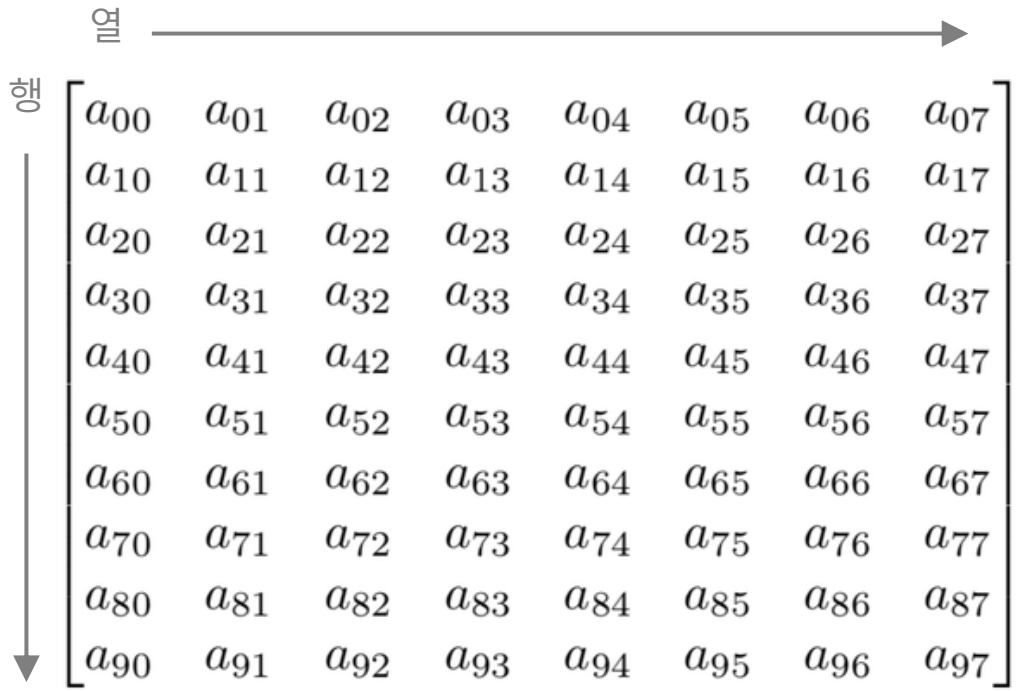
Numpy

Numpy

- 2차원 (배열)과 같이, 다차원 배열을 다룹니다.
- 각 차원의 순서 번호의 시작은 0 입니다.

numpy → 배열을 다루는 도구



Numpy

- “Numerical Python” 다차원 데이터를 쉽게 처리할 수 있게 도와주는 패키지
- 다차원 배열, 행렬의 생성과 연산, 정렬 등 편리한 기능들을 많이 포함하고 있음

1차원 배열:

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

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

2차원 배열:

```
np.array([[1,2,3,4],[11,12,13,14]])
```

```
array([[ 1,  2,  3,  4],
       [11, 12, 13, 14]])
```

3차원 배열:

```
np.array([[[1,2,3,4],[11,12,13,14]],[[21,22,23,24],[31,32,33,34]]])
```

```
array([[[ 1,  2,  3,  4],
        [11, 12, 13, 14]],

       [[21, 22, 23, 24],
        [31, 32, 33, 34]]])
```

numpy 기본 1. 다차원 배열 만들기

- numpy 임포트

```
import numpy as np
```

- numpy 배열 생성

np.array(인자1_리스트)

리스트로 넘파이 배열을 만드는 함수
리스트가 중첩된 깊이가 곧 배열의 차원이 됨

[인자 설명]
- 인자1_리스트: 가장 안쪽의 []가 넘파이 배열에서 가장 마지막 차원이 된다.

```
# 1차원 배열 생성하기
넘파이_배열1 = np.array([1,2,3,4])
넘파이_배열1
```

