3  6]
 [ 4  9 15 22]
 [ 8 17 27 38]]
[[ 0  1  2  3]
 [ 4  6  8 10]
 [12 15 18 21]]
```

cumsum 함수: 누적처리 예시

Weights 리스트를 받고 누적값을 산출하여 새로운 리스트 cum_weights 만듬 계산시 오차는 발생함

```
import numpy as np

weights = [0.2, 0.5, 0.3]
cum_weights = [0] + list(np.cumsum(weights))

print(cum_weights)
```

```
[0, 0.2, 0.7, 1.0]
```

절대값/부호 처리

abs, fabs : 각 원소의 절대값을 구함. 복소수가 아닌 경우에는 fabs로 빠른 연산을 처리

sign: 부호에 대한 처리, 1은 양수, -1은 음수

```
import numpy as np

a = np.linspace(1, 10, 10)
print(-a)
print(np.fabs(-a))
s = np.sign(-a)
print(s)
print(a*s)
```

```
[ -1.  -2.  -3.  -4.  -5.  -6.  -7.  -8.  -9. -10.]
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
[ -1.  -2.  -3.  -4.  -5.  -6.  -7.  -8.  -9. -10.]
```

지수와 로그

exp는 지수 계산

log,log10,log2,log1p는 자연로그, 로그10, 로그2, 로그(1+x)

```
import numpy as np

a = np.linspace(1, 10, 10)
print(np.exp(a))
print(np.log(a))
print(np.log10(a))
print(np.log2(a))
print(np.log1p(a))
print(np.log(1+a))
```

```
[2.71828183e+00 7.38905610e+00 2.00855369e+01 5.45981500e+01
 1.48413159e+02 4.03428793e+02 1.09663316e+03 2.98095799e+03
 8.10308393e+03 2.20264658e+04]
[0.         0.69314718 1.09861229 1.38629436 1.60943791 1.79175947
 1.94591015 2.07944154 2.19722458 2.30258509]
[0.         0.30103    0.47712125 0.60205999 0.69897    0.77815125
 0.84509804 0.90308999 0.95424251 1.         ]
[0.         1.         1.5849625  2.         2.32192809 2.5849625
 2.80735492 3.         3.169925   3.32192809]
[0.69314718 1.09861229 1.38629436 1.60943791 1.79175947 1.94591015
 2.07944154 2.19722458 2.30258509 2.39789527]
[0.69314718 1.09861229 1.38629436 1.60943791 1.79175947 1.94591015
 2.07944154 2.19722458 2.30258509 2.39789527]
```

거듭제곱/제곱근

square는 거듭제곱, sqrt는 제곱근, power는 두 배열의 거듭제곱

```
import numpy as np

a = np.linspace(1, 10, 10)
print(np.square(a))
print(np.sqrt(a))
print(np.power(a, 2))
```

```
[  1.   4.   9.  16.  25.  36.  49.  64.  81. 100.]
[1.         1.41421356 1.73205081 2.         2.23606798 2.44948974
 2.64575131 2.82842712 3.         3.16227766]
[  1.   4.   9.  16.  25.  36.  49.  64.  81. 100.]
```

절사/절상

ceil : 각 원소의 값보다 같거나 큰 정수 중 가장 큰 정수를 반환

floor : 각 원소의 값보다 작거나 같은 정수 중 가장 작은수 반환

rint : 각 원소의 소수자리를 반올림하고 dtype 유지

```
import numpy as np

b = np.linspace(1, 5, 10)
print(b)
print(np.ceil(b))
print(np.floor(b))
print(np.rint(b))
```

```
[1.         1.44444444 1.88888889 2.33333333 2.77777778 3.22222222
 3.66666667 4.11111111 4.55555556 5.        ]
[1. 2. 2. 3. 3. 4. 4. 5. 5. 5.]
[1. 1. 1. 2. 2. 3. 3. 4. 4. 5.]
[1. 1. 2. 2. 3. 3. 4. 4. 5. 5.]
```

사칙연산

add, subtract, multiply, divide, floor_divide 처리

```
import numpy as np

a = np.linspace(1, 10, 10)
print(a)
print(np.add(a, a))
print(np.subtract(a, a))
print(np.multiply(a, a))
print(np.divide(a, a))
print(np.floor_divide(a, a))
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[ 2.  4.  6.  8. 10. 12. 14. 16. 18. 20.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[  1.   4.   9.  16.  25.  36.  49.  64.  81. 100.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Mod 연산

modf : 단항연산으로 자신의 나머지와 몫 구하기
mod : 이항연산으로 나머지만 구함

```
import numpy as np

a = np.linspace(1, 10, 10)
print(a)
print(np.modf(a))
print(np.mod(1+a, a))
```

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

관계연산

두 배열간의 관계를 표시하거나 부울 표시된 결과에 대해 any/all로 전체 결과를 확인

```
import numpy as np

a = np.linspace(1, 10, 10)
print(np.equal(a, a))
print(np.not_equal(a, a))
# a > 3
b = np.greater(a, 3)
print(b)
print(np.any(b))
print(np.all(b))
# a >= 3
c = np.greater_equal(a, 3)
print(c)
# a < 3
d = np.less(a, 3)
print(d)
# a <= 3
e = np.less_equal(a, 3)
print(e)
```

```
[ True  True  True  True  True  True  True  True  True  True]
[False False False False False False False False False False]
[False False False  True  True  True  True  True  True  True]
True
False
[False False  True  True  True  True  True  True  True  True]
[ True  True False False False False False False False False]
[ True  True  True False False False False False False False]
```

where 연산

where 연산은 조건을 표현한 배열을 기준으로 True일 경우 첫번째 배열의 요소, False일 경우는 두번째 배열의 요소로 처리

```
import numpy as np

a = np.linspace(1, 10, 10)
print(a)
result = np.where(a>3, 3, 0)
print(result)

b = a>3
print(b)
result = np.where(b, a, 0)
print(result)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[0 0 0 3 3 3 3 3 3 3]
[False False False  True  True  True  True  True  True  True]
[ 0.  0.  0.  4.  5.  6.  7.  8.  9. 10.]
```

hypot 함수

hypot 함수

sqrt($a^2 + b^2$)을 처리하는 함수

```
import numpy as np

print(np.hypot(np.array([3]), np.array([4])))
print(np.hypot(3, 4))

np.sqrt(3**2 + 4**2)

np.hypot(3 * np.ones((3, 3)), [4])
```

```
[5.]
5.0
array([[5., 5., 5.],
       [5., 5., 5.],
       [5., 5., 5.]])
```

unique/in1d

집합처럼 만드는 unique 함수와 포함관계를 표시하는 in1d함수

```
import numpy as np

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

# 집합생성
A = np.unique(y)
print(A)
# 집합 내의 포함된원소인지 확인
print(np.in1d([2, 3, 7], A))
```

```
[1 2 3 4 5]
[ True  True False]
```

집합연산하기

집합연산하기

집합연산은 일단 1차원적으로 변환해서 합집합, 교집합, 차집합, 대칭차집합처리

```
import numpy as np

y = np.array([1, 2, 3, 4, 5, 1, 2, 3])
z = np.array([1, 2, 3, 1, 2, 3])

# 집합생성
A = np.unique(y)
B = np.unique(z)

# 합집합, 교집합, 차집합, 대칭차집합
print(np.union1d(A, B))
print(np.intersect1d(A, B))
print(np.setdiff1d(A, B))
print(np.setxor1d(A, B))
```

