Python 과학 프로그래밍 기초

7. NumPy (2)

한림대학교 소프트웨어융합대학
박섭형

2021년 1학기

Ndarray의 Axis

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

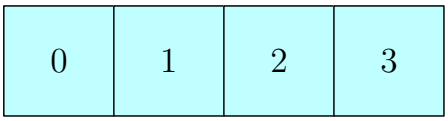
1차원 배열

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

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

[2]:
```
x.ndim
```

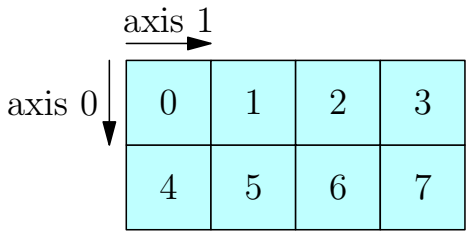
[2]:
```
1
```



2차원 배열

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

```
2
```



[4]:
```
print(x.sum())
print(x.sum(axis=0))
print(x.sum(axis=1))
```

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

[5]:
```
print(x.sum(axis=0, keepdims=True))
print(x.sum(axis=1, keepdims=True))
```

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

3차원 배열

[6]:
```
x = np.arange(24).reshape(3,2,4)
x
```

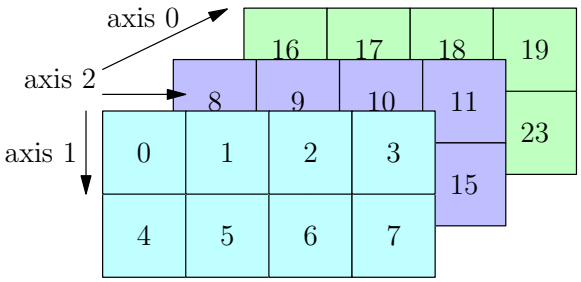
[6]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

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

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

[7]:
```
np.arange(24)
```

[7]:
```
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])
```



[8]:
```
x.sum(0)
```

[8]:
```
array([[24, 27, 30, 33],
       [36, 39, 42, 45]])
```

[9]:
```
x.sum(1)
```

[9]:
```
array([[ 4,  6,  8, 10],
       [20, 22, 24, 26],
       [36, 38, 40, 42]])
```

[10]:
```
x.sum(2)
```

[10]:
```
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)

[11]:
```
x[2,0,1]
```

[11]:
```
17
```

특별한 ndarray를 생성하는 내장 함수들

numpy.arange( )

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

[12]:
```
n = np.arange(5)
n
```

[12]:
```
array([0, 1, 2, 3, 4])
```

[13]:
```
n.dtype
```

[13]:
```
dtype('int32')
```

[14]:
```
n = np.arange(5, 0, -1)
n
```

[14]:
```
array([5, 4, 3, 2, 1])
```

[15]:
```
x = np.arange(0, 1, 0.1)
x
```

[15]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

[16]:
```
x.dtype
```

[16]:
```
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

[17]:
```
x = np.linspace(1, 2, 5)
x
```

[17]:
```
array([1.  , 1.25, 1.5 , 1.75, 2.  ])
```

[18]:
```
x = np.linspace(1, 2, 5, endpoint=False)
x
```

[18]:
```
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

[19]:
```
np.logspace(1, 5, 5)
```

[19]:
```
array([1.e+01, 1.e+02, 1.e+03, 1.e+04, 1.e+05])
```

[20]:
```
np.logspace(0, 2, 3)
```

[20]:
```
array([  1.,  10., 100.])
```

np.zeros( )

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

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

[21]:
```
x = np.zeros(5)
x
```

[21]:
```
array([0., 0., 0., 0., 0.])
```

[22]:
```
x.dtype
```

[22]:
```
dtype('float64')
```

[23]:
```
x = np.zeros(5, dtype=np.int32)
x
```

[23]:
```
array([0, 0, 0, 0, 0])
```

[24]:
```
x.dtype
```

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

[25]:
```
np.zeros((3, 4))
```

[25]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[26]:
```
np.zeros((3, 4, 2))
```

[26]:
```
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

[27]:
```
x = np.arange(5)
x, x.dtype, x.shape
```

[27]:
```
(array([0, 1, 2, 3, 4]), dtype('int32'), (5,))
```

[28]:
```
y = np.linspace(0, 1, 2)
y, y.dtype, y.shape
```

[28]:
```
(array([0., 1.]), dtype('float64'), (2,))
```

[29]:
```
a = np.zeros_like(x)
a, a.dtype, a.shape
```

[29]:
```
(array([0, 0, 0, 0, 0]), dtype('int32'), (5,))
```

[30]:
```
b = np.zeros_like(y)
b, b.dtype, b.shape
```

[30]:
```
(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

[31]:
```
x = np.ones(5)
x
```

[31]:
```
array([1., 1., 1., 1., 1.])
```

[32]:
```
x.dtype
```

[32]:
```
dtype('float64')
```

[33]:
```
x = np.ones(5, dtype=np.int32)
x
```

[33]:
```
array([1, 1, 1, 1, 1])
```

[34]:
```
x.dtype
```

[34]:
```
dtype('int32')
```

[35]:
```
np.ones((3, 4))
```

[35]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[36]:
```
np.ones((3, 4, 2))
```

[36]:
```
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

