

파이썬 라이브러리를 활용한 데이터 분석

3장 numpy 기본: 배열과 벡터 연산

2020.06.19(금) 2h

실습

- **교재 파일**
 - ch04.ipynb
 - Ch04-study.ipynb로 복사해서 연습
- **난수와 실수의 정확도**
 - import numpy as np
 - np.random.seed(12345)
 - 난수를 발생시키기 위한 초기 값 지정
 - 이후 난수가 동일하게 발생
 - np.set_printoptions(precision=4, suppress=True)
 - precision=4: 소수점 이하 반올림해 4개 표시
 - np.array(3.123456)
 - array(3.1235)
 - suppress=True: e-04와 같은 scientific notation을 억제하고 싶으면
- **Alt + Enter**
 - 현재 셀 실행 후, 다음 셀 삽입
- **Ctrl + shift + enter**
 - 셀 분리

브로드캐스팅 슬라이싱

4.1.3 numpy 배열의 산술 연산

4.1.4 색인과 슬라이스 기초

4.1.3 numpy 배열 산술 연산

• 벡터화

- for 문 없이 데이터를 일괄 처리
 - 같은 크기의 배열은 각 원소 별로 연산
 - 스칼라 인자
 - 배열 내의 모든 원소에 스칼라 인자가 적용
- 배열의 비교 연산
 - 결과: 불리언 배열

```
In [57]: arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
```

```
In [58]: arr2
```

```
Out[58]:
```

```
array([[ 0.,  4.,  1.],
        [ 7.,  2., 12.]])
```

```
In [59]: arr2 > arr
```

```
Out[59]:
```

```
array([[False,  True, False],
        [ True, False,  True]], dtype=bool)
```

• 브로드캐스팅

- 크기가 다른 배열 간의 연산

```
In [51]: arr = np.array([[1., 2., 3.], [4., 5., 6.]])
```

```
In [52]: arr
```

```
Out[52]:
```

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

```
In [53]: arr * arr
```

```
Out[53]:
```

```
array([[ 1.,  4.,  9.],
        [16., 25., 36.]])
```

```
In [54]: arr - arr
```

```
Out[54]:
```

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

```
In [55]: 1 / arr
```

```
Out[55]:
```

```
array([[ 1.    ,  0.5   ,  0.3333],
        [ 0.25  ,  0.2   ,  0.1667]])
```

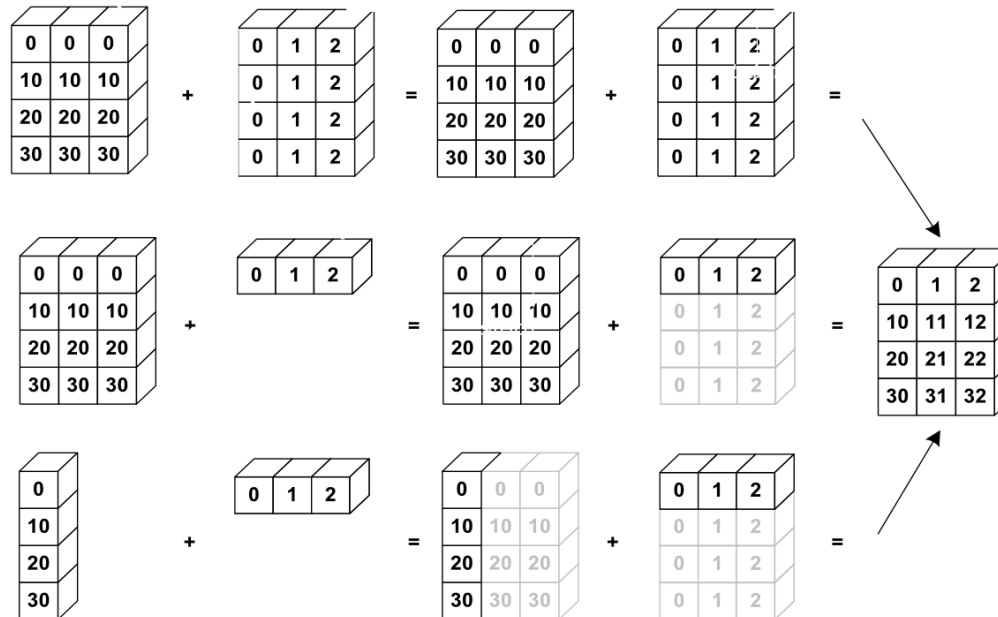
```
In [56]: arr ** 0.5
```

```
Out[56]:
```

```
array([[ 1.    ,  1.4142,  1.7321],
        [ 2.    ,  2.2361,  2.4495]])
```