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

```
import numpy as np

y = np.array([[1, 2, 3, 4], [5, 1, 2, 3]])
z = np.array([[1, 2, 3], [1, 2, 3]])

# 집합생성
A = np.unique(y)
B = np.unique(z)

# 합집합, 교집합, 차집합, 대칭차집합
print(np.union1d(A, B))
print(np.intersect1d(A, B))
print(np.setdiff1d(A, B))
print(np.setxor1d(A, B))
```

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

합/평균/분산/표준편차

통계 기본 함수 제공하며 cumsum/cumprod는 각 원소의 값을 누적하는 함수

```
import numpy as np

a = np.linspace(1, 10, 10)
print(a)
print(np.sum(a))
print(np.cumsum(a))
print(np.cumprod(a))
print(np.mean(a))
print(np.var(a))
print(np.std(a))
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
55.0
[ 1.  3.  6. 10. 15. 21. 28. 36. 45. 55.]
[1.0000e+00 2.0000e+00 6.0000e+00 2.4000e+01 1.2000e+02 7.2000e+02
 5.0400e+03 4.0320e+04 3.6288e+05 3.6288e+06]
5.5
8.25
2.8722813232690143
```

최대값/최소값

값이 최대값/최소값를 구하거나 값의 최대값과 최소값의 인덱스를 구함

```
import numpy as np

a = np.linspace(1, 10, 10)
b = a + 1
print(np.min(a))
print(np.max(a))
print(np.minimum(a, b))
print(np.maximum(a, b))
```

```
1.0
10.0
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[ 2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
```

```
import numpy as np

arr = np.array([9, 19, 29, 39, 49])

print(" index ")
print(arr.argmax())
print(arr.argmin())

print(" value ")
print(arr[np.argmax(arr)])
print(arr[np.argmin(arr)])
```

```
 index
4
0
 value
49
9
```

연습문제

1. 무작위 값을 가지는 4x4 크기의 두 개의 2차원 배열 A, B를 생성하고, 각 배열을 2x8 크기로 재배열하세요. 그리고 나서 이 두 배열을 수직으로 연결하여 새로운 2차원 배열을 생성하고, 이 배열의 모양(shape)을 출력하세요.

연습문제 코드

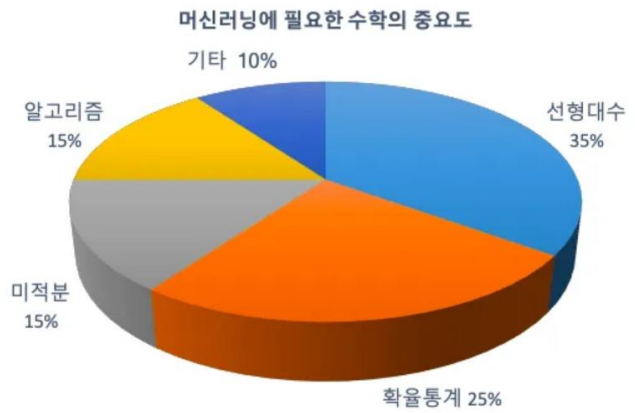
```
import numpy as np

# 1번 문제
# 무작위 값을 가지는 4x4 크기의 두 개의 2차원 배열 A, B 생성
A = np.random.rand(4, 4)
B = np.random.rand(4, 4)

# 각 배열을 2x8 크기로 재배열
A = A.reshape(2, 8)
B = B.reshape(2, 8)

# 두 배열을 수직으로 연결하여 새로운 2차원 배열 생성
C = np.concatenate((A, B), axis = 0)

# 새로운 배열의 모양 출력
print(C.shape)
```

```
(4, 8)
```

선형대수의 기초

Why Linear Algebra?

선형대수는 선형방정식을 풀기 위한 방법론들을 배우는 학문
→ 대량의 데이터를 한번에 처리 할 수 있도록 도움
→ Vector Space같은 인공지능 개념에 대한 이해



$$k_1 = w_1 x_1 + w_2 x_2 + 1 \times b_1$$
$$k_2 = w_1 y_1 + w_2 y_2 + 1 \times b_2$$
$$k_3 = w_1 z_1 + w_2 z_2 + 1 \times b_3$$

$$\mathbf{k} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} x_1 & x_2 & b_1 \\ y_1 & y_2 & b_2 \\ z_1 & z_2 & b_3 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ 1 \end{bmatrix}$$

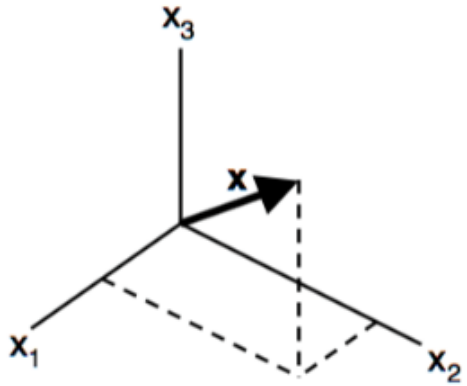
$$\mathbf{k} = \mathbf{A}\mathbf{w}$$

Why Linear Algebra?

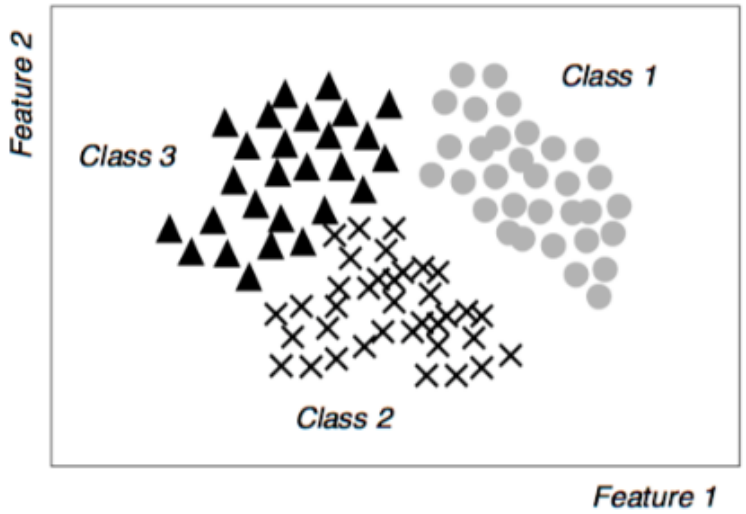
→ 데이터의 특징을 Vector로 표현 가능
→ Vector Space에 feature vector를 이용하여 mapping
→ Vector 공간상의 거리를 이용하여 분류

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \\ x_d \end{bmatrix}$$

Feature vector



Feature space (3D)

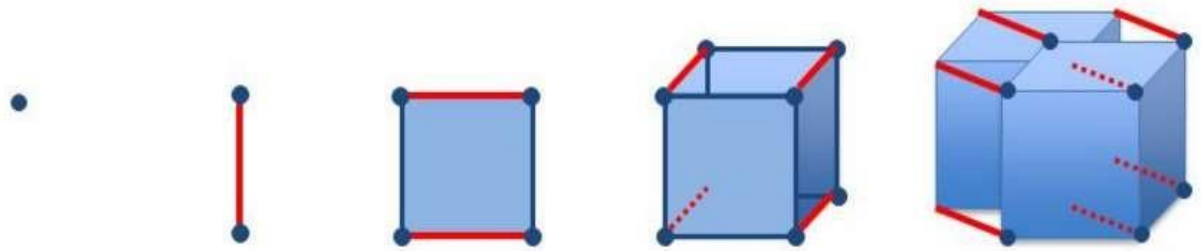


Scatter plot (2D)