[37]:
```
x = np.arange(5)
x, x.dtype, x.shape
```

[37]:
```
(array([0, 1, 2, 3, 4]), dtype('int32'), (5,))
```

[38]:
```
y = np.linspace(0, 1, 2)
y, y.dtype, y.shape
```

[38]:
```
(array([0., 1.]), dtype('float64'), (2,))
```

[39]:
```
a = np.ones_like(x)
a, a.dtype, a.shape
```

[39]:
```
(array([1, 1, 1, 1, 1]), dtype('int32'), (5,))
```

[40]:
```
b = np.ones_like(y)
b, b.dtype, b.shape
```

[40]:
```
(array([1., 1.]), dtype('float64'), (2,))
```

np.full( )

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

- 모든 원소가 fill_value인 ndarray 객체를 반환

[41]:
```
x = np.full(5, 3)
x
```

[41]:
```
array([3, 3, 3, 3, 3])
```

[42]:
```
x.dtype
```

[42]:
```
dtype('int32')
```

[43]:
```
x = np.full(5, 3.)
x
```

[43]:
```
array([3., 3., 3., 3., 3.])
```

[44]:
```
x.dtype
```

[44]:
```
dtype('float64')
```

[45]:
```
x = np.full((3, 4), 3.)
x
```

[45]:
```
array([[3., 3., 3., 3.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.]])
```

[46]:
```
x.dtype
```

[46]:
```
dtype('float64')
```

[47]:
```
x = np.full((3, 4, 2), 3+2j)
x
```

[47]:
```
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]]])
```

[48]:
```
x.dtype
```

[48]:
```
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

[49]:
```
x = np.arange(5)
x, x.dtype, x.shape
```

[49]:
```
(array([0, 1, 2, 3, 4]), dtype('int32'), (5,))
```

[50]:
```
y = np.ones((2, 3))
y, y.dtype, y.shape
```

[50]:
```
(array([[1., 1., 1.],
        [1., 1., 1.]]),
 dtype('float64'),
 (2, 3))
```

[51]:
```
a = np.full_like(x, 3.)
a, a.dtype, a.shape
```

[51]:
```
(array([3, 3, 3, 3, 3]), dtype('int32'), (5,))
```

[52]:
```
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
```

[52]:
```
(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

[53]:
```
x = np.eye(3)
x
```

[53]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[54]:
```
x.dtype
```

[54]:
```
dtype('float64')
```

[55]:
```
x = np.eye(3, 4)
x
```

[55]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]])
```

[56]:
```
x.dtype
```

[56]:
```
dtype('float64')
```

[57]:
```
np.eye(3, 2)
```

[57]:
```
array([[1., 0.],
       [0., 1.],
       [0., 0.]])
```

[58]:
```
np.eye(4)
```

[58]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

[59]:
```
np.eye(4, k=1)
```

[59]:
```
array([[0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.]])
```

[60]:
```
np.eye(4, k=2)
```

[60]:
```
array([[0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[61]:
```
np.eye(4, k=3)
```

[61]:
```
array([[0., 0., 0., 1.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[62]:
```
np.eye(4, k=4)
```

[62]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[63]:
```
np.eye(4, k=-1)
```

[63]:
```
array([[0., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]])
```

[64]:
```
np.eye(4, k=-2)
```

[64]:
```
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

[65]:
```
x = np.arange(16).reshape((4,4))
x
```

[65]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

[66]:
```
np.diag(x)
```

[66]:
```
array([ 0,  5, 10, 15])
```

[67]:
```
np.diag(x, k=1)
```

[67]:
```
array([ 1,  6, 11])
```

[68]:
```
np.diag(x, k=2)
```

[68]:
```
array([2, 7])
```

[69]:
```
np.diag(x, k=3)
```

[69]:
```
array([3])
```

[70]:
```
np.diag(x, k=4)
```

[70]:
```
array([], dtype=int32)
```

[71]:
```
x = np.arange(12).reshape((3,4))
x
```

[71]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[72]:
```
np.diag(x)
```

[72]:
```
array([ 0,  5, 10])
```

[73]:
```
np.diag([1,2,3])
```

[73]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

[74]:
```
np.diag([1,2,3], 1)
```

[74]:
```
array([[0, 1, 0, 0],
       [0, 0, 2, 0],
       [0, 0, 0, 3],
       [0, 0, 0, 0]])
```