브로드캐스팅

- Shape이 같은 두 배열에 대한 이항 연산은 배열의 요소별로 수행
- 두 배열 간의 Shape이 다를 경우
 - 크기가 다른 배열 간의 연산
 - 두 배열 간의 형상을 맞추는 Broadcasting 과정을 거쳐 계산



브로드캐스팅 예

```
In [8]: # 데모 배열 생성
a = np.arange(5).reshape((1, 5))
pprint(a)
b = np.arange(5).reshape((5, 1))
pprint(b)

shape: (1, 5), dimension: 2, dtype:int64
Array's Data
[[0 1 2 3 4]]
shape: (5, 1), dimension: 2, dtype:int64
Array's Data
[[0]
 [1]
 [2]
 [3]
 [4]]
```

0	1	2	3	4
0	1	2	3	4
0	1	2	3	4
0	1	2	3	4
0	1	2	3	4

0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

```
In [9]: a+b
```

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

4.1.4 색인과 슬라이싱 기초

- **부분이나 한 원소의 참조**
 - 반환 값의 배열 조각은 원본 배열의 뷰
 - 그러므로 반환 값을 수정하면 원본에도 반영
 - 대용량의 데이터 처리를 위해 설계
 - 복사하기 위해서는
 - `arr[5:8].copy()`
 - 2차원 배열 원소 참조
 - `arr[행][열]`
 - `arr[행, 열]`

슬라이싱

- 여러 개의 배열 요소를 참조할 때 슬라이싱을 사용
- axis 별로 범위를 지정하여 실행
 - 범위 from_index:to_index 형태로 지정
 - from_index는 범위의 시작 인덱스이며, to_index는 범위의 종료 인덱스
 - to_index는 결과에 포함되지 않음
 - from_index는 생략 가능, 생략할 경우 0을 지정한 것으로 간주
 - to_index 역시 생략 가능, 이 경우 마지막 인덱스로 설정
 - ":" 형태로 지정된 범위는 전체 범위를 의미

인덱싱과 슬라이싱



[Python NumPy]

Indexing and Slicing of an ndarray

Indexing a subset of 1D array

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

`a[0:5]`

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

Not a copy, but a VIEW!!!

<http://rfriend.tistory.com>

Indexing a subset of 2D array

```
d = array([[ 0,  1,  2,  3,  4],
           [ 5,  6,  7,  8,  9],
           [10, 11, 12, 13, 14],
           [15, 16, 17, 18, 19]])
```

`d[0:3, 1:3]`

```
array([[ 1,  2],
       [ 6,  7],
       [11, 12]])
```

슬라이싱

Array Slicing

SLICING WORKS MUCH LIKE
STANDARD PYTHON SLICING

```
>>> a[0,3:5]
array([3, 4])
```

```
>>> a[4:,4:]
array([[44, 45],
       [54, 55]])
```

```
>>> a[:,2]
array([2, 12, 22, 32, 42, 52])
```

STRIDES ARE ALSO POSSIBLE

```
>>> a[2::2,::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

정수 색인: 차원이 작아짐

모두 슬라이스 사용: 차원이 같음

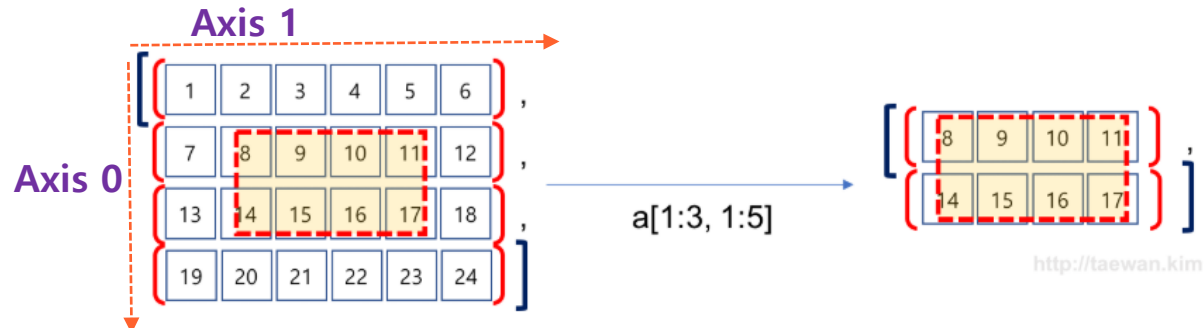
0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

2차원 배열의 축과 슬라이싱

```
In [1]: # 데모 배열 생성
a1 = np.arange(1, 25).reshape((4, 6)) #2차원 배열
pprint(a1)
```

```
shape: (4, 6), dimension: 2, dtype:int64
Array's Data
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