기하 차원

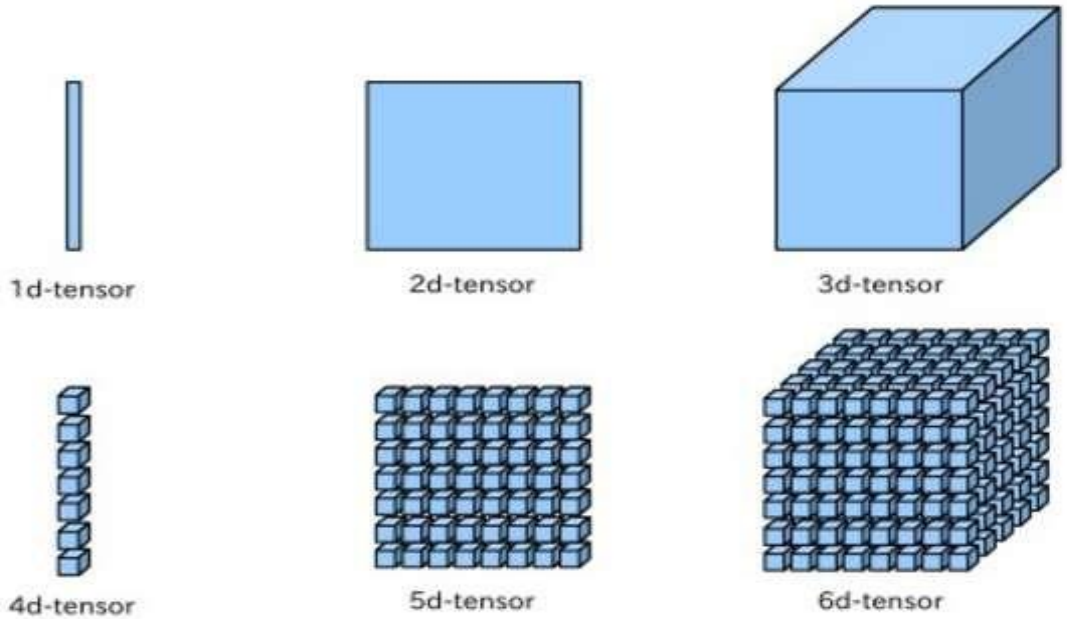
차원(次元)은 수학에서 공간 내에 있는 점 등의 위치를 나타내기 위해 필요한 축의 개수를 말함

Point - Line - Square - Cube - Hypercube - ...
0D 1D 2D 3D 4D 5D



tensor 차원

Tensor는 n차원 array를 표시, 1차원은 벡터, 2차원은 matrix, 3차원은 cube, 4차원은 cube의 , vector, 5차원은 cube의 matrix, 6차원은 cube의 cube



스칼라/벡터/행렬

스칼라는 number, vector는 숫자들의 list(row or column), matrix는 숫자들의 array(rows, columns) 그리고 vector는 Matrix

Scalar

24

Vector

$$\begin{bmatrix} 2 & -8 & 7 \end{bmatrix}$$

row

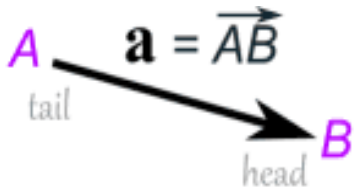
or column

$$\begin{bmatrix} -6 \\ -4 \\ 27 \end{bmatrix}$$

Matrix

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$$

row(s) × column(s)



벡터는 tail부터 head까지의 유향선분으로 표시

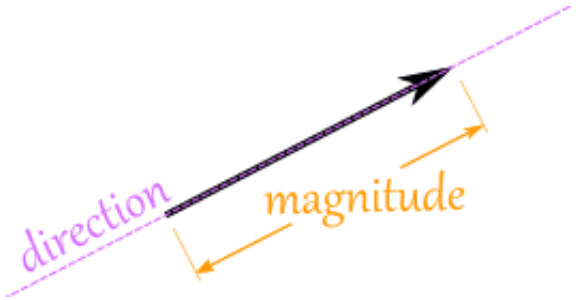
배열과 vertor 구분

ndarray 는 벡터 1xN, Nx1, 그리고 N크기의 1차원 배열이 모두 각각 다르며, 벡터는 그 자체로 특정 좌표를 나타내기도 하지만 방향을 나타냄

scalar	배열	vector
양, 정적 위치	양, 정적 위치	변위, 속도, 힘(방향성)
1차원	N 차원	N 차원
단순 값	행,열 구분 없음	행벡터, 열벡터

벡터 크기

||v|| =sqrt(v0^2 + v1^2 + v2^2... + vn^2) 로 표현



$$||u|| = \sqrt{u_x^2 + u_y^2}$$

벡터 **b** = (6,8) 의 크기

$|\mathbf{b}| = \sqrt{(6^2 + 8^2)} = \sqrt{(36+64)} = \sqrt{100} = 10$

vector 크기

벡터의 길이 즉 크기를 구함

$$\vec{a} = \|\vec{a}\|\,\vec{e}$$

length	:	$\|\vec{a}\|$		
direction vector	:	$\vec{e}$	;	$\|\vec{e}\| = 1$
zero vector	:	$\vec{0}$		
unit vector	:	$\vec{e}$	;	$\|\vec{e}\| = 1$

```
import math
import numpy as np

a = np.array([4, 2, 7])

print(np.linalg.norm(a))
print(math.sqrt(sum([n**2 for n in a])))

b = np.array([0, 0, 0])
print(np.linalg.norm(b))
print(math.sqrt(sum([n**2 for n in b])))

e = np.array([1, 0, 0])
print(np.linalg.norm(e))
print(math.sqrt(sum([n**2 for n in e])))
```

```
8.306623862918075
8.306623862918075
0.0
0.0
1.0
1.0
```

Vector 크기 계산

벡터의 크기(Magnitude)는 원소들의 제곱을 더하고 이에 대한 제곱근의 값

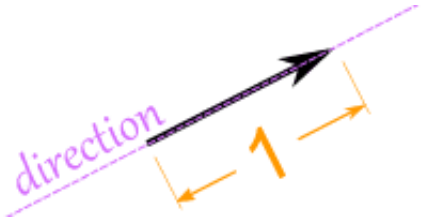
벡터의 크기는 x축의 변위와 y축의 변위를 이용하여 피타고라스 정리

```
import math
import numpy as np

x = np.array([1, 2])
mag = lambda x: math.sqrt(sum(i**2 for i in x))
print(mag(x))

print(np.linalg.norm(x))
```

```
2.23606797749979
2.23606797749979
```

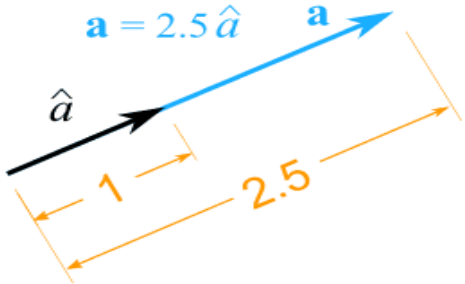
단위벡터

단위벡터(unit vector)는 크기가 1인 벡터



크기가 1인 벡터

표기법은 문자에 모자(hat)을
사용해서 표시



모든 벡터는 단위벡터에 대해 sclae
배수 만큼의 크기를 가진 벡터

단위벡터 정규화

해당 벡터를 0 ~ 1의 값으로 정규화

```
import math
import numpy as np

def add(u, v):
  return [ u[i]+v[i] for i in range(len(u)) ]

def magnitude(v):
  return math.sqrt(sum(v[i]*v[i] for i in range(len(v))))

def normalize(v):
  vmag = magnitude(v)
  return [ v[i]/vmag for i in range(len(v))]

l = [1, 1, 1]
v = [0, 0, 0]
h = normalize(add(l, v))
print(magnitude(add(l, v)))
print(h)
```

```
1.7320508075688772
[0.5773502691896258, 0.5773502691896258, 0.5773502691896258]
```