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

1	2	3	4

numpy 기본 1. 다차원 배열 만들기

- 리스트로 배열 생성

```
# 2차원 배열 생성하기
넘파이_배열2 = np.array([[1,2,3,4],[11,12,13,14]])
넘파이_배열2
```

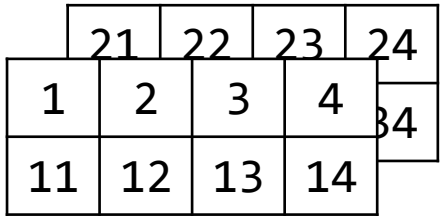
```
array([[ 1,  2,  3,  4],
       [11, 12, 13, 14]])
```

1	2	3	4
11	12	13	14

```
# 3차원 배열 생성하기
넘파이_배열3 = np.array([ [[1,2,3,4],[11,12,13,14]],
                          [[21,22,23,24],[31,32,33,34]] ])
넘파이_배열3
```

```
array([[[ 1,  2,  3,  4],
        [11, 12, 13, 14]],

       [[21, 22, 23, 24],
        [31, 32, 33, 34]]])
```



numpy 기본 1. 다차원 배열 만들기

- .shape 로 배열의 모양 보기

```
# 배열의 모양 보기
print(넘파이_배열1.shape)
print(넘파이_배열2.shape)
print(넘파이_배열3.shape)
```

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

numpy 기본 1. 다차원 배열 만들기

- 초기화 함수로 넘파이 배열 생성

```
z1 = np.zeros([4])    # 0 으로 채운 배열 만들기
o2 = np.ones([3,4])   # 1 로 채운 배열 만들기
r2 = np.random.randn(2,3)   # 랜덤한 숫자 배열 만들기
```

```
z1, o2, r2
```

```
(array([0., 0., 0., 0.]),
 array([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]),
 array([[ 1.37200043, -0.35335282, -0.18658457],
        [ 1.82544   ,  0.84035419, -0.35604833]]))
```

```
# 배열의 모양 보기
print(z1.shape)
print(o2.shape)
print(r2.shape)
```

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

numpy 기본 1. 다차원 배열 만들기

```
np.random.randn( d1, d2, d3, d4, … )
```

- 초기화 함수로 넘파이 배열 생성

```
# 무척 큰 4차원 배열 생성해보기
r4 = np.random.randn(2,2,4,5).round(3)
r4
```

```
array([[[[-1.101,  0.079,  0.205,  2.346,  0.669],
         [ 0.677,  1.141, -1.377, -1.571,  0.169],
         [-0.585, -0.452,  0.238, -1.564,  1.686],
         [ 1.694, -0.216, -0.564,  0.384,  0.129]],

        [[ 0.161,  0.297,  0.596,  1.314, -0.412],
         [ 0.713,  2.157,  1.426, -0.355, -0.313],
         [-0.227,  1.946, -1.508, -0.062, -0.5  ],
         [-1.319, -1.201, -1.03 , -0.49 , -1.089]]],


       [[[ 3.403,  0.308,  0.706,  0.235, -0.502],
         [ 0.702,  0.098,  0.107,  0.411,  0.144],
         [-1.382,  1.464,  1.968, -0.406, -1.506],
         [-0.359,  0.55 , -1.35 , -2.287, -0.483]],

        [[-0.58 , -1.597, -0.312, -1.778,  1.725],
         [-0.652,  0.756, -1.225, -1.987, -0.143],
         [ 0.619,  1.436,  0.242,  0.823, -1.976],
         [-1.239,  1.14 , -0.799, -0.858, -0.366]]]])
```

[0][][][]

[1][0][][]

4×5

[1][1][][]