가운데 요소 가져오기



		axis 1		
		0	1	2
axis 0	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

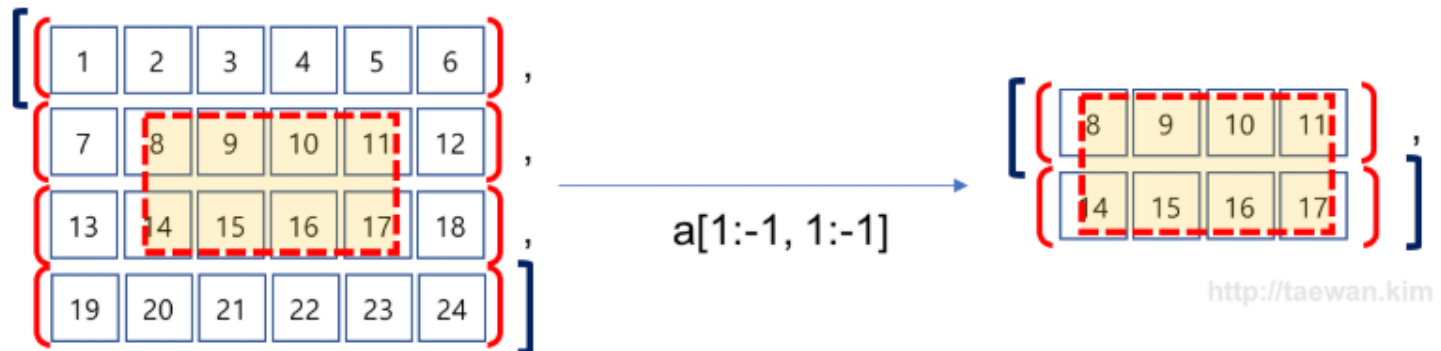
```
In [2]: a1[1:3, 1:5]
```

```
Out[2]: array([[ 8,  9, 10, 11],
               [14, 15, 16, 17]])
```

음수 인덱스를 이용한 범위 설정

음수 인덱스

- 지정한 axis의 마지막 요소로 부터 반대 방향의 인덱스
- -1은 마지막 요소의 인덱스를 의미



```
In [3]: a1[1:-1, 1:-1]
```

```
Out[3]: array([[ 8,  9, 10, 11],
               [14, 15, 16, 17]])
```

다양한 슬라이싱 활용

• 슬라이싱에 대입도 가능

```
In [4]: # 데모 대상 배열 조회
pprint(a1)

shape: (4, 6), dimension: 2, dtype:int64
Array's Data
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

```
In [5]: # 슬라이싱 배열
slide_arr = a1[1:3, 1:5]
pprint(slide_arr)

shape: (2, 4), dimension: 2, dtype:int64
Array's Data
[[ 8  9 10 11]
 [14 15 16 17]]
```

```
In [6]: # 슬라이싱 결과 배열에 슬라이싱을 적용하여 4개 요소 참조
slide_arr[:, 1:3]
pprint(slide_arr)

shape: (2, 4), dimension: 2, dtype:int64
Array's Data
[[ 8  9 10 11]
 [14 15 16 17]]
```

```
In [7]: # 슬라이싱을 적용하여 참조한 4개 요소 업데이트 및 슬라이싱
        배열 조회
slide_arr[:, 1:3]=99999
pprint(slide_arr)

shape: (2, 4), dimension: 2, dtype:int64
Array's Data
[[  8 99999 99999  11]
 [ 14 99999 99999  17]]
```

슬라이싱 정리





	Expression	Shape
	<code>arr[:2, 1:]</code>	<code>(2, 2)</code>
	<code>arr[2]</code>	<code>(3,)</code>
	<code>arr[2, :]</code>	<code>(3,)</code>
	<code>arr[2:, :]</code>	<code>(1, 3)</code>
	<code>arr[:, :2]</code>	<code>(3, 2)</code>
	<code>arr[1, :2]</code>	<code>(2,)</code>
	<code>arr[1:2, :2]</code>	<code>(1, 2)</code>

Figure 4-2. Two-dimensional array slicing

4.1.5 불리언 값으로 행 선택하기

• 불리언 배열

- 색인하려는 축(행, 0)
의 길이와 같아야 함
 - 7

Boolean Indexing

```
In [14]: names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
data = np.random.randn(7, 4)
names
```

```
Out[14]: array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')
```

```
In [15]: data
```

```
Out[15]: array([[ -0.2047,  0.4789, -0.5194, -0.5557],
 [ 1.9658,  1.3934,  0.0929,  0.2817],
 [ 0.769 ,  1.2464,  1.0072, -1.2962],
 [ 0.275 ,  0.2289,  1.3529,  0.8864],
 [-2.0016, -0.3718,  1.669 , -0.4386],
 [-0.5397,  0.477 ,  3.2489, -1.0212],
 [-0.5771,  0.1241,  0.3026,  0.5238]])
```