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|}$$

multiply 함수 : 곱셈

Multiply, Matmul 함수

multiply 함수는 1차원 ndarray에서는 * 연산자와 같은 계산 결과가 나옴

```
import numpy as np

a = np.array([[2, -3], [1, 1]])
b = np.array([1, 1])

print(np.matmul(a, b))
print(a * b)
print(np.multiply(a, b))

a = np.array([[2, -3], [1, 1]])
b = np.array([[1, 1], [2, 2]])

print(np.matmul(a, b))
print(a * b)
print(np.multiply(a, b))
```

```
[-1  2]
[[ 2 -3]
 [ 1  1]]
[[ 2 -3]
 [ 1  1]]
[[-4 -4]
 [ 3  3]]
[[ 2 -3]
 [ 2  2]]
[[ 2 -3]
 [ 2  2]]
```

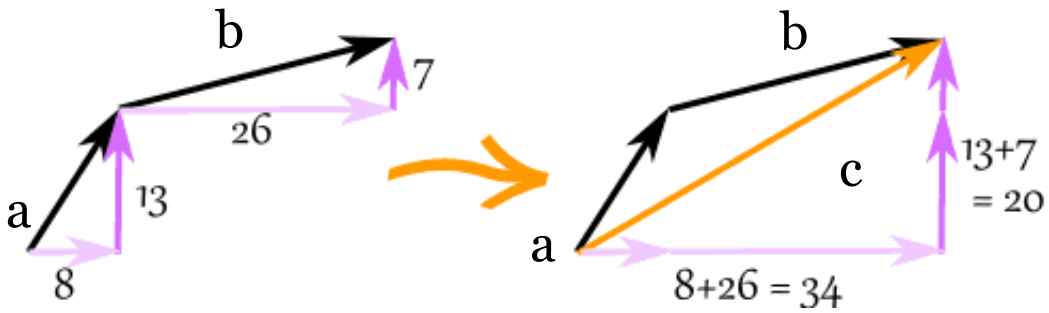
벡터: +

The vector (8,13) and the vector (26,7) add up to the vector (34,20)

Example: add the vectors

a = (8,13) and b = (26,7) c = a + b

c = (8,13) + (26,7) = (8+26,13+7) = (34,20)

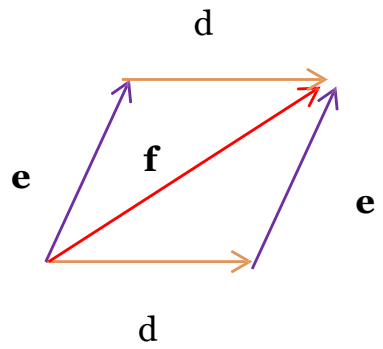


Vector 연산: +

두 벡터 평행 이동해 평행사변형을 만든 후 가운데 벡터가 실제 덧셈한 벡터를 표시

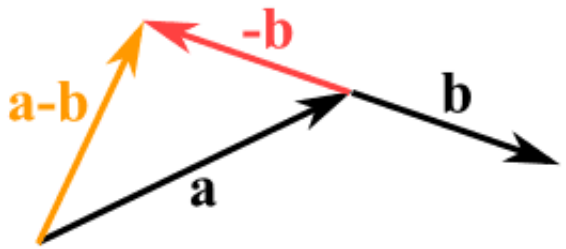
```
import math
import numpy as np

d = np.array([4, 5])
e = np.array([3, 8])

f = d + e
print(f)
```

```
[ 7 13]
```



벡터 : -

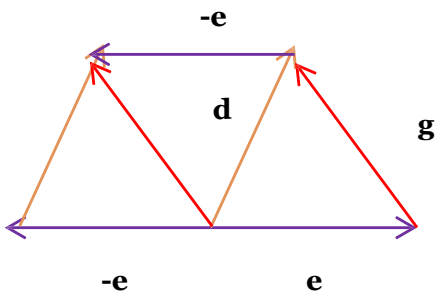
벡터의 방향성을 반대로 이동한 실제 벡터를 처리



Example: subtract **k** = (4,5) from **v** = (12,2)
a = **v** + −**k**
a = (12,2) + −(4,5) = (12,2) + (−4,−5) = (12−4,2−5) = (8,−3)

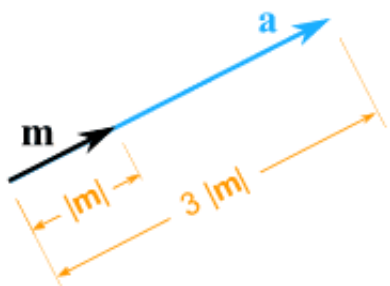
```
import math
import numpy as np

d = np.array([1, 2])
e = np.array([2, 1])
g = d - e
print(g)
```

```
[-1  1]
```



벡터: 스칼라곱

벡터의 각 원소에 스칼라값만큼 곱하여 표시



벡터 m = [7,3]

A = 3m= [21,9]

```
import math
import numpy as np

d = np.array([1, 2])
i = 3 * d
print(i)
```

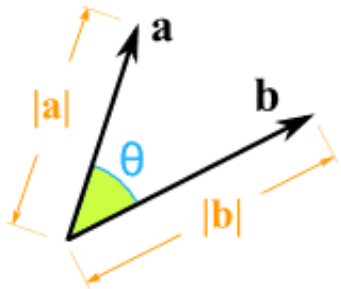
```
[3 6]
```

내적 vs 외적

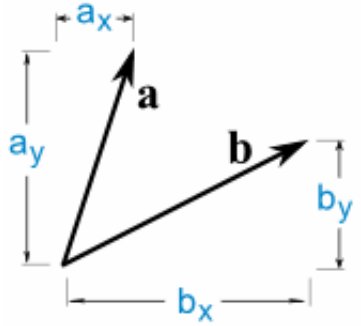
구분	내적	외적
명칭	Inner product, dot product, scalar product	Outer product, vector product, cross product
표기	.(Dot)	X(cross)
대상 벡터	n 차원	3 차원
공식	$a_1 b_1 + a_2 b_2 + ... + a_n b_n$	$(a_2 b_3 - a_3 b_2, a_3 b_1 - a_1 b_3, a_1 b_2 - a_2 b_1)$
	|a||b| cos 각도	|a||b| sin각도 n
결과	scalar	vector