[1][][][]

numpy 기본 2. Indexing

- Indexing으로 넘파이 배열의 일부 추출하기

```
my_np_arr[가져올 번호][가져올번호]…
```

넘파이 배열의 일부를 가져오는 코드

[ ] 대괄호 하나 당 하나의 차원으로, 높은 차원부터 접근한다.
첫번째 원소의 번호는 0, 두번째 원소는 1, …

```
# r4 배열에서, [두번째][첫번째] 접근해서 가져오기
r4[1][0]
```

```
array([[ 3.403,  0.308,  0.706,  0.235, -0.502],
       [ 0.702,  0.098,  0.107,  0.411,  0.144],
       [-1.382,  1.464,  1.968, -0.406, -1.506],
       [-0.359,  0.55 , -1.35 , -2.287, -0.483]])
```

```
array([[[[-1.101,  0.079,  0.205,  2.346,  0.669],
         [ 0.677,  1.141, -1.377, -1.571,  0.169],
         [-0.585, -0.452,  0.238, -1.564,  1.686],
         [ 1.694, -0.216, -0.564,  0.384,  0.129]],

        [[ 0.161,  0.297,  0.596,  1.314, -0.412],
         [ 0.713,  2.157,  1.426, -0.355, -0.313],
         [-0.227,  1.946, -1.508, -0.062, -0.5  ],
         [-1.319, -1.201, -1.03 , -0.49 , -1.089]]],


       [[[ 3.403,  0.308,  0.706,  0.235, -0.502],
         [ 0.702,  0.098,  0.107,  0.411,  0.144],
         [-1.382,  1.464,  1.968, -0.406, -1.506],
         [-0.359,  0.55 , -1.35 , -2.287, -0.483]],

        [[-0.58 , -1.597, -0.312, -1.778,  1.725],
         [-0.652,  0.756, -1.225, -1.987, -0.143],
         [ 0.619,  1.436,  0.242,  0.823, -1.976],
         [-1.239,  1.14 , -0.799, -0.858, -0.366]]]])
```

r4

numpy 기본 2. Indexing

- Indexing으로 넘파이 배열의 일부 추출하기

```
# r4 배열에서, [두번째][첫번째] 접근해서 가져오기
r4[1][0]
```

```
array([[ 3.403,  0.308,  0.706,  0.235, -0.502],
       [ 0.702,  0.098,  0.107,  0.411,  0.144],
       [-1.382,  1.464,  1.968, -0.406, -1.506],
       [-0.359,  0.55 , -1.35 , -2.287, -0.483]])
```

```
r4[1][0][2]
```

```
array([-1.382,  1.464,  1.968, -0.406, -1.506])
```

```
r4[1][0][2][4]
```

```
-1.506
```

```
r4[1,0,2,4]   # 대괄호 대신 쉼표 , 도 차원 접근이 가능
```

```
-1.506
```

```
array([[[[-1.101,  0.079,  0.205,  2.346,  0.669],
         [ 0.677,  1.141, -1.377, -1.571,  0.169],
         [-0.585, -0.452,  0.238, -1.564,  1.686],
         [ 1.694, -0.216, -0.564,  0.384,  0.129]],

        [[ 0.161,  0.297,  0.596,  1.314, -0.412],
         [ 0.713,  2.157,  1.426, -0.355, -0.313],
         [-0.227,  1.946, -1.508, -0.062, -0.5  ],
         [-1.319, -1.201, -1.03 , -0.49 , -1.089]]],


       [[[ 3.403,  0.308,  0.706,  0.235, -0.502],
         [ 0.702,  0.098,  0.107,  0.411,  0.144],
         [-1.382,  1.464,  1.968, -0.406, -1.506],
         [-0.359,  0.55 , -1.35 , -2.287, -0.483]],

        [[-0.58 , -1.597, -0.312, -1.778,  1.725],
         [-0.652,  0.756, -1.225, -1.987, -0.143],
         [ 0.619,  1.436,  0.242,  0.823, -1.976],
         [-1.239,  1.14 , -0.799, -0.858, -0.366]]]])
```