```
In [16]: names == 'Bob'
```

```
Out[16]: array([ True, False, False,  True, False, False, False])
```

```
In [24]: data[names == 'Bob']
```

```
Out[24]: array([[ -0.2047,  0.4789, -0.5194, -0.5557],
 [ 0.275 ,  0.2289,  1.3529,  0.8864]])
```

```
In [26]: data[:4:3]
```

```
Out[26]: array([[ -0.2047,  0.4789, -0.5194, -0.5557],
 [ 0.275 ,  0.2289,  1.3529,  0.8864]])
```

논리 값 배열로 열 선택

• 열의 길이에 맞게

– 4

```
In [34]: data
```

```
Out[34]: array([[ -0.2047,  0.4789, -0.5194, -0.5557],
                [ 1.9658,  1.3934,  0.0929,  0.2817],
                [ 0.769 ,  1.2464,  1.0072, -1.2962],
                [ 0.275 ,  0.2289,  1.3529,  0.8864],
                [-2.0016, -0.3718,  1.669 , -0.4386],
                [-0.5397,  0.477 ,  3.2489, -1.0212],
                [-0.5771,  0.1241,  0.3026,  0.5238]])
```

```
In [33]: test = np.array([True, False, True, False])
        test
```

```
Out[33]: array([ True, False,  True, False])
```

```
In [32]: data[:, test] #0, 3 열 선택
```

```
Out[32]: array([[ -0.2047, -0.5194],
                [ 1.9658,  0.0929],
                [ 0.769 ,  1.0072],
                [ 0.275 ,  1.3529],
                [-2.0016,  1.669 ],
                [-0.5397,  3.2489],
                [-0.5771,  0.3026]])
```


4.1.6 팬시 인덱싱(Fancy Indexing)

- 특정 순서로 로우를 선택
 - 순서가 명시된 ndarray 나 리스트를 사용
- 특정 순서로 열을 선택

```
In [49]: arr = np.arange(32).reshape((8, 4))
arr
```

```
Out[49]: array([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11],
                [12, 13, 14, 15],
                [16, 17, 18, 19],
                [20, 21, 22, 23],
                [24, 25, 26, 27],
                [28, 29, 30, 31]])
```

```
In [50]: arr[:, [1, 0]]
```

```
Out[50]: array([[ 1,  0],
                [ 5,  4],
                [ 9,  8],
                [13, 12],
                [17, 16],
                [21, 20],
                [25, 24],
                [29, 28]])
```

Fancy Indexing

```
In [35]: arr = np.empty((8, 4))
for i in range(8):
    arr[i] = i
arr
```

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

```
In [37]: arr[[4, 3, 0, 6]]
```

```
Out[37]: array([[4., 4., 4., 4.],
                [3., 3., 3., 3.],
                [0., 0., 0., 0.],
                [6., 6., 6., 6.]])
```

```
In [39]: arr[4, 3, 0, 6]
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-39-0e5c5006e5da> in <module>
----> 1 arr[4, 3, 0, 6]
```

```
IndexError: too many indices for array
```

```
In [40]: arr[[-3, -5, -7]]
```

```
Out[40]: array([[5., 5., 5., 5.],
                [3., 3., 3., 3.],
                [1., 1., 1., 1.]])
```

팬시 인덱싱(Fancy Indexing)

- 배열에 인덱스 배열을 전달하여 요소를 참조하는 방법

```
In [2]: arr = np.arange(1, 25).reshape((4, 6))
        pprint(arr)

shape: (4, 6), dimension: 2, dtype:int64
Array's Data
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

Fancy Case 1

```
In [3]: [arr[0,0], arr[1, 1], arr[2, 2], arr[3, 3]]
```

```
Out[3]: [1, 8, 15, 22]
```

```
In [4]: # 두 배열을 전달==> (0, 0), (1,1), (2,2), (3, 3)
        arr[[0, 1, 2, 3], [0, 1, 2, 3]]
```

```
Out[4]: array([ 1,  8, 15, 22])
```

Fancy Case 2

```
In [5]: # 전체 행에 대해서, 1, 2번 컬럼 참조
        arr[:, [1, 2]]
```

```
Out[5]: array([[ 2,  3],
               [ 8,  9],
               [14, 15],
               [20, 21]])
```