내적 산식

내적(Inner Product)산식은 두벡터의 크기에 cos각을 곱한 결과 또는 두벡터간의 원소들이 곱의 합산과 같은 결과



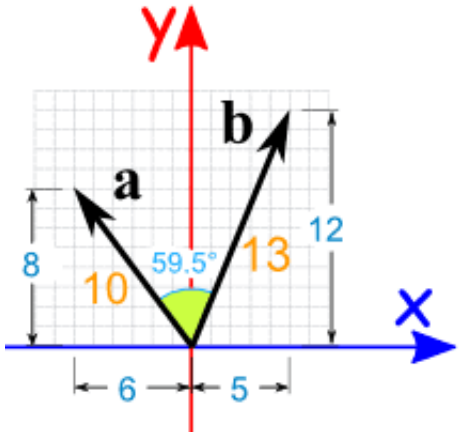
a · **b** = |**a**| × |**b**| × cos(θ)
Where:
|**a**| : vector **a** 크기
|**b**| : vector **b** 크기
θ : **a** and **b** 사이의 각



a · **b** = $a_x \times b_x + a_y \times b_y$

내적 수학적 예시 : 2 차원

두벡터에 내적 연산에 대한 수학적 처리 예시



$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| \times |\mathbf{b}| \times \cos(\theta)$
$\mathbf{a} \cdot \mathbf{b} = 10 \times 13 \times \cos(59.5°)$
$\mathbf{a} \cdot \mathbf{b} = 10 \times 13 \times 0.5075...$
$\mathbf{a} \cdot \mathbf{b} = 65.98... = 66$ (rounded)

$\mathbf{a} \cdot \mathbf{b} = a_x \times b_x + a_y \times b_y$
$\mathbf{a} \cdot \mathbf{b} = -6 \times 5 + 8 \times 12$
$\mathbf{a} \cdot \mathbf{b} = -30 + 96$
$\mathbf{a} \cdot \mathbf{b} = 66$

Dot, inner 함수 (내적)

np.dot, np.inner 모두 사용 가능.
np.dot은 2차원 부터 matrix multifaction을 하기 때문에 2차원 부터는 결과값이 달라짐

```
import numpy as np

A = np.array([1, 2, 3, 4])
B = np.array([5, 6, 7, 8])

print(np.dot(A, B))
print(np.inner(A, B))

A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

print(np.dot(A, B))
print(np.inner(A, B))
```

```
70
70
[[19 22]
 [43 50]]
[[17 23]
 [39 53]]
```

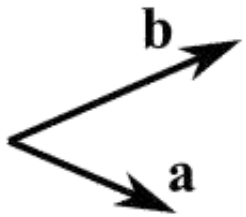
For the (0,0) entry of the result: 1x*5* + *2x*6 = 17
For the (0,1) entry of the result: 1x*7* + *2x*8 = 23
For the (1,0) entry of the result: 3x*5* + *4x*6 = 39
For the (1,1) entry of the result: 3x*7* + *4x*8 = 53

Cross 함수 (vector 곱)

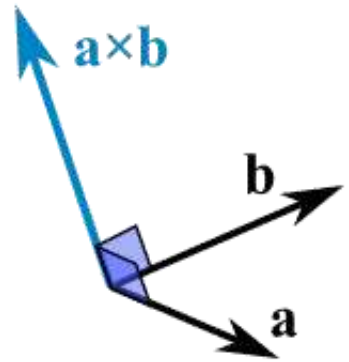
벡터 a 와 b 의 외적은 a × b 로 정의된다.

외적의 결과로 나온 벡터 c 는 벡터 a 와 b 의 수직인 벡터로 오른손 법칙의 방향

Vector로 결과가 나옴



Vector product
Cross product



$$A = |\mathbf{a} \times \mathbf{b}| = |\mathbf{a}||\mathbf{b}| \sin\theta.$$

Cross 함수 (vector 곱)

2차원 벡터는 스칼라 값으로 나옴 3차원 벡터이상 표시 됨

→ 2차원 백터간의 cross produc의 방향은 2차원으로 표현이 안되기 때문에 magnitude만 표시

```
import numpy as np

x = np.array([0, 0, 1])
y = np.array([0, 1, 0])

print(np.cross(x, y))
print(np.cross(y, x))
```

```
[-1  0  0]
[1 0 0]
```

```
import numpy as np

A = np.array([1, 2])
B = np.array([4, 4])

print(np.cross(A, B))

A = np.array([1, 0, 0])
B = np.array([0, 0, 1])

print(np.cross(A, B))
```

```
-4
[ 0 -1  0]
```

outer

Outer 함수 (외적)

두 벡터간의 텐서곱을 뜻함.
Cross-product와 다르게 vector가 아닌 행렬을 산출

$$\mathrm{u}\otimes\mathrm{v} = \mathrm{u}v^T = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}(v_1\ v_2\ v_3) = \text{matrix A} = \begin{pmatrix} u_1v_1 & u_1v_2 & u_1v_3 \\ u_2v_1 & u_2v_2 & u_2v_3 \\ u_3v_1 & u_3v_2 & u_3v_3 \end{pmatrix}$$

where

$$\mathrm{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \qquad \mathrm{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

Outer 함수 (외적)

Out는 두개의 벡터에 대한 행렬로 구성
out[i, j] = a[i] * b[j]
벡터의 값이 문자일 경우 문자 배수만 큼 처리

```
import numpy as np

a = np.array([[1, 2, 3]])
b = np.array([[4, 5]])

print(np.outer(a, b))

a = np.array([[1, 0]])
b = np.array([[4, 1]])

print(np.outer(a, b))
```

```
[[ 4  5]
 [ 8 10]
 [12 15]]
[[4 1]
 [0 0]]
```

```
import numpy as np

x = np.array(['a', 'b', 'c'], dtype = object)
print(np.outer(x, [1, 2, 3]))

y = np.array([1, 2, 3], dtype = object)
print(np.outer(y, ['a', 'b', 'c']))
```

```
[['a' 'aa' 'aaa']
 ['b' 'bb' 'bbb']
 ['c' 'cc' 'ccc']]
[['a' 'b' 'c']
 ['aa' 'bb' 'cc']
 ['aaa' 'bbb' 'ccc']]
```

연산 정리

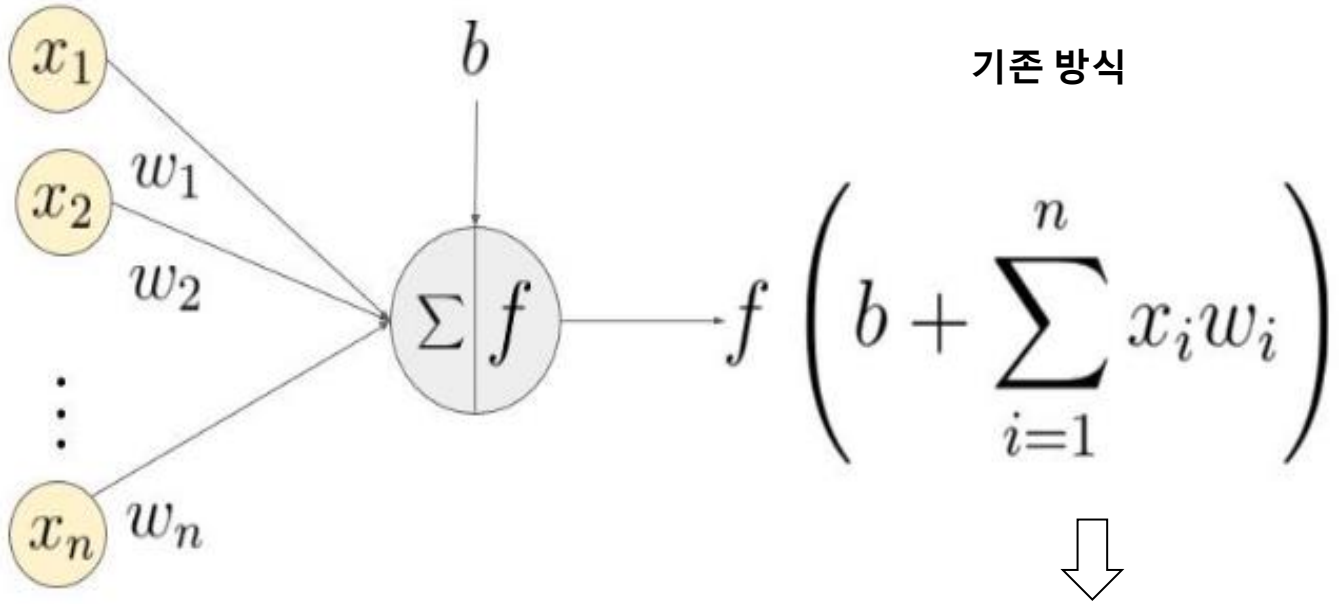
function	description
Cross	3차원 공간상에서 cross product를 산출 (2차원 vector는 scalar 산출)
Matmul	행렬 곱
dot	1차원 배열 : 내적, 2차원 행렬곱
Inner	내적
*	원소간 (element wise) 곱
Multiply	
outer	두 벡터의 외적 연산 (배열을 산출)

```
import numpy as np

A = np.array([[2, -3], [1, 1]])
B = np.array([1, 1])

print(np.cross(A, B))
print(np.matmul(A, B))
print(np.inner(A, B))
print(np.dot(A, B))
print(A * B)
print(np.multiply(A, B))
print(np.outer(A, B))
```

```
[5 0]
[-1  2]
[-1  2]
[-1  2]
[[ 2 -3]
 [ 1  1]]
[[ 2 -3]
 [ 1  1]]
[[ 2  2]
 [-3 -3]
 [ 1  1]
 [ 1  1]]
```

