영상처리프로그래밍 : NumPy (1)

한림대학교 소프트웨어융합대학
박섭형

2022년 1학기

NumPy 개요

- OpenCV나 Matplotlib의 image 모듈 등은 NumPy의 다차원 배열인 ndarray 객체를 이용해서 영상 데이터를 표현
- 영상 처리에서 NnumPy는 매우 중요한 역할을 담당
- NumPy Homepage
- 데이터 구조
  - ndarray 클래스 : array 클래스라고 부르기도 함
  - 동일한 데이터형 원소들(주로 숫자형 데이터)의 다차원 배열
  - indexing: 정수들의 tuple로 인덱스를 표현
  - broadcasting
- 다양한 수치 연산 도구
  - 다차원 배열에 적용할 수 있는 벡터화 함수
  - 선형 대수, 푸리에 변환, 콘볼루션, 통계 라이브러리 등

NumPy의 장점

- 다차원 배열 또는 행렬 연산의 편리함
- 우수한 메모리 효율
- 빠른 처리 속도

NumPy 다차원 배열 클래스 ndarray의 주요 attributes

- ndarray.ndim: axes (dimensions)의 수
- ndarray.shape: axes별 원소의 개수를 튜플로 표현
- ndarray.size: 원소의 개수

- ndarray.dtype: numpy.int32, numpy.int16, numpy.float64 등
- ndarray.itemsize: 각 원소의 크기를 byte 단위
- ndarray.nbytes: 전체 데이터의 크기를 byte 단위로 표시

[2]:
```
import numpy as np
np.__version__
```

[2]:
```
'1.20.3'
```

[3]:
```
na = np.random.randint(1, 10, (3, 4))
na
```

[3]:
```
array([[6, 7, 2, 7],
       [8, 3, 9, 7],
       [2, 6, 7, 5]])
```

[4]:
```
na.ndim
```

[4]:
```
2
```

[5]:
```
na.shape
```

[5]:
```
(3, 4)
```

[6]:
```
na.dtype
```

[6]:
```
dtype('int32')
```

[7]:
```
na.size
```

[7]:
```
12
```

[8]:
```
na.itemsize
```

[8]:
```
4
```

[9]:
```
na.nbytes
```

[9]:
```
48
```

NumPy ndarray 객체의 데이터 형

dtype	문자 코드	설명	호환되는 C의 data type
ndarray.int8	‘b’ or ‘i1’	8-bit signed integer	char
ndarray.int16	‘h’ or ‘i2’	16-bit signed integer	short
ndarray.int32	‘i’ or ‘i4’	32-bit signed integer	int
ndarray.int64	‘q’ or ‘i8’	64-bit signed integer	long
ndarray.uint8	‘B’ or ‘u1’	8-bit unsigned integer	unsigned char
ndarray.uint16	‘H’ or ‘u2’	16-bit unsigned integer	unsigned short
ndarray.uint32	‘I’ or ‘u4’	32-bit unsigned integer	unsigned int
ndarray.uint64	‘Q’ or ’u8	64-bit unsigned integer	unsigned long

정수 데이터 형

[10]:
```
int_dt = [np.int8, np.uint8,
      np.int16, np.uint16,
      np.int32, np.uint32,
      np.int64, np.uint64, 'i1', 'i2', 'i4', 'i8', 'u1', 'u2', 'u4', 'u8']

for dt in int_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
int8 1
uint8 1
int16 2
uint16 2
int32 4
uint32 4
int64 8
uint64 8
int8 1
int16 2
int32 4
int64 8
uint8 1
uint16 2
```

```
uint32 4
uint64 8
```

[11]:
```
int_dt = ['<b', '<B',
          '<h', '<H',
          '<i', '<I',
          '<q', '<Q']

for dt in int_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
int8 1
uint8 1
int16 2
uint16 2
int32 4
uint32 4
int64 8
uint64 8
```

dtype	문자 코드	설명 (sign, exponent, mantissa)	호환되는 C의 data type
ndarray.float16	'e' or 'f2'	16-bit floating point (1, 5, 10)	
ndarray.float32	'f' or 'f4'	32-bit floating point (1, 8, 23)	float
ndarray.float64	'd' or 'f8'	64-bit floating point (1, 11, 52)	double