다차원 색인 배열

• 각각의 색인 튜플에 대응하는 1차원 배열 선택

```
In [51]: arr = np.arange(32).reshape((8, 4))
arr
```

```
Out[51]: array([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11],
                [12, 13, 14, 15],
                [16, 17, 18, 19],
                [20, 21, 22, 23],
                [24, 25, 26, 27],
                [28, 29, 30, 31]])
```

```
In [59]: arr[[1, 5, 7, 2], [0, 3, 1, 2]]
```

```
Out[59]: array([ 4, 23, 29, 10])
```

```
In [63]: s = [(x, y) for x, y in zip([1, 5, 7, 2], [0, 3, 1, 2])]
s
```

```
Out[63]: [(1, 0), (5, 3), (7, 1), (2, 2)]
```

```
In [62]: s = [arr[x, y] for x, y in zip([1, 5, 7, 2], [0, 3, 1, 2])]
s
```

```
Out[62]: [4, 23, 29, 10]
```

4.1.7 배열 전치와 축 바꾸기

• 배열 전치

- 행 열을 바꾸는 것
 - 모양이 바뀐 뷰를 반환
 - T라는 속성

• 다차원 배열

- 축 번호를 받아서 교환
 - 첫 번째와 두 번째 축 번호가 바뀜
- 첨자 교환 예, 5
 - 첨자: (0, 1, 1)
 - 교환
 - 수정된 첨자: (1, 0, 1)

```
In [64]: arr = np.arange(16).reshape((2, 2, 4))
arr # (2, 2, 4)
```

```
Out [64]: array([[[ 0,  1,  2,  3],
                  [ 4,  5,  6,  7]],
                 [[ 8,  9, 10, 11],
                  [12, 13, 14, 15]]])
```

```
In [ ]: # p157
arr.transpose((0, 1, 2)) # (2, 2, 4)
```

```
In [ ]: arr.transpose((0, 2, 1)) # , 축 1과 2를 바꾸기 (2, 4, 2)
```

```
In [65]: arr.transpose((1, 0, 2)) # (2, 2, 4)
```

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

배열 형태 변경: 전치(Transpose)

- 행렬의 인덱스가 바뀌는 변환

- `[numpy.ndarray 객체].T` 속성

$$\begin{bmatrix} 1 & 2 \end{bmatrix}^T = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

```
In [1]: # 행렬 생성
a = np.random.randint(1, 10, (2, 3))
pprint(a)

shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[6 6 1]
 [4 1 3]]
```

```
In [2]: #행렬의 전치
pprint(a.T)

shape: (3, 2), dimension: 2, dtype:int64
Array's Data
[[6 4]
 [6 1]
 [1 3]]
```

배열 형태 변경: reshape 메서드 이해

- [numpy.ndarray 객체]의 shape을 변경한 것을 반환
 - 자체는 변환되지 않음

```
In [1]: # 대상 행렬 속성 확인
a = np.random.randint(1, 10, (2, 3))
pprint(a)

shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[7 7 8]
 [3 9 1]]
```

```
In [2]: result = a.reshape((3, 2, 1))
pprint(result)

shape: (3, 2, 1), dimension: 3, dtype:int64
Array's Data
[[[7]
 [7]]

 [[8]
 [3]]

 [[9]
 [1]]]
```

배열 형태 변경: reshape 메서드의 활용

```
>>> a = np.array([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3]])
>>> a
array([[1, 1, 1, 1],
       [2, 2, 2, 2],
       [3, 3, 3, 3]])
>>> b = a.reshape(2, 6) # same as np.reshape(a, [2, 6])
>>> b
array([[1, 1, 1, 1, 2, 2],
       [2, 2, 3, 3, 3, 3]])
```

Original

(3, 4)

1	1	1	1
2	2	2	2
3	3	3	3

(6, 2)

1	1
1	1
2	2
2	2
3	3
3	3

(2, 6)

1	1	1	1	2	2
2	2	3	3	3	3

(4, 3)

1	1	1
1	2	2
2	2	3
3	3	3

(1, 12)