내적(dot)

배열 연산의 의미



기존 방식

Vector Space를 이해한후의 방식

$$\mathbf{Z} = \mathbf{X} \cdot \mathbf{W}$$

target **Input** **Weight**

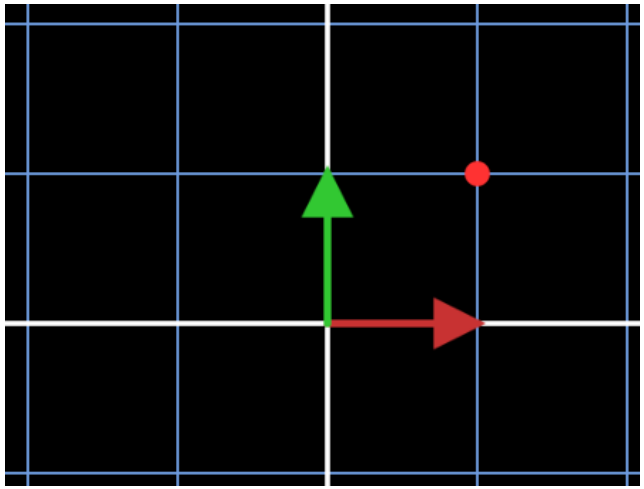
출처 : 인공지능신문(https://www.aitimes.kr)

배열 연산의 의미

배열의 곱은 선형 변환이다.

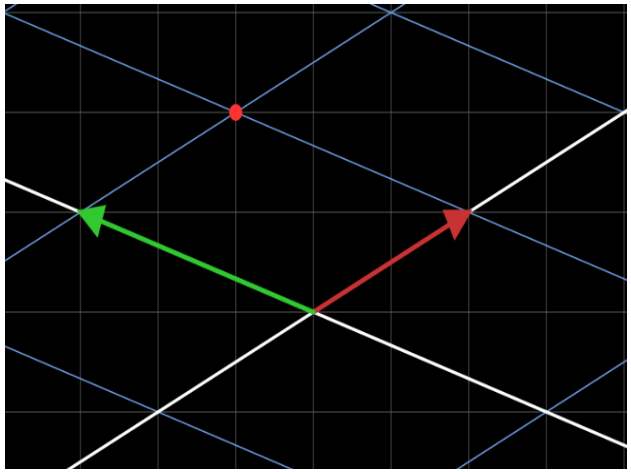
$$A = \begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix} \quad \vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$A\vec{x} = \begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$



선형 변환

$$\rightarrow \begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix} \rightarrow$$



대각행렬 : diag

정사각행렬 A = (aij)(i, j = 1, 2, 3,..., n)의 원소 aij가 aij =0(i≠j)을 만족시키는 행렬
A의 주대각선 위에 있는 원소(대각선원소) aij(i = j) 외의 원소 aij(i≠j)가 모두 0인 행렬

$$AB = BA = \begin{bmatrix} a_1b_1 & 0 & \cdots & 0 \\ 0 & a_2b_2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & a_nb_n \end{bmatrix}$$

(A,B는 각각 대각행렬)

```
import numpy as np

x = np.arange(9).reshape((3, 3))
print(x)
# 대각행렬의 요소를 추출
print(np.diag(x))
print(np.diag(x, k = 1))
print(np.diag(x, k = -1))
# 대각행렬을 다시 표현하기
print(np.diag(np.diag(x)))
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[0 4 8]
[1 5]
[3 7]
[[0 0 0]
 [0 4 0]
 [0 0 8]]
```

항등행렬: identity

모든 행렬과 dot 연산시 자기 자신이 나오게 하는 단위행렬

$$I_1 = [1],\ I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ \cdots,\ I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

```python
import numpy as np

# 항등행렬(identity matrix)
a = np.identity(2)
print(a)
b = np.array([[4, 1], [3, 2]])
print(np.dot(b, a))
print(np.dot(a, b))
```

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

삼각행렬 : tril/triu

상삼각 행렬(Upper triangular matrix)
과 하삼각 행렬(lower triangular matrix)을 총칭
하여 일컫는 말.

Upper triangular matrix

$$\begin{bmatrix} 1 & 4 & 100 \\ 0 & 3 & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

lower triangular matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 8 & 0 \\ 4 & 9 & 7 \end{bmatrix}$$

```
import numpy as np

print(np.tril([[1, 2, 3], [4, 5, 6], [7, 8, 9]], -1))
print(np.triu([[1, 2, 3], [4, 5, 6], [7, 8, 9]], 1))
```

```
[[0 0 0]
 [4 0 0]
 [7 8 0]]
[[0 2 3]
 [0 0 6]
 [0 0 0]]
```

행렬 전치 : T

전치: 행렬의 행과 열을 서로 바꾸는 것. 수학책에서는 위첨자 T로 행렬 A의 전치를 나타낸다.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

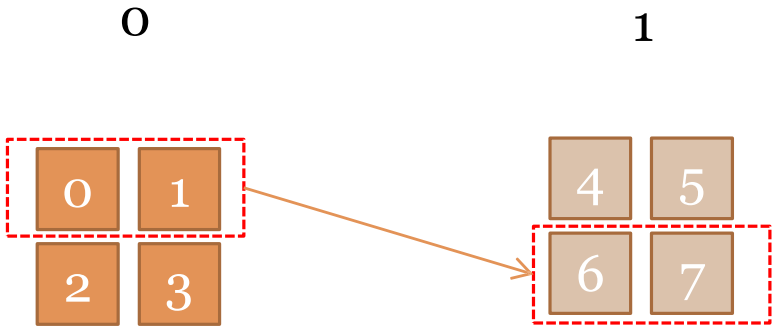
$$A^T = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

```
import math
import numpy as np

a = np.array([[1, 2], [3, 4]])
print(a.T)
print(np.transpose(a))
```

```
[[1 3]
 [2 4]]
[[1 3]
 [2 4]]
```

Trace : 3차원 행렬

3차원(2,2,2) 대각행렬의 합은 첫번째 차원의 1과 두번째의 마지막을 합산해서 출력




```
import numpy as np

a = np.array([[1, 0], [1, 0]])
b = np.array([[4, 1], [4, 1]])
print(np.trace(a))
print(np.trace(b))

a = np.arange(8).reshape((2, 2, 2))
print(a)
print(np.trace(a))
```

```
1
5
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
[6 8]
```

행렬식(det)

정방행렬에 하나의 수를 대응시킴으로써,
- 연립방정식의 해를 구하거나,
- 연립방정식 해의 존재성을 살피려고 할 때 쓰여짐

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Equation

$$\det A = \det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

If det A == 0, 역행렬이 존재 하지 않음

```
import math
import numpy as np

a = np.array([[3, 1], [2, 2]])
print(np.linalg.det(a))
```

```
4.000000000000001
```

numpy.linalg

다양한 선형대수 function을 정의하고 있는 numpy sub class.