실수 데이터 형

[12]:
```
float_dt = [np.float16, np.float32, np.float64, 'f2', 'f4', 'f8']

for dt in float_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
float16 2
float32 4
```

```
float64 8
float16 2
float32 4
float64 8
```

[13]:
```
float_dt = ['<e', '<f', '<d']

for dt in float_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
float16 2
float32 4
float64 8
```

dtype	문자 코드	설명
ndarray.complex64	'F' or 'c8'	32-bit floating point x 2
ndarray.complex128	'G' or 'c16'	64-bit floating point x 2

복소수 데이터 형

[14]:
```
complex_dt = [np.csingle, np.cdouble]

for dt in complex_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
complex64 8
complex128 16
```

[15]:
```
complex_dt = ['<F', '<G']

for dt in complex_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
complex64 8
complex128 16
```

NumPy 정수형의 overflow와 underflow

[16]:
```
import numpy as np
a = np.array([100, 200], np.int8)
a
```

[16]:
```
array([100, -56], dtype=int8)
```

[17]:
```
np.iinfo(np.int8)
```

[17]:
```
iinfo(min=-128, max=127, dtype=int8)
```

[18]:
```
a + a
```

[18]:
```
array([ -56, -112], dtype=int8)
```

[19]:
```
3 * a
```

[19]:
```
array([44, 88], dtype=int8)
```

[20]:
```
b = np.array([100, 200], np.uint8)
b
```

[20]:
```
array([100, 200], dtype=uint8)
```

[21]:
```
np.iinfo(np.uint8)
```

[21]:
```
iinfo(min=0, max=255, dtype=uint8)
```

0.1 Ndarray 생성

리스트와 튜플을 이용한 ndarray 생성

[22]:
```
data1 = (1, 2, 3, 4, 5) # tuple의 모든 원소의 데이터 유형이 같아야 한다
arr1 = np.array(data1) # 1d array
```

[23]:
```
arr1
```

[23]:
```
array([1, 2, 3, 4, 5])
```

[24]:
```
arr1.dtype
```

[24]:
```
dtype('int32')
```

[25]:
```
arr1.ndim
```

[25]:
```
1
```

[26]:
```
arr1.size
```

[26]:
```
5
```

[27]:
```
data2 = [[1, 2, 3., 4], [5, 6, 7, 8.]] # nested list
arr2 = np.array(data2) # 2d array
```

[28]:
```
data2
```

[28]:
```
[[1, 2, 3.0, 4], [5, 6, 7, 8.0]]
```

[29]:
```
arr2
```

[29]:
```
array([[1., 2., 3., 4.],
       [5., 6., 7., 8.]])
```

[30]:
```
arr2.dtype
```

[30]:
```
dtype('float64')
```

[31]:
```
arr2.ndim
```

[31]:
```
2
```

[32]:
```
arr2.size
```

[32]:
```
8
```

[33]:
```
data3 = [[1, 2, 3., 4], [5+1j, 6, 7, 8.]] # nested list
arr3 = np.array(data3) # 2d array
```

[34]:
```
data3
```

[34]:
```
[[1, 2, 3.0, 4], [(5+1j), 6, 7, 8.0]]
```

[35]:
```
arr3
```

[35]:
```
array([[1.+0.j, 2.+0.j, 3.+0.j, 4.+0.j],
       [5.+1.j, 6.+0.j, 7.+0.j, 8.+0.j]])
```