[75]:
```
np.diag([1,2,3], -2)
```

[75]:
```
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]])
```

[76]:
```
np.diag([1,2,3])[:, ::-1]
```

[76]:
```
array([[0, 0, 1],
       [0, 2, 0],
       [3, 0, 0]])
```

[77]:
```
np.diag([1,2,3])[::-1, :]
```

[77]:
```
array([[0, 0, 3],
       [0, 2, 0],
       [1, 0, 0]])
```

[78]:
```
np.diag([1,2,3])[::-1, ::-1]
```

[78]:
```
array([[3, 0, 0],
       [0, 2, 0],
       [0, 0, 1]])
```

NumPy 특수 다차원 배열: Random 모듈

- randint(low, high=None, size=None, dtype='l'): discrete uniform in [low, high)
- random(size=None): uniform in [0, 1)
- uniform(low=0.0, high=1.0, size=None)
- normal(loc=0.0, scale=1.0, size=None): 평균이 loc, 표준편차가 scale인 정규 분포

[79]:
```
np.random.randint(5)
```

[79]:
```
0
```

[80]:
```
np.random.randint(5)
```

[80]:
```
1
```

[81]:
```
a = np.random.randint(5, size=0)
a
```

[81]:
```
array([], dtype=int32)
```

[82]:
```
a = np.random.randint(5, size=5)
a
```

[82]:
```
array([0, 3, 1, 1, 0])
```

[83]:
```
a.dtype
```

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

[84]:
```
np.random.randint(1, 5, 3)
```

[84]:
```
array([4, 3, 4])
```

[85]:
```
np.random.randint(0, 10, (2,3))
```

[85]:
```
array([[3, 2, 8],
       [3, 9, 2]])
```

NumPy 특수 다차원 배열: Random 모듈

- random(size=None): uniform in [0, 1)

[86]:
```
np.random.random()
```

[86]:
```
0.08099517116500765
```

[87]:
```
np.random.random(5)
```

[87]:
```
array([0.15844388, 0.02709438, 0.31469885, 0.03887372, 0.70487734])
```

[88]:
```
np.random.random((2,3, 4))
```

[88]:
```
array([[[0.79102182, 0.15437033, 0.5984633 , 0.70466964],
        [0.98460769, 0.92573979, 0.86382694, 0.56805604],
        [0.42101223, 0.51701756, 0.62339756, 0.76542793]],

       [[0.82238187, 0.16044849, 0.00315128, 0.16383448],
        [0.01075955, 0.81549588, 0.14928465, 0.12992061],
        [0.65738482, 0.37404742, 0.50494732, 0.867269  ]]])
```

NumPy 특수 다차원 배열: Random 모듈

- uniform(low=0.0, high=1.0, size=None)

[89]:
```
np.random.uniform()
```

[89]:
```
0.904477030075104
```

[90]:
```
np.random.uniform(1.5)
```

[90]:
```
1.0689757457080422
```

[91]:
```
np.random.uniform(-1, 1)
```

[91]:
```
-0.6380194746263803
```

[92]:
```
np.random.uniform(-1, 1, 4)
```

[92]:
```
array([ 0.16311255, -0.29497972,  0.63802044, -0.49072912])
```

[93]:
```
np.random.uniform(-2, 2, (2, 3, 4))
```

[93]:
```
array([[[-1.99568049, -0.36937427,  0.9346248 , -1.01997593],
        [ 1.31505814,  1.63084835, -1.2456362 , -1.11097509],
        [-1.43620001, -1.17238989,  0.55586831, -1.30034263]],

       [[-0.37553704, -0.36082677, -0.63862288,  1.5112186 ],
```

```
       [ 0.72325147, -1.30125953, -1.01682735,  1.85444955],
       [ 1.42186015, -0.96812752, -0.07569636, -0.55337475]]])
```

NumPy 특수 다차원 배열: Random 모듈

- normal(loc=0.0, scale=1.0, size=None): 평균이 loc, 표준편차가 scale인 정규 분포

[94]:
```
np.random.normal(5, 0.3, 4)
```

[94]:
```
array([4.7723514 , 5.06688916, 4.99655035, 4.60064481])
```

NumPy 특수 다차원 배열: Random 모듈

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

[95]:
```
np.random.choice(5)
```

[95]:
```
0
```

[96]:
```
np.random.choice(5, (2,3))
```

[96]:
```
array([[1, 3, 2],
       [2, 0, 4]])
```

[97]:
```
lst = ['S', 'D', 'C', 'H']
np.random.choice(lst)
```

[97]:
```
'D'
```

[98]:
```
lst = ['S', 'D', 'C', 'H']
np.random.choice(lst, (2,3))
```

[98]:
```
array([['H', 'D', 'S'],
       ['H', 'H', 'S']], dtype='<U1')
```

[99]:
```
lst = ['S', 'D', 'C', 'H']
s = np.random.choice(lst, 200, p=[0.1, 0.2, 0.3, 0.4])
```

[100]:
```
from collections import Counter
Counter(s)
```

[100]:
```
Counter({'C': 64, 'H': 89, 'D': 21, 'S': 26})
```