Matrix eigenvalues

`linalg.eig`(a)	Compute the eigenvalues and right eigenvectors of a square array.
`linalg.eigh`(a[, UPLO])	Return the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix.
`linalg.eigvals`(a)	Compute the eigenvalues of a general matrix.
`linalg.eigvalsh`(a[, UPLO])	Compute the eigenvalues of a complex Hermitian or real symmetric matrix.

Norms and other numbers

`linalg.norm`(x[, ord, axis, keepdims])	Matrix or vector norm.
`linalg.cond`(x[, p])	Compute the condition number of a matrix.
`linalg.det`(a)	Compute the determinant of an array.

출처 : https://numpy.org/doc/stable/reference/routines.linalg.html

역행렬(inv)

역행렬은 수반행렬에 행렬식으로 나눗값이 됨
연립방정식의 해를 찾을때 사용

Equation

$$A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

2x+3y=4

5x+6y=5

$$\begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

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

```
import numpy as np

A = np.array([[2, 3], [5, 6]])
B = np.array([4, 5])
C = np.linalg.inv(A)
D = np.dot(C, B)
print(D)
```

```
[-3.          3.33333333]
```

역행렬(inv) – 2차원

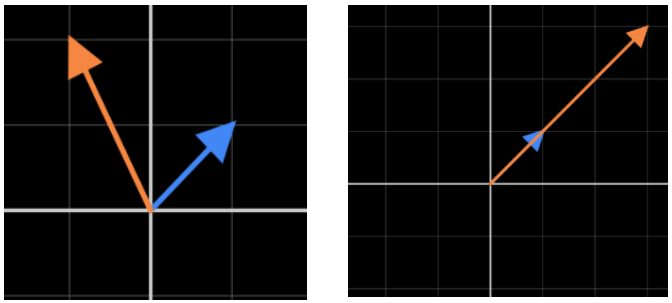
고유값 (Eigenvalue), 고유벡터 (Eigenvector):

아래와 같은 수식으로 임의의 nxm 행렬을 분해하는 방법 (PCA같은 차원압축에 사용)

$$A\vec{x} = \lambda\vec{x}$$

λ : eigen value

x : eigen vector



```
import numpy as np

e = np.array([[4, 2], [3, 5]])
print(e)

w, v = np.linalg.eig(e)
# w: the eigenvalues lambda
print(w)
# v: the corresponding eigenvectors, one eigenvector per column
print(v)

# eigenvector of eigenvalue lambda 2
print(v[:, 0])

# eigenvector of eigenvalue lambda 7
print(v[:, 1])
```

```
[[4 2]
 [3 5]]
[2. 7.]
[[-0.70710678 -0.5547002 ]
 [ 0.70710678 -0.83205029]]
[-0.70710678  0.70710678]
[-0.5547002  -0.83205029]
```

Singular Value Decomposition

아래와 같은 수식으로 임의의 nxm 행렬을 분해하는 방법

$$A = U\Sigma V^T$$

$$= \begin{pmatrix} | & | & & | \\ \vec{u}_1 & \vec{u}_2 & \cdots & \vec{u}_m \\ | & | & & | \end{pmatrix} \begin{pmatrix} \sigma_1 & & & 0 \\ & \sigma_2 & & 0 \\ & & \ddots & 0 \\ & & & \sigma_m \; 0 \end{pmatrix} \begin{pmatrix} - & \vec{v}_1^T & - \\ - & \vec{v}_2^T & - \\ & \vdots & \\ - & \vec{v}_n^T & - \end{pmatrix}$$

$$= \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \cdots + \sigma_m \vec{u}_m \vec{v}_m^T$$

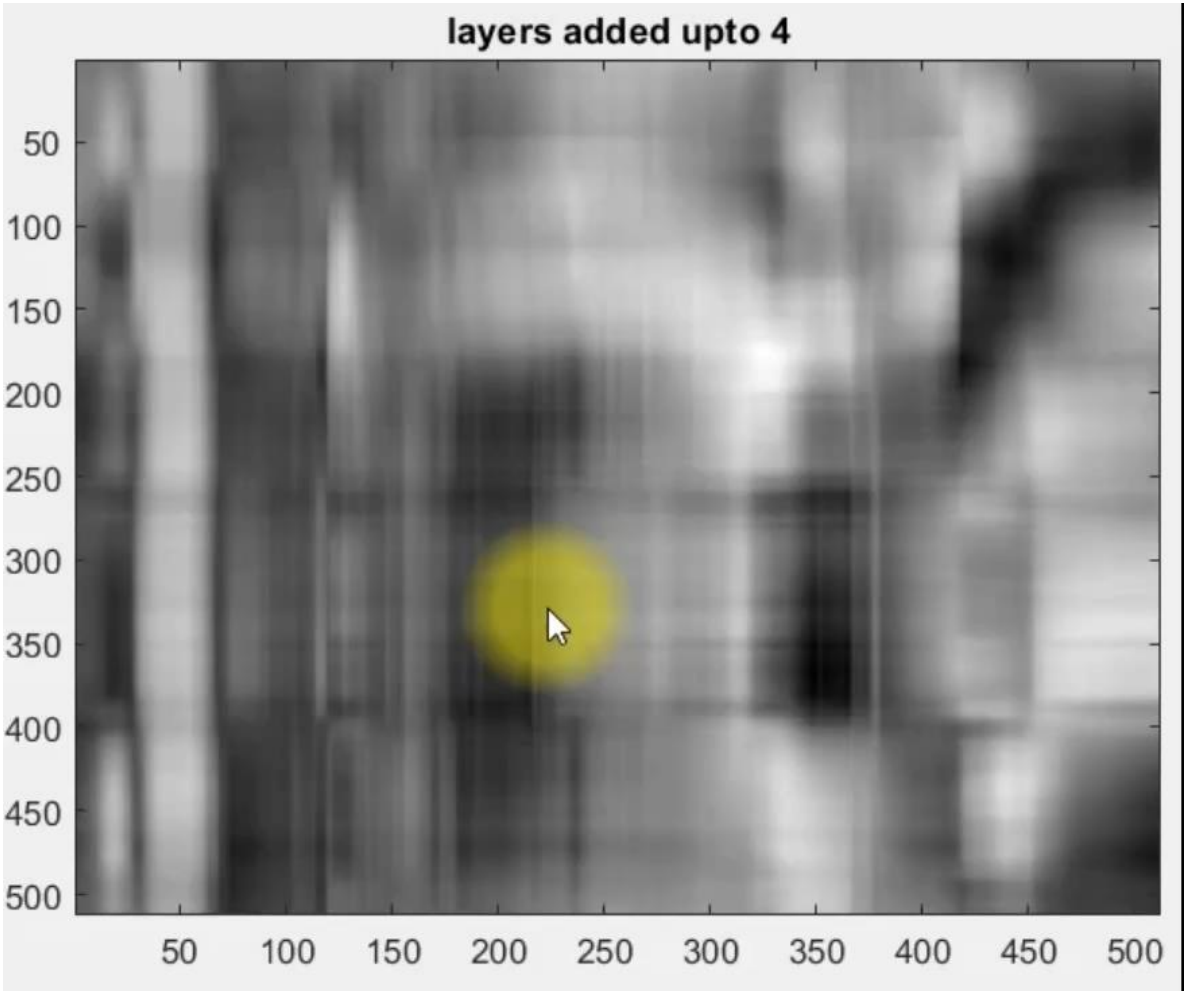
where
A: m×n rectangular matrix
U: m×m orthogonal matrix
ΣΣ: m×n diagonal matrix
V: n×n orthogonal matrix

```
import numpy as np

A = np.array([[3, 6], [2, 3], [0, 0], [0, 0]])
print(A)
u, s, vh = np.linalg.svd(A)
print(u)
print(s)
print(vh)
```

```
[[3 6]
 [2 3]
 [0 0]
 [0 0]]
[[-0.8816746  -0.47185793  0.          0.        ]
 [-0.47185793  0.8816746   0.          0.        ]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
[7.60555128 0.39444872]
[[-0.47185793 -0.8816746 ]
 [ 0.8816746  -0.47185793]]
```