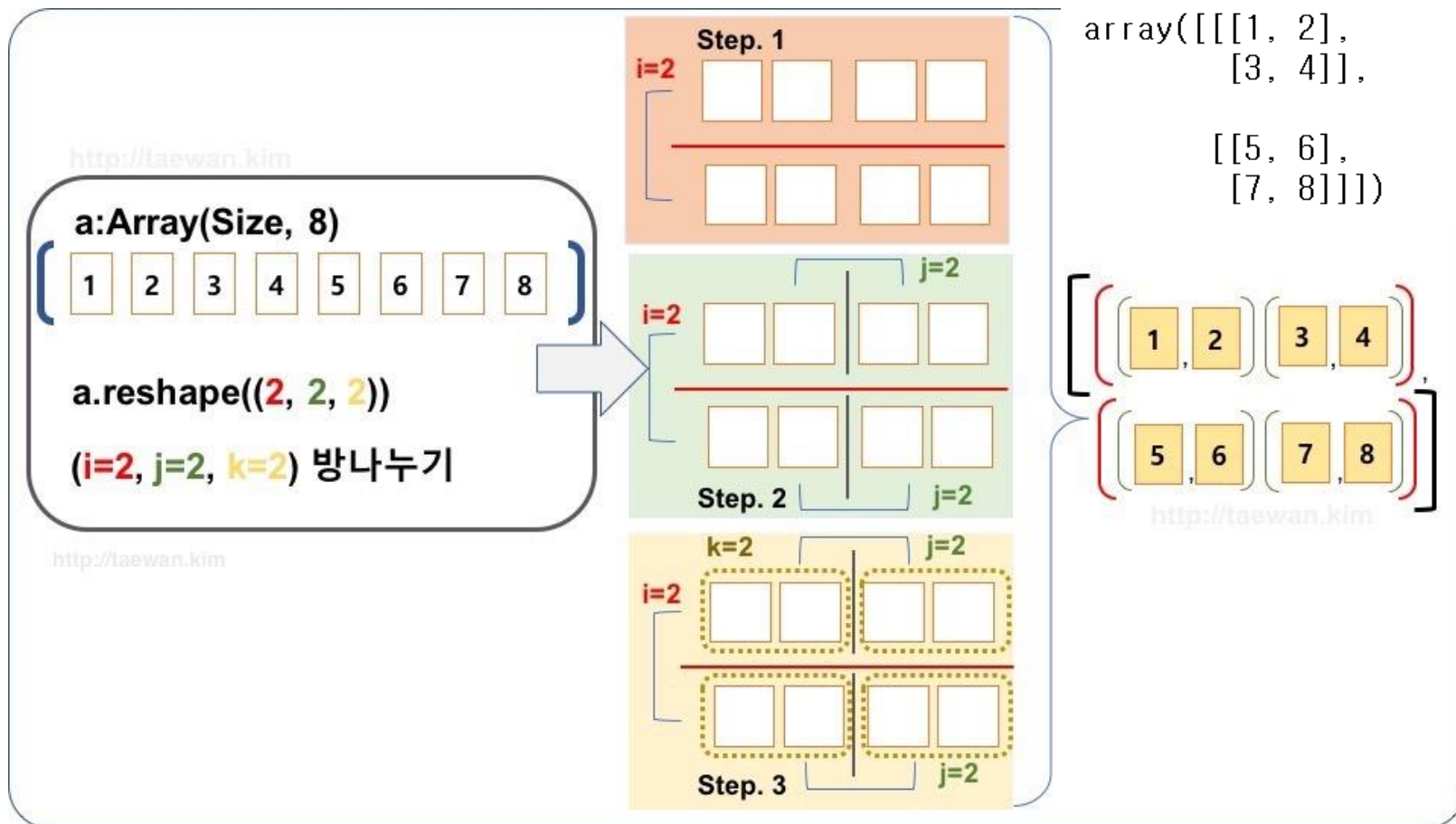
1	1	1	1	2	2	2	2	3	3	3	3
---	---	---	---	---	---	---	---	---	---	---	---

(12, 1)

1
1
1
1
2
2
2
2
3
3
3
3

삼차원 배열의 이해

- `np.arange(1, 9).reshape(2, 2, 2)`



3차원 배열의 전치 transpose

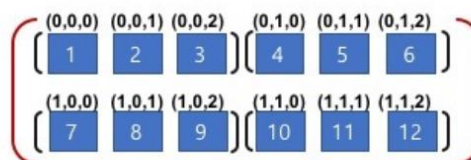
```
In [275]: a = np.arange(1, 13)
a
```

```
Out[275]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

```
In [279]: aaa = a.reshape((2, 2, 3))
print(aaa.shape)
aaa
```

Shape: (2, 2, 3)

```
Out[279]: array([[[ 1,  2,  3],
                  [ 4,  5,  6]],
                [[ 7,  8,  9],
                  [10, 11, 12]]])
```



<http://taewan.kim>

```
In [283]: ccc = np.transpose(aaa, (1, 2, 0))
print(ccc.shape)
ccc
```

(2, 3, 2)