r4

numpy 기본 2. Slicing

- Slicing으로 넘파이 배열의 일부 추출하기

```
my_np_arr[시작:끝:간격][시작:끝:간격]…
```

넘파이 배열의 여러 개의 원소를 가져오는 코드

[ ] 대괄호 당, 각 차원에서 [시작, 시작+간격, 시작+2간격, … , 끝-간격] 에 해당하는 원소를 가져온다. (생략 시 기본값, 시작:0, 끝: 길이, 간격:1)
가져오는 **마지막 원소가 끝을 포함하지 않는 것**을 주의하자

```
r4[1][0][2]
```

```
array([-1.382,  1.464,  1.968, -0.406, -1.506])
```

0 1 2 3

```
r4[1][0][2][0:4:2]
```

```
array([-1.382,  1.968])
```

```
array([[[[-1.101,  0.079,  0.205,  2.346,  0.669],
         [ 0.677,  1.141, -1.377, -1.571,  0.169],
         [-0.585, -0.452,  0.238, -1.564,  1.686],
         [ 1.694, -0.216, -0.564,  0.384,  0.129]],

        [[ 0.161,  0.297,  0.596,  1.314, -0.412],
         [ 0.713,  2.157,  1.426, -0.355, -0.313],
         [-0.227,  1.946, -1.508, -0.062, -0.5  ],
         [-1.319, -1.201, -1.03 , -0.49 , -1.089]]],


       [[[ 3.403,  0.308,  0.706,  0.235, -0.502],
         [ 0.702,  0.098,  0.107,  0.411,  0.144],
         [-1.382,  1.464,  1.968, -0.406, -1.506],
         [-0.359,  0.55 , -1.35 , -2.287, -0.483]],

        [[-0.58 , -1.597, -0.312, -1.778,  1.725],
         [-0.652,  0.756, -1.225, -1.987, -0.143],
         [ 0.619,  1.436,  0.242,  0.823, -1.976],
         [-1.239,  1.14 , -0.799, -0.858, -0.366]]]])
```

r4

numpy 기본 2. Slicing

- Slicing으로 넘파이 배열의 일부 추출하기

```
r4[1][0][1:4:2]
```

```
array([[ 0.702,  0.098,  0.107,  0.411,  0.144],
       [-0.359,  0.55 , -1.35 , -2.287, -0.483]])
```

```
# Indexing과 Slicing 같이 적용하기
r4[1][0][1:4:2][1][3:]
```

```
array([-2.287, -0.483])
```

```
r4[1,0][1:4:2][1,3:]   # 모두 다 짬뽕!!
```

```
array([-2.287, -0.483])
```

```
array([[[[-1.101,  0.079,  0.205,  2.346,  0.669],
         [ 0.677,  1.141, -1.377, -1.571,  0.169],
         [-0.585, -0.452,  0.238, -1.564,  1.686],
         [ 1.694, -0.216, -0.564,  0.384,  0.129]],

        [[ 0.161,  0.297,  0.596,  1.314, -0.412],
         [ 0.713,  2.157,  1.426, -0.355, -0.313],
         [-0.227,  1.946, -1.508, -0.062, -0.5  ],
         [-1.319, -1.201, -1.03 , -0.49 , -1.089]]],


       [[[ 3.403,  0.308,  0.706,  0.235, -0.502],
         [ 0.702,  0.098,  0.107,  0.411,  0.144],
         [-1.382,  1.464,  1.968, -0.406, -1.506],
         [-0.359,  0.55 , -1.35 , -2.287, -0.483]],

        [[-0.58 , -1.597, -0.312, -1.778,  1.725],
         [-0.652,  0.756, -1.225, -1.987, -0.143],
         [ 0.619,  1.436,  0.242,  0.823, -1.976],
         [-1.239,  1.14 , -0.799, -0.858, -0.366]]]])
```

r4

numpy 기본 3. 배열 간 연산

- 넘파이 배열 간 사칙연산
  - 동일한 모양의 두 배열을 사칙연산하면, 각 원소끼리 해당 사칙연산이 적용됩니다.

```
arr = np.array([[1,2,3,4],[10,11,12,13]])
arr
```

```
array([[ 1,  2,  3,  4],
       [10, 11, 12, 13]])
```

```
arr + arr    # 배열끼리 연산하기 (+ - */ 모두 가능)
```

```
array([[ 2,  4,  6,  8],
       [20, 22, 24, 26]])
```

numpy 기본 3. Broadcasting

```
array([[ 1,  2,  3,  4],
       [10, 11, 12, 13]])
```

arr

Broadcasting 연산 (Scalar)

- 다차원 넘파이 배열과 하나의 숫자를 사칙 연산 하는 경우,
  넘파이 배열의 모든 원소에 해당 숫자 사칙 연산이 확장되어 적용됨

```
arr + 20      # 덧셈 브로드캐스팅
```

```
array([[21, 22, 23, 24],
       [30, 31, 32, 33]])
```

```
arr * 100     # 곱셈 브로드캐스팅
```

```
array([[ 100,  200,  300,  400],
       [1000, 1100, 1200, 1300]])
```

numpy 기본 3. Aggregation

```
array([[ 1,  2,  3,  4],
       [10, 11, 12, 13]])
```

arr

Aggregation (집계)

- 넘파이 배열에 대해 모든 원소를 집계하는 연산
- 수학: sum, mean, prod / 최대 최소: max, min, argmax, argmin
- axis 인자를 주어, 특정 축으로 집계할 수 있다

```
# 배열 전체의 덧셈, 평균, 곱셈, 최댓값, 최솟값
arr.sum(), arr.mean(), arr.prod(), arr.max(), arr.min()
```

```
(56, 7.0, 411840, 13, 1)
```

```
# 배열에서의 최대인 원소의 번호, 최소인 원소의 번호
arr.argmax(), arr[1].argmax(), arr.argmin()
```

```
(7, 3, 0)
```

numpy 기본 3. Aggregation

Aggregation (집계)

```
array([[ 1,  2,  3,  4],
       [10, 11, 12, 13]])
```

arr

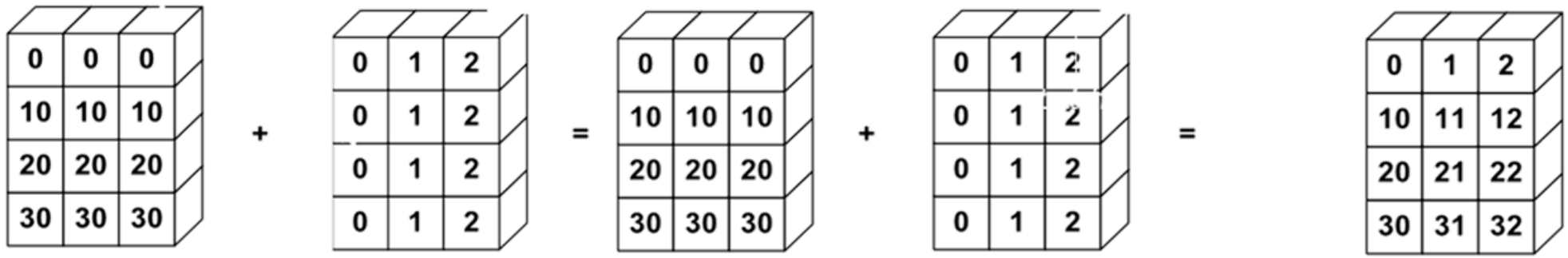
```
# 특정 축(차원)으로 덧셈 수행하기
arr.sum(axis=0), arr.sum(axis=1)
```

```
(array([11, 13, 15, 17]), array([10, 46]))
```

```
# Indexing + Slicing + Aggregation 응용
arr[1].min(), arr[1][2:].sum()
```

```
(10, 25)
```

numpy 기본 4. 고급 Broadcasting



