수DA쟁이

Python for Data Analysis

A.1(p146~)~A.4

Department of Mathematics
Gyeongsang National University
Youngmin Shin

A.1 NumPy ndarray 객체 구조

A.1 NumPy ndarray 객체 구조

In [2]:
```
a=np.ones((10,5))
a
```

Out[2]:
```
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., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

In [3]:
```
np.ones((10,5)).shape
# shape이라는 함수를 사용해서 배열의 크기를 알아낼 수 있다
```

Out[3]:
```
(10, 5)
```

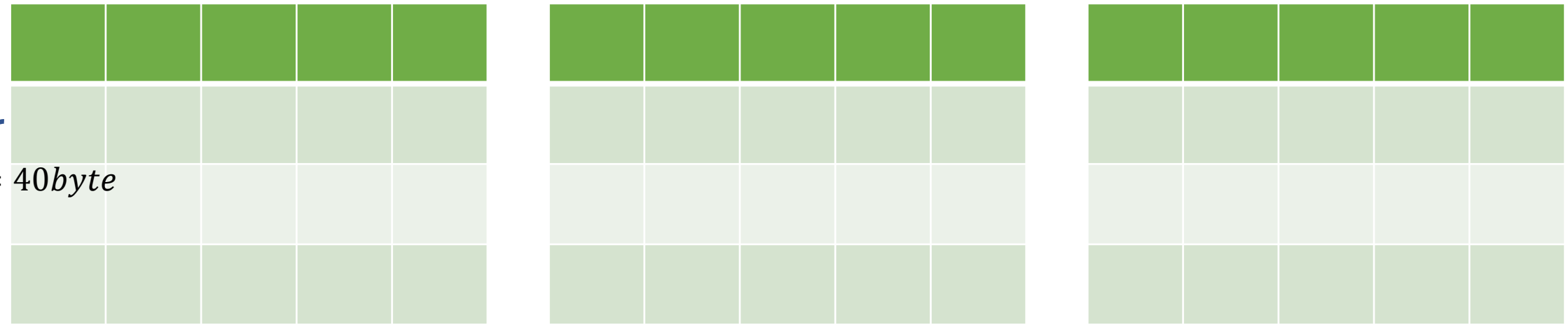
In [4]:
```
np.ones((3,4,5))
```

Out[4]:
```
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., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

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

In [5]:
```
np.ones((3,4,5),dtype=np.float64).strides
# (4*5*8,5*8,1*8) float64는 8바이트 이기 때문이다.
# 이 때 strides 값이 음수이면 뒤로 이동하는 의미이다.
```

Out[5]:
```
(160, 40, 8)
```

하나당 $8byte$



$5 \times 8 = 40byte$

$4 \times 5 \times 8 = 160byte$

A.1.1 NumPy dtype 구조

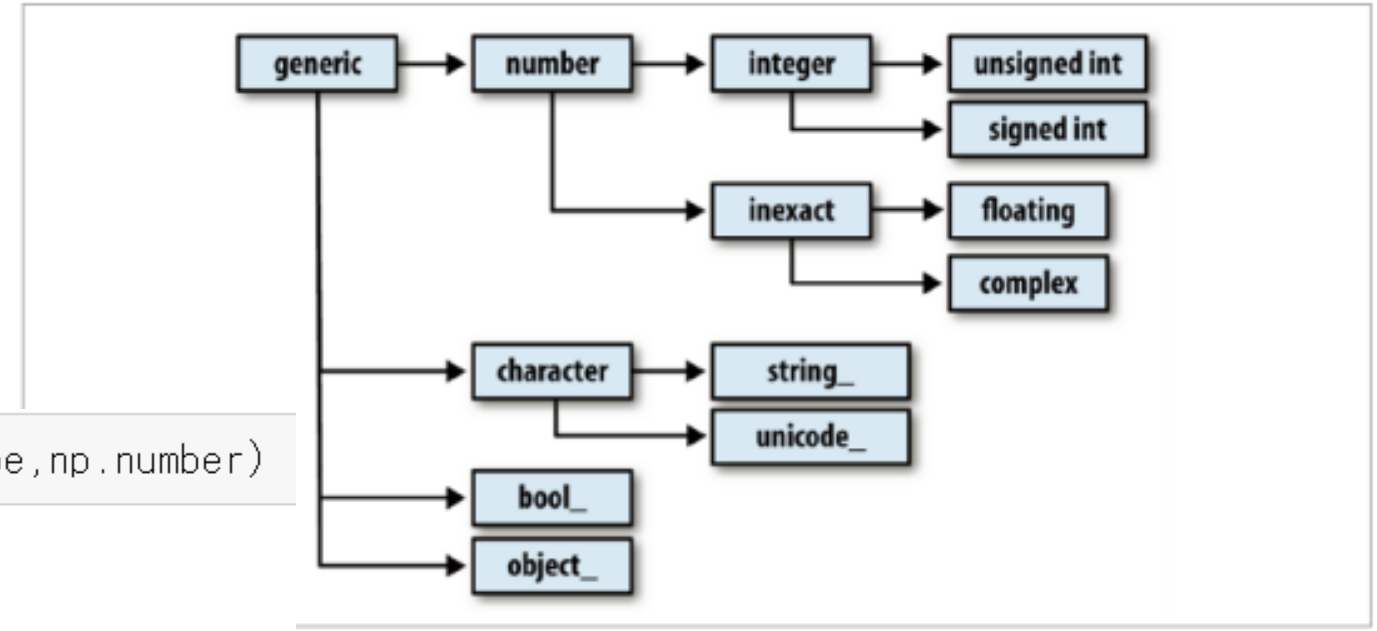
```
In [6]: ints=np.ones(10,dtype=np.uint16)

In [7]: floats=np.ones(10,dtype=np.float32)

In [8]: np.issubdtype(ints.dtype,np.integer)
Out[8]: True

In [9]: np.issubdtype(floats.dtype,np.floating)
Out[9]: True
```