[36]:
```
arr3.dtype
```

[36]:
```
dtype('complex128')
```

[37]:
```
arr3.ndim
```

[37]:
```
2
```

[38]:
```
arr3.size
```

[38]:
```
8
```

[39]:
```
data4 = ['a', 'b']
arr4 = np.array(data4)
```

[40]:
```
arr4.dtype
```

[40]:
```
dtype('<U1')
```

[41]:
```
arr4.itemsize
```

[41]:
```
4
```

[42]:
```
data5 = ['ab', 'b']
arr5 = np.array(data5)
```

[43]:
```
arr5.dtype
```

[43]:
```
dtype('<U2')
```

[44]:
```
arr5.itemsize
```

[44]:
```
8
```

[45]:
```
data6 = ['abc', 'b']
arr6 = np.array(data6)
```

[46]:
```
arr6.dtype
```

[46]:
```
dtype('<U3')
```

[47]:
```
arr6.itemsize
```

[47]:
```
12
```

[48]:
```
from sys import getsizeof
getsizeof(np.array([])), getsizeof(arr6)
```

[48]:
```
(104, 128)
```

[49]:
```
for n in range(0, 53):
    data_s = 'abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz'[:n]
    arr_s = np.array(data_s)
    print(arr_s, arr_s.dtype, arr_s.itemsize, getsizeof(arr_s))
```

```
 <U1 4 92
a <U1 4 92
ab <U2 8 96
abc <U3 12 100
abcd <U4 16 104
abcde <U5 20 108
abcdef <U6 24 112
abcdefg <U7 28 116
abcdefgh <U8 32 120
abcdefghi <U9 36 124
abcdefghij <U10 40 128
abcdefghijk <U11 44 132
abcdefghijkl <U12 48 136
abcdefghijklm <U13 52 140
abcdefghijklmn <U14 56 144
abcdefghijklmno <U15 60 148
abcdefghijklmnop <U16 64 152
abcdefghijklmnopq <U17 68 156
abcdefghijklmnopqr <U18 72 160
abcdefghijklmnopqrs <U19 76 164
abcdefghijklmnopqrst <U20 80 168
abcdefghijklmnopqrstu <U21 84 172
abcdefghijklmnopqrstuv <U22 88 176
abcdefghijklmnopqrstuvw <U23 92 180
abcdefghijklmnopqrstuvwx <U24 96 184
abcdefghijklmnopqrstuvwxy <U25 100 188
abcdefghijklmnopqrstuvwxyz <U26 104 192
```

```
abcdefghijklmnopqrstuvwxyza <U27 108 196
abcdefghijklmnopqrstuvwxyzab <U28 112 200
abcdefghijklmnopqrstuvwxyzabc <U29 116 204
abcdefghijklmnopqrstuvwxyzabcd <U30 120 208
abcdefghijklmnopqrstuvwxyzabcde <U31 124 212
abcdefghijklmnopqrstuvwxyzabcdef <U32 128 216
abcdefghijklmnopqrstuvwxyzabcdefg <U33 132 220
abcdefghijklmnopqrstuvwxyzabcdefgh <U34 136 224
abcdefghijklmnopqrstuvwxyzabcdefghi <U35 140 228
abcdefghijklmnopqrstuvwxyzabcdefghij <U36 144 232
abcdefghijklmnopqrstuvwxyzabcdefghijk <U37 148 236
abcdefghijklmnopqrstuvwxyzabcdefghijkl <U38 152 240
abcdefghijklmnopqrstuvwxyzabcdefghijklm <U39 156 244
abcdefghijklmnopqrstuvwxyzabcdefghijklmn <U40 160 248
abcdefghijklmnopqrstuvwxyzabcdefghijklmno <U41 164 252
abcdefghijklmnopqrstuvwxyzabcdefghijklmnop <U42 168 256
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopq <U43 172 260
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqr <U44 176 264
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrs <U45 180 268
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrst <U46 184 272
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstu <U47 188 276
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuv <U48 192 280
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvw <U49 196 284
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwx <U50 200 288
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxy <U51 204 292
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz <U52 208 296
```

Ndarray의 Axis

- Ndarray에 연산을 적용하면 내부에서 iteration 과정 발생
- Axis: numpy 연산에서 iteration이 진행되는 방향

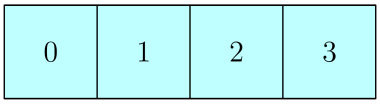
1차원 배열

[50]:
```
import numpy as np
x = np.array([0, 1, 2, 3])
x
```

[50]:
```
array([0, 1, 2, 3])
```

[51]:
```
x.ndim
```

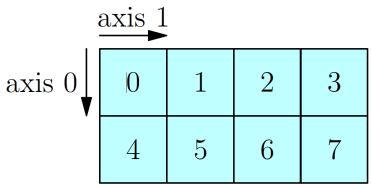
[51]:
```
1
```



2차원 배열

[52]:
```
x = np.array([[0, 1, 2, 3], [4, 5, 6, 7]])
print(x.ndim)
```

```
2
```