```
Out[283]: array([[[ 1,  7],
                  [ 2,  8],
                  [ 3,  9]],
                [[ 4, 10],
                  [ 5, 11],
                  [ 6, 12]]])
```

i=0, j=1, k=2

(1, 2, 0)로 axis를 지정하면
aaa의 첫 번째 Shape이
출력 텐서의 마지막으로 이동

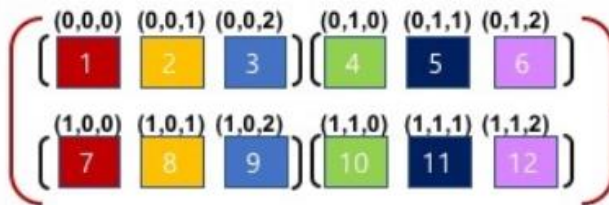
입력 텐서의 **ijk** 요소가
출력 텐서에서는 **jki**로 이동

<http://taewan.kim>

arr.transpose의 이해

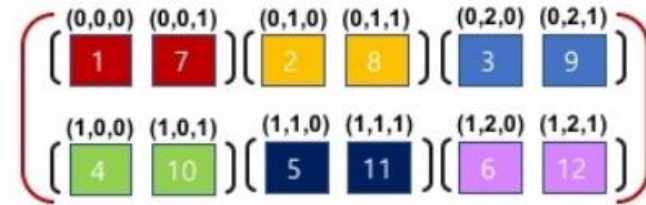
<http://taewan.kim>

aaa



`np.transpose(aaa, (1, 2, 0))`

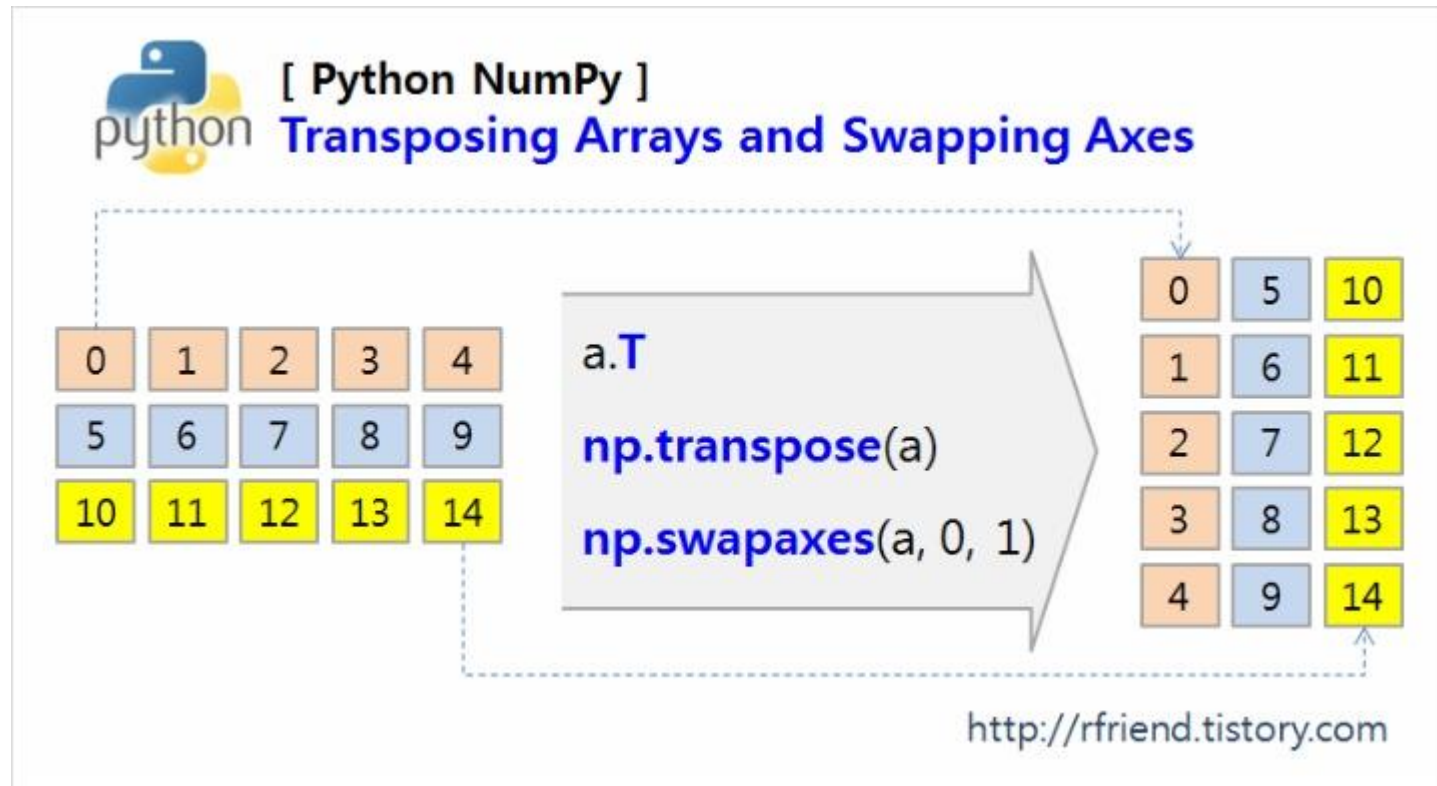
returned/
transposed
tensor



Element	Input Tensor Index	Transpose Axis	output Tensor Index
1	(<i>i</i> , <i>j</i> , <i>k</i>) = (0, 0, 0)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (0, 0, 0)
2	(<i>i</i> , <i>j</i> , <i>k</i>) = (0, 0, 1)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (0, 1, 0)
3	(<i>i</i> , <i>j</i> , <i>k</i>) = (0, 0, 2)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (0, 2, 0)
4	(<i>i</i> , <i>j</i> , <i>k</i>) = (0, 1, 0)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (1, 0, 0)
5	(<i>i</i> , <i>j</i> , <i>k</i>) = (0, 1, 1)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (1, 1, 0)
6	(<i>i</i> , <i>j</i> , <i>k</i>) = (0, 1, 2)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (1, 2, 0)
7	(<i>i</i> , <i>j</i> , <i>k</i>) = (1, 0, 0)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (0, 0, 1)
8	(<i>i</i> , <i>j</i> , <i>k</i>) = (1, 0, 1)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (0, 1, 1)
9	(<i>i</i> , <i>j</i> , <i>k</i>) = (1, 0, 2)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (0, 2, 1)
10	(<i>i</i> , <i>j</i> , <i>k</i>) = (1, 1, 0)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (1, 0, 1)
11	(<i>i</i> , <i>j</i> , <i>k</i>) = (1, 1, 1)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (1, 1, 1)
12	(<i>i</i> , <i>j</i> , <i>k</i>) = (1, 1, 2)	(1, 2, 0)	(<i>j</i> , <i>k</i> , <i>i</i>) = (1, 2, 1)

arr.swapaxes()

- 두 개의 축 번호를 교환



배열 형태 변경: ravel 메서드

- 배열의 shape을 1차원 배열로 만드는 메서드
 - [numpy.ndarray 객체].ravel()
 - 배열을 1차원 배열로 반환하는 메서드
 - 자체는 변환되지 않음
 - 주의
 - ravel이 반환하는 배열은 a 행렬의 view
 - 반환 행렬의 데이터를 변경하면 a 행렬도 변경