배열에 담긴 값의 datatype을 확인해야 할 필요가 있음. 실수에도 다양한 형태가 있다. dtype은 np.issubtype 함수와 결합하여 사용할 수 있는 np.integer나 np.floating 같은 부모 클래스를 가진다.

A.1.1 NumPy dtype 구조

```
In [10]: np.float64.mro()
         # 부모 클래스는 mro 메서드를 사용해서 알 수 있다.

Out[10]: [numpy.float64,
          numpy.floating,
          numpy.inexact,
          numpy.number,
          numpy.generic,
          float,
          object]
```

그림 A-2 NumPy dtype 클래스 계층




```
In [11]: np.issubdtype(ints.dtype,np.number)

Out[11]: True
```

A.2 고급 배열 조작 기법

A.2.1 배열 재 형성하기

```
In [12]: arr=np.arange(8)

In [13]: arr
Out[13]: array([0, 1, 2, 3, 4, 5, 6, 7])

In [14]: b=arr.reshape((4,2))
         b
         # reshape 함수를 사용해서 배열의 모양을 변환
Out[14]: array([[0, 1],
                [2, 3],
                [4, 5],
                [6, 7]])
```

```
In [15]: c=arr.reshape((4,2),order='C')
         c
         # 'C' 로우 우선
Out[15]: array([[0, 1],
                [2, 3],
                [4, 5],
                [6, 7]])

In [16]: d=arr.reshape((4,2),order='F')
         d
         # 'F' 컬럼 우선
Out[16]: array([[0, 4],
                [1, 5],
                [2, 6],
                [3, 7]])
```

A.2.1 배열 재 형성하기

In [17]:
```
arr
```

Out[17]:
```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [18]:
```
arr.reshape((4,2)).reshape((2,4))
# 다차원 배열 또한 재형성이 가능하다
```

Out[18]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

In [19]:
```
arr=np.arange(15)
arr
```

Out[19]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 1
3, 14])
```

In [20]:
```
arr.reshape((5,-1))
# reshape에 넘기는 값 중 하나가 -1이 될 수 도 있는데
# 원본 데이터를 참조해서 적절한 값을 추론
```

Out[20]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
```

In [21]:
```
other_arr=np.ones((3,5))
other_arr
```

Out[21]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

In [22]:
```
other_arr.shape
```

Out[22]:
```
(3, 5)
```

In [23]:
```
arr.reshape(other_arr.shape)
```

Out[23]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

배열의 shape속성은 튜플이기 때문에 reshape 메서드에 이를 넘기는 것도 가능하다.

A.2.2 C 순서와 포트란 순서

로우와 컬럼 우선 순서는 각각 C순서와 포트란 순서로 알려져 있다.

In [28]:
```
arr=np.arange(12).reshape((3,4))
arr
```

Out[28]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [29]:
```
arr.ravel()
```

Out[29]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [30]:
```
arr.ravel('C')
```

Out[30]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

로우 우선: C순서 상위 차원을 우선 탐색한다. 1번 축을 0번 축보다 우선 탐색한다.

In [31]:
```
arr.ravel('F')
```

Out[31]:
```
array([ 0,  4,  8,  1,  5,  9,  2,  6, 10,  3,  7, 11])
```

컬럼 우선: 포트란 순서 상위 차원을 나중에 탐색한다. 0번 축을 1번 축보다 우선 탐색한다.

A.2.3 배열 이어 붙이고 나누기

In [32]:
```
arr1=np.array([[1,2,3],[4,5,6]])
arr1
```

Out[32]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

concatenate는 매열의 목록(튜플, 리스트등)을 받아서 주어진 축에 따라서 하나의 배열로 합쳐준다

In [33]:
```
arr2=np.array([[7,8,9],[10,11,12]])
arr2
```

Out[33]:
```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

In [34]:
```
np.concatenate([arr1,arr2],axis=0)
```

Out[34]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [35]:
```
np.concatenate([arr1,arr2],axis=1)
```

Out[35]:
```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

A.2.3 배열 이어 붙이고 나누기

vstack, hstack메서트를 사용해서 쉽게 할 수도 있음

In [36]:
```
np.vstack((arr1,arr2))
```

Out[36]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [37]:
```
np.hstack((arr1,arr2))
```

Out[37]:
```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

A.2.3 배열 이어 붙이고 나누기

In [38]:
```
arr=np.random.randn(5,2)
arr
```

Out[38]:
```
array([[-0.52224927,  0.81850932],
       [ 0.59269854,  1.76305625],
       [ 1.28943228,  1.34497584],
       [ 0.62533562, -1.65389657],
       [-0.1085907 , -0.99068988]])
```

In [39]:
```
first,second,third=np.split(arr,[1,3])
```

[1,3]은 배열을 나눌 대 기준이 되는 위치를 나타낸다.

In [40]:
```
first
```

Out[40]:
```
array([[-0.52224927,  0.81850932]])
```

In [41]:
```
second
```

Out[41]:
```
array([[0.59269854, 1.76305625],
       [1.28943228, 1.34497584]])
```

In [42]:
```
third
```

Out[42]:
```
array([[ 0.62533562, -1.65389657],
       [-0.1085907 , -0.99068988]])
```

A.2.3 배열 이어 붙이고 나누기

In [43]:
```
arr=np.arange(6)
arr
```

Out[43]:
```
array([0, 1, 2, 3, 4, 5])
```

In [44]:
```
arr1=arr.reshape((3,2))
arr1
```

Out[44]:
```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

In [45]:
```
arr2=np.random.randn(3,2)
arr2
```

Out[45]:
```
array([[ 1.39942999,  1.37051122],
       [-0.31084685, -1.49108117],
       [ 1.07936522, -0.06299435]])
```

In [46]:
```
np.r_[arr1,arr2]
```

Out[46]:
```
array([[ 0.        ,  1.        ],
       [ 2.        ,  3.        ],
       [ 4.        ,  5.        ],
       [ 1.39942999,  1.37051122],
       [-0.31084685, -1.49108117],
       [ 1.07936522, -0.06299435]])
```

In [47]:
```
np.c_[np.r_[arr1,arr2],arr]
```

Out[47]:
```
array([[ 0.        ,  1.        ,  0.        ],
       [ 2.        ,  3.        ,  1.        ],
       [ 4.        ,  5.        ,  2.        ],
       [ 1.39942999,  1.37051122,  3.        ],
       [-0.31084685, -1.49108117,  4.        ],
       [ 1.07936522, -0.06299435,  5.        ]])
```

In [48]:
```
np.c_[1:6,-10:-5]
# 슬라이스를 배열로 변환해준다.
```

Out[48]:
```
array([[  1, -10],
       [  2,  -9],
       [  3,  -8],
       [  4,  -7],
       [  5,  -6]])
```

A.2.4 원소 반복하기: repeat 과 tile

```
In [49]: arr=np.arange(3)
         arr
Out[49]: array([0, 1, 2])

In [50]: arr.repeat(3)
Out[50]: array([0, 0, 0, 1, 1, 1, 2, 2, 2])
```

기본적으로 정수를 넘기면 각 배열은 그 수 만큼 반복된다.

```
In [51]: arr.repeat([2,3,4])
Out[51]: array([0, 0, 1, 1, 1, 2, 2, 2, 2])
```

정수의 배열을 넘긴다면 각 원손는 배열에 담긴 정수만큼 반복

```
In [52]: # 다차원 배열의 경우 특정 축을 따라 각 원소가 반복된다.
         arr=np.random.randn(2,2)
         arr
Out[52]: array([[ 0.47606022, -0.37203466],
                [ 0.68152238,  0.0905015 ]])

In [53]: arr.repeat(2,axis=0)
Out[53]: array([[ 0.47606022, -0.37203466],
                [ 0.47606022, -0.37203466],
                [ 0.68152238,  0.0905015 ],
                [ 0.68152238,  0.0905015 ]])

In [54]: arr.repeat([2,3],axis=0)
Out[54]: array([[ 0.47606022, -0.37203466],
                [ 0.47606022, -0.37203466],
                [ 0.68152238,  0.0905015 ],
                [ 0.68152238,  0.0905015 ],
                [ 0.68152238,  0.0905015 ]])
```

A.2.4 원소 반복하기: repeat 과 tile

In [55]:
```
arr.repeat([2,3],axis=1)
```

Out[55]:
```
array([[ 0.47606022,  0.47606022, -0.37203466, -0.372034
66, -0.37203466],
       [ 0.68152238,  0.68152238,  0.0905015 ,  0.090501
5 ,  0.0905015 ]])
```

In [56]:
```
arr.repeat(2)
# axis인지를 넘기지 않으면 평탄화 된다.
```

Out[56]:
```
array([ 0.47606022,  0.47606022, -0.37203466, -0.3720346
6,  0.68152238,
        0.68152238,  0.0905015 ,  0.0905015 ])
```

A.2.4 원소 반복하기: repeat 과 tile

In [58]:
```
np.tile(arr,2)
# tile메서드는 축을 따라서 배열을 복사해서 쌓는 함수
```

Out[58]:
```
array([[ 0.47606022, -0.37203466,  0.47606022, -0.37203466],
       [ 0.68152238,  0.0905015 ,  0.68152238,  0.0905015 ]])
```

In [59]:
```
np.tile(arr,(2,1))
```

Out[59]:
```
array([[ 0.47606022, -0.37203466],
       [ 0.68152238,  0.0905015 ],
       [ 0.47606022, -0.37203466],
       [ 0.68152238,  0.0905015 ]])
```

tile메서드의 두 번째 인자는 타일을 이어 붙일 모양을 나타내는 튜플이 될 수 있다.

A.2.5 팬시 색인 take과 put

In [61]:
```
arr=np.arange(10)*100
arr
```

Out[61]:
```
array([  0, 100, 200, 300, 400, 500, 600, 700, 800, 90
0])
```

In [62]:
```
inds=[7,1,2,6]
```

In [63]:
```
arr[inds]
```

Out[63]:
```
array([700, 100, 200, 600])
```

정수 배열을 사용한 팬시 색인 기능을 통해 배열의 일부값을 지정하거나 가져올 수 있다.

In [64]:
```
arr.take(inds)
```

Out[64]:
```
array([700, 100, 200, 600])
```

A.2.5 팬시 색인 take과 put

In [65]:
```
arr.put(inds,42)
arr
# inds 부분에 42를 넣음
```

Out[65]:
```
array([  0,  42,  42, 300, 400, 500,  42,  42, 800, 900])
```

In [66]:
```
arr.put(inds,[40,41,42,43])
arr
# inds 부분에 차례로 넣음
```

Out[66]:
```
array([  0,  41,  42, 300, 400, 500,  43,  40, 800, 900])
```

A.3 브로드캐스팅

```
In [70]: arr=np.arange(5)
         arr
Out[70]: array([0, 1, 2, 3, 4])

In [71]: arr*4
Out[71]: array([ 0,  4,  8, 12, 16])
```

여기서 스칼라 값 4는 곱셈 연산 과정에서 배열의 모든 원소로 브로드캐스트 됐다

브로드 캐스팅 규칙

만일 이어지는 각 차원(시작과 끝까지)에 대해 축의 길이가 일치하거나 둘 중 하나의 길이가 1이라면 두 배열은 브로드캐스팅 호환이다. 브로드캐스팅은 누락된 혹은 길이가 1인 차원에 대해 수행된다.

A.3.1 다른 축에 대해 브로드캐스팅 하기

```
In [82]: arr

Out[82]: array([[0.15272933, 0.46795312, 0.06512914],
                [0.89228971, 0.11555545, 0.01238624],
                [0.74775293, 0.94823709, 0.55490903],
                [0.63417543, 0.27608015, 0.83996631]])

In [83]: arr.mean(1)

Out[83]: array([0.22860387, 0.34007713, 0.75029969, 0.5834073 ])

In [135]: arr-arr.mean(1)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-135-b5e089451669> in <module>
----> 1 arr-arr.mean(1)

ValueError: operands could not be broadcast together with shapes (4,5) (4,)
```

A.3.1 다른 축에 대해 브로드캐스팅 하기

```
In [85]: arr.mean(1).reshape((4,1))

Out[85]: array([[0.22860387],
                [0.34007713],
                [0.75029969],
                [0.5834073 ]])
```

```
In [86]: arr-arr.mean(1).reshape((4,1))

Out[86]: array([[-0.07587454,  0.23934926, -0.16347472],
                [ 0.55221258, -0.22452169, -0.32769089],
                [-0.00254676,  0.19793741, -0.19539065],
                [ 0.05076813, -0.30732715,  0.25655902]])
```

A.3.2 브로드캐스팅을 이용해서 배열에 값 대입하기

```
In [97]: arr=np.zeros((4,3))
         arr

Out[97]: array([[0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.]])
```

```
In [98]: arr[:]=5
         arr

Out[98]: array([[5., 5., 5.],
                [5., 5., 5.],
                [5., 5., 5.],
                [5., 5., 5.]])
```

```
In [99]: col=np.array([1.28,-0.42,0.44,1.6])
         col

Out[99]: array([ 1.28, -0.42,  0.44,  1.6 ])
```

```
In [100]: arr[:]=col[:,np.newaxis]
          arr

Out[100]: array([[ 1.28,  1.28,  1.28],
                 [-0.42, -0.42, -0.42],
                 [ 0.44,  0.44,  0.44],
                 [ 1.6 ,  1.6 ,  1.6 ]])
```

```
In [101]: arr[:2]=[[-1.37],[0.509]]
          arr

Out[101]: array([[-1.37 , -1.37 , -1.37 ],
                 [ 0.509,  0.509,  0.509],
                 [ 0.44 ,  0.44 ,  0.44 ],
                 [ 1.6  ,  1.6  ,  1.6  ]])
```

A.4 고급 ufunc 사용법

A.4.1 브로드캐스팅을 이용해서 배열에 값 대입하기 ufunc 인스턴스 메서드

In [102]:
```
arr=np.arange(10)
arr
```

Out[102]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [103]:
```
np.add.reduce(arr)
```

Out[103]:
```
45
```

reduce는 하나의 배열을 받아서 순차적인 이항 연산을 통해 축에 따라 그 값을 집계해준다.

In [104]:
```
arr.sum()
```

Out[104]:
```
45
```

In [105]:
```
np.random.seed(12346)
```

동일한 난수 발생을 위해 시드값 직접 지정

In [106]:
```
arr=np.random.randn(5,5)
```

In [107]:
```
arr[::2].sort(1) # 일부 로우를 정렬
```

In [108]:
```
arr[:,:-1]<arr[:,1:]
```

Out[108]:
```
array([[ True,  True,  True,  True],
       [False,  True, False, False],
       [ True,  True,  True,  True],
       [ True, False,  True,  True],
       [ True,  True,  True,  True]])
```

A.4.1 브로드캐스팅을 이용해서 배열에 값 대입하기 ufunc 인스턴스 메서드

In [116]:
```
np.logical_and.reduce(arr[:,:-1]<arr[:,1:],axis=1)
```

Out[116]:
```
array([ True, False,  True, False,  True])
```

In [117]:
```
arr=np.arange(15).reshape((3,5))
arr
```

Out[117]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

In [118]:
```
np.add.accumulate(arr,axis=1)
# 누계
```

Out[118]:
```
array([[ 0,  1,  3,  6, 10],
       [ 5, 11, 18, 26, 35],
       [10, 21, 33, 46, 60]], dtype=int32)
```

In [119]:
```
arr=np.arange(3).repeat([1,2,2])
arr
```

Out[119]:
```
array([0, 1, 1, 2, 2])
```

In [120]:
```
np.multiply.outer(arr,np.arange(5))
# oute메서드는 두 배열간의 외적을 계산한다.
```

Out[120]:
```
array([[0, 0, 0, 0, 0],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 2, 4, 6, 8],
       [0, 2, 4, 6, 8]])
```

A.4.2 파이썬으로 사용자 정의 ufunc 정의 하기

```
In [129]: def add_elements(x,y):
              return x+y

In [130]: add_them=np.frompyfunc(add_elements,2,1)

In [131]: add_them(np.arange(8),np.arange(8))
Out[131]: array([0, 2, 4, 6, 8, 10, 12, 14], dtype=object)
```

frompyfunc을 이용해서 생성한 함수는 파이선 객체가 담긴 배열을 반환

```
In [132]: add_them=np.vectorize(add_elements,otypes=[np.float64])

In [133]: add_them(np.arange(8),np.arange(8))
Out[133]: array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14.])
```

Thank you!