[53]:
```
print(x.sum())
print(x.sum(axis=0))
print(x.sum(axis=1))
```

```
28
[ 4  6  8 10]
[ 6 22]
```

[54]:
```
print(x.sum(axis=0, keepdims=True))
print(x.sum(axis=1, keepdims=True))
```

```
[[ 4  6  8 10]]
[[ 6]
 [22]]
```

3차원 배열

[55]:
```
x = np.arange(24).reshape(3,2,4)
x
```

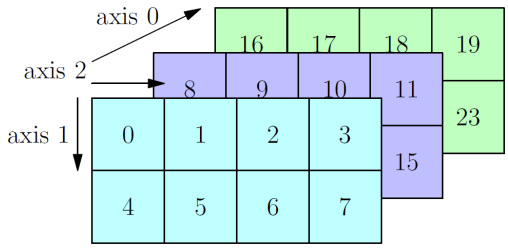
[55]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]],

       [[16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

[56]:
```
np.arange(24)
```

[56]:
```
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])
```



[57]:
```
x.sum(0)
```

[57]:
```
array([[24, 27, 30, 33],
       [36, 39, 42, 45]])
```

[58]:
```
x.sum(1)
```

[58]:
```
array([[ 4,  6,  8, 10],
       [20, 22, 24, 26],
       [36, 38, 40, 42]])
```

[59]:
```
x.sum(2)
```

[59]:
```
array([[ 6, 22],
       [38, 54],
       [70, 86]])
```

- x의 shape이 (3,2,4) 인 경우
  - sum(0) 의 shape: (2,4)
  - sum(1) 의 shape: (3,4)
  - sum(2) 의 shape: (3,2)

[60]:
```
x[2,0,1]
```

[60]:
```
17
```

특별한 ndarray를 생성하는 내장 함수들

numpy.arange( )

```
numpy.arange(start, stop[, step])
```

[61]:
```
n = np.arange(5)
n
```

[61]:
```
array([0, 1, 2, 3, 4])
```

[62]:
```
n.dtype
```

[62]:
```
dtype('int32')
```

[63]:
```
n = np.arange(5, 0, -1)
n
```

[63]:
```
array([5, 4, 3, 2, 1])
```

[64]:
```
x = np.arange(0, 1, 0.1)
x
```

[64]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

[65]:
```
x.dtype
```

[65]:
```
dtype('float64')
```

numpy.linspace( )

```
np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)
```

- [start, stop] 구간에서 균등하게 나누어진 num 개의 숫자들을 ndarray 객체로 반환

[66]:
```
x = np.linspace(1, 2, 5)
x
```

[66]:
```
array([1.  , 1.25, 1.5 , 1.75, 2.  ])
```

[67]:
```
x = np.linspace(1, 2, 5, endpoint=False)
x
```

[67]:
```
array([1. , 1.2, 1.4, 1.6, 1.8])
```

numpy.logspace( )

`np.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)`
- `[base**start, base**stop]` 구간에서 `log` 스케일로 균등하게 나누어진 `num` 개의 숫자들을 `ndarray` 객체

[68]:
```
np.logspace(1, 5, 5)
```

[68]:
```
array([1.e+01, 1.e+02, 1.e+03, 1.e+04, 1.e+05])
```

[69]:
```
np.logspace(0, 2, 3)
```

[69]:
```
array([  1.,  10., 100.])
```

np.zeros( )

`np.zeros(shape, dtype=None, order='C')`

- 모든 원소가 0.인 ndarray 객체를 반환
- default type: np.float64

[70]:
```
x = np.zeros(5)
x
```

[70]:
```
array([0., 0., 0., 0., 0.])
```

[71]:
```
x.dtype
```

[71]:
```
dtype('float64')
```

[72]:
```
x = np.zeros(5, dtype=np.int32)
x
```

[72]:
```
array([0, 0, 0, 0, 0])
```

[73]:
```
x.dtype
```

[73]:
```
dtype('int32')
```

[74]:
```
np.zeros((3, 4))
```

[74]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[75]:
```
np.zeros((3, 4, 2))
```

[75]:
```
array([[[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]],

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

       [[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]]])
```

np.zeros_like( )

```
np.zeros_like(a, dtype=None, order='K', subok=True, shape=None)
```