```
In [1]: # 데모 배열 생성
a = np.random.randint(1, 10, (2, 3))
pprint(a)

shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[7 9 6]
 [2 1 1]]
```

```
In [2]: a.ravel()
```

```
Out[2]: array([7, 9, 6, 2, 1, 1])
```

```
In [3]: b = a.ravel()
pprint(b)

shape: (6,), dimension: 1, dtype:int64
Array's Data
[7 9 6 2 1 1]
```

```
In [4]: b[0]=99
pprint(b)

shape: (6,), dimension: 1, dtype:int64
Array's Data
[99 9 6 2 1 1]
```

```
In [5]: # b 배열 변경이 a 행렬에 반영되어 있습니다.
pprint(a)

shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[99 9 6]
 [ 2 1 1]]
```

배열 슬라이스도 배열을 참조

• P144

- Pythontutor.com에서 테스트

```
import numpy as np
arr = np.arange(10)
a = arr[3:5]
a[0] = 20
print(arr)
```

← → ↻ 주의 요함 | pythontutor.com/visualize.html#mode=display

Get live help! These Python Tutor users are asking for help right now. Please volunteer to help!

- user_1b7 from Auckland, New Zealand needs help with Python3 - 2 people chatting - [click to help](#) (active a few seconds ago, requested 26 minutes ago)
- user_8fe from Jacksonville, Florida, US needs help with Python3 - [click to help](#) (active a few seconds ago, requested a few seconds ago)

Start private chat

Python 3.6 with [Anaconda 5.2 EXPERIMENTAL!](#)
(much slower than [regular Python 3.6](#), but lets you import more modules)

```
1 import numpy as np
2 arr = np.arange(10)
3 a = arr[3:5]
4 a[0] = 20
→ 5 print(arr)
```

[Edit this code](#)

→ line that just executed
→ next line to execute

Print output (drag lower right corner to resize)

```
[ 0  1  2 20  4  5  6  7  8  9]
```

Frames

Global frame

- np → module instance
- arr → ndarray instance [0 1 2 20 4 5 6 7 8 9]
- a → ndarray instance [20 4]

Done running (5 steps)

[Customize visualization](#) (NEW!)