역행렬(inv) – 2차원

Singular Value Decomposition

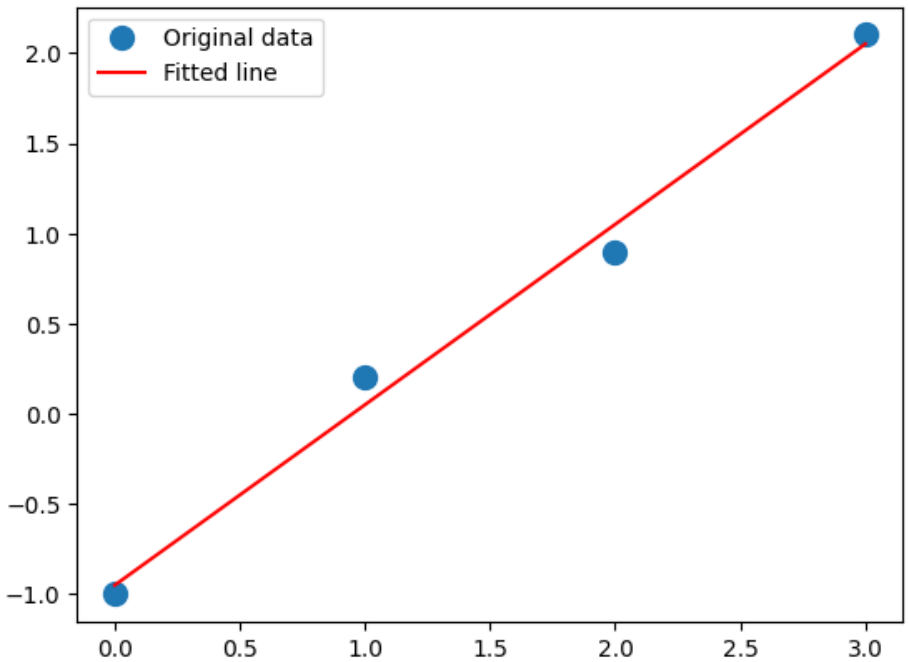


Least-squares solution

최소자승법(Least-squares method)으로 잔차 제곱합을 최소화하는 회귀계수를 추정

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([0, 1, 2, 3])
y = np.array([-1, 0.2, 0.9, 2.1])
A = np.vstack([x, np.ones(len(x))]).T
print(A)
m, c = np.linalg.lstsq(A, y, rcond = None)[0]
print(m, c)

plt.plot(x, y, 'o', label = 'Original data', markersize = 10)
plt.plot(x, m*x+c, 'r', label = 'Fitted line')
plt.legend()
plt.show()
```

```
[[0. 1.]
 [1. 1.]
 [2. 1.]
 [3. 1.]]
0.9999999999999999 -0.9499999999999997
```



강의 요약

- **조건절 여러 개 쓰기 연습**
  - 두 가지 이상 조건절을 한번에 사용
  - and, or, boolean형 활용

- **조건문 오류 찾기**
  - 조건문 문법 이해하고 사용하기

연습문제

1. **아래와 같은 x와 y의 관계를 가지는 데이터가 주어졌을때 y=ax+b형태의 최적 선형 모델을 찾아보세요.**

```
x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([0, 0.8, 1.9, 3.1, 3.9, 5.1])
```

연습문제 코드

```
import numpy as np

# 1번 문제
# 주어진 x, y 데이터
x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([0, 0.8, 1.9, 3.1, 3.9, 5.1])

# X 행렬과 Y 벡터 생성
X = np.vstack([x, np.ones(len(x))]).T
Y = y
# 최소 제곱 문제 해결
a, _, _, _ = np.linalg.lstsq(X, Y, rcond=None)

print("a (slope):", a[0])
print("b (intercept):", a[1])
```

```
a (slope): 1.0285714285714285
b (intercept): -0.10476190476190406
```

강의 요약

- **Numpy 기초**
  - 구조, 생성, 연산, 인덱싱 익히기

- **선형대수 개념**
  - 차원, 데이터 구조, 벡터, 배열 개념 확보
  - 데이터를 바라보는 관점 변화

Quiz

- 주식 데이터 분석
- 문제: 당신은 5개의 다른 주식에 대한 1년간의 일일 종가 데이터를 가지고 있습니다. 이 데이터는 (252, 5) 형태의 2D NumPy 배열로 주어져 있습니다 (252은 대략적인 1년의 거래일 수입니다). 각 열은 다른 주식의 가격을 나타냅니다.
- 다음 작업을 수행하세요:
  1. 임의의 (252,5) 형태의 2d NumPy array를 생성하세요.
  2. 각 주식의 일별 수익률을 계산하세요. 일별 수익률은 다음과 같이 계산됩니다:
  3. (오늘의 가격 - 어제의 가격) / 어제의 가격
  4. 각 주식의 평균 일별 수익률을 계산하세요.
  5. 가장 높은 평균 일별 수익률을 가진 주식을 찾으세요.
  6. 각 주식의 수익률의 표준 편차를 계산하세요. 이 값은 주식의 위험성을 나타냅니다.
  7. 가장 위험성이 높은 주식을 찾으세요.

연습문제 코드

```
import numpy as np

# 임의로 주식 데이터 생성
np.random.seed(0)
prices = np.random.rand(252, 5)

# 일별 수익률 계산
returns = (prices[1:] - prices[:-1]) / prices[:-1]

# 평균 일별 수익률 계산
mean_returns = np.mean(returns, axis = 0)

# 가장 높은 평균 일별 수익률을 가진 주식 찾기
best_stock = np.argmax(mean_returns)
print(f'Best stock (highest mean daily return): {best_stock}')

# 수익률의 표준편차 계산
risks = np.std(returns, axis = 0)

# 가장 위험성이 높은 주식 찾기
riskiest_stock = np.argmax(risks)
print(f'Riskiest stock (highest standard deviation of daily return): {riskiest_stock}')
```

```
Best stock (highest mean daily return): 2
Riskiest stock (highest standard deviation of daily return): 2
```

감사합니다

[1차시]