- a와 shape과 dtype이 같은 모든 원소가 0.인 ndarray 객체를 반환

[76]:
```
x = np.arange(5)
x, x.dtype, x.shape
```

[76]:
```
(array([0, 1, 2, 3, 4]), dtype('int32'), (5,))
```

[77]:
```
y = np.linspace(0, 1, 2)
y, y.dtype, y.shape
```

[77]:
```
(array([0., 1.]), dtype('float64'), (2,))
```

[78]:
```
a = np.zeros_like(x)
a, a.dtype, a.shape
```

[78]:
```
(array([0, 0, 0, 0, 0]), dtype('int32'), (5,))
```

[79]:
```
b = np.zeros_like(y)
b, b.dtype, b.shape
```

[79]:
```
(array([0., 0.]), dtype('float64'), (2,))
```

np.ones( )

```
np.ones(shape, dtype=None, order='C')
```

- 모든 원소가 1.인 ndarray 객체를 반환
- default type: np.float64

[80]:
```
x = np.ones(5)
x
```

[80]:
```
array([1., 1., 1., 1., 1.])
```

[81]:
```
x.dtype
```

[81]:
```
dtype('float64')
```

[82]:
```
x = np.ones(5, dtype=np.int32)
x
```

[82]:
```
array([1, 1, 1, 1, 1])
```

[83]:
```
x.dtype
```

[83]:
```
dtype('int32')
```

[84]:
```
np.ones((3, 4))
```

[84]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[85]:
```
np.ones((3, 4, 2))
```

[85]:
```
array([[[1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.]]])
```

np.ones_like( )

`np.ones_like(a, dtype=None, order='K', subok=True, shape=None)`

- a와 shape과 dtype이 같은 모든 원소가 0. 인 ndarray 객체를 반환

[86]:
```
x = np.arange(5)
x, x.dtype, x.shape
```

[86]:
```
(array([0, 1, 2, 3, 4]), dtype('int32'), (5,))
```

[87]:
```
y = np.linspace(0, 1, 2)
y, y.dtype, y.shape
```

[87]:
```
(array([0., 1.]), dtype('float64'), (2,))
```

[88]:
```
a = np.ones_like(x)
a, a.dtype, a.shape
```

[88]:
```
(array([1, 1, 1, 1, 1]), dtype('int32'), (5,))
```

[89]:
```
b = np.ones_like(y)
b, b.dtype, b.shape
```

[89]:
```
(array([1., 1.]), dtype('float64'), (2,))
```

np.full( )

```
np.full(shape, fill_value, dtype=None, order='C')
```

- 모든 원소가 fill_value인 ndarray 객체를 반환

[90]:
```
x = np.full(5, 3)
x
```

[90]:
```
array([3, 3, 3, 3, 3])
```

[91]:
```
x.dtype
```

[91]:
```
dtype('int32')
```

[92]:
```
x = np.full(5, 3.)
x
```

[92]:
```
array([3., 3., 3., 3., 3.])
```

[93]:
```
x.dtype
```

[93]:
```
dtype('float64')
```

[94]:
```
x = np.full((3, 4), 3.)
x
```

[94]:
```
array([[3., 3., 3., 3.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.]])
```

[95]:
```
x = np.full((3, 4, 2), 3+2j)
x
```

[95]:
```
array([[[3.+2.j, 3.+2.j],
        [3.+2.j, 3.+2.j],
        [3.+2.j, 3.+2.j],
        [3.+2.j, 3.+2.j]],

       [[3.+2.j, 3.+2.j],
        [3.+2.j, 3.+2.j],
        [3.+2.j, 3.+2.j],
        [3.+2.j, 3.+2.j]],
```

```
       [[3.+2.j, 3.+2.j],
        [3.+2.j, 3.+2.j],
        [3.+2.j, 3.+2.j],
        [3.+2.j, 3.+2.j]]])
```

[96]:
```
x.dtype
```

[96]:
```
dtype('complex128')
```

np.full_like( )

`np.full_like(a, fill_value, dtype=None, order='K', subok=True, shape=None)`

- a와 shape과 dtype이 같고 모든 원소가 fill_value인 ndarray 객체를 반환

[97]:
```
x = np.arange(5)
x, x.dtype, x.shape
```

[97]:
```
(array([0, 1, 2, 3, 4]), dtype('int32'), (5,))
```

[98]:
```
y = np.ones((2, 3))
y, y.dtype, y.shape
```

[98]:
```
(array([[1., 1., 1.],
        [1., 1., 1.]]),
 dtype('float64'),
 (2, 3))
```

[99]:
```
a = np.full_like(x, 3.)
a, a.dtype, a.shape
```

[99]:
```
(array([3, 3, 3, 3, 3]), dtype('int32'), (5,))
```

[100]:
```
b = np.full_like(y, 3-2j)
b, b.dtype, b.shape
```

```
<__array_function__ internals>:5: ComplexWarning: Casting complex values to real
discards the imaginary part
```

[100]:
```
(array([[3., 3., 3.],
        [3., 3., 3.]]),
 dtype('float64'),
 (2, 3))
```

np.eye( )

`np.eye(N, M=None, k=0, dtype=<class 'float'>, order='C')`

- 대각선이 모두 1이고, shape이 (N,M) 인 2 차원 ndarray 객체를 반환
- k는 대각선의 인덱스

[101]:
```
x = np.eye(3)
x
```

[101]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[102]:
```
x.dtype
```

[102]:
```
dtype('float64')
```

[103]:
```
x = np.eye(3, 4)
x
```

[103]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]])
```

[104]:
```
x.dtype
```

[104]:
```
dtype('float64')
```

[105]:
```
np.eye(3, 2)
```

[105]:
```
array([[1., 0.],
       [0., 1.],
       [0., 0.]])
```

[106]:
```
np.eye(4)
```

[106]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

[107]:
```
np.eye(4, k=1)
```

[107]:
```
array([[0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.]])
```

[108]:
```
np.eye(4, k=2)
```

[108]:
```
array([[0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[109]:
```
np.eye(4, k=3)
```

[109]:
```
array([[0., 0., 0., 1.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[110]:
```
np.eye(4, k=4)
```

[110]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[111]:
```
np.eye(4, k=-1)
```

[111]:
```
array([[0., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]])
```

[112]:
```
np.eye(4, k=-2)
```

[112]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.]])
```

np.diag( )

`np.diag(v, k=0)`

- 2차원 ndarray 객체에서 대각선을 추출하여 1차원 ndarray 객체로 반환
- 1차원 ndarray 객체 v를 대각선 원소로 갖는 2차원 ndarray 객체를 반환
- k는 대각선의 위치를 결정하는 인덱스

[113]:
```
x = np.arange(16).reshape((4,4))
x
```

[113]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

[114]:
```
np.diag(x)
```

[114]:
```
array([ 0,  5, 10, 15])
```

[115]:
```
np.diag(x, k=1)
```

[115]:
```
array([ 1,  6, 11])
```

[116]:
```
np.diag(x, k=2)
```

[116]:
```
array([2, 7])
```

[117]:
```
np.diag(x, k=3)
```

[117]:
```
array([3])
```

[118]:
```
np.diag(x, k=4)
```

[118]:
```
array([], dtype=int32)
```

[119]:
```
x = np.arange(12).reshape((3,4))
x
```

[119]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[120]:
```
np.diag(x)
```

[120]:
```
array([ 0,  5, 10])
```

[121]:
```
np.diag([1,2,3])
```

[121]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

[122]:
```
np.diag([1,2,3], 1)
```

[122]:
```
array([[0, 1, 0, 0],
       [0, 0, 2, 0],
       [0, 0, 0, 3],
       [0, 0, 0, 0]])
```

[123]:
```
np.diag([1,2,3], -2)
```

[123]:
```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0]])
```

[124]:
```
np.diag([1,2,3])[:, ::-1]
```

[124]:
```
array([[0, 0, 1],
       [0, 2, 0],
       [3, 0, 0]])
```

[125]:
```
np.diag([1,2,3])[::-1, :]
```

[125]:
```
array([[0, 0, 3],
       [0, 2, 0],
       [1, 0, 0]])
```

[126]:
```
np.diag([1,2,3])[::-1, ::-1]
```

[126]:
```
array([[3, 0, 0],
       [0, 2, 0],
       [0, 0, 1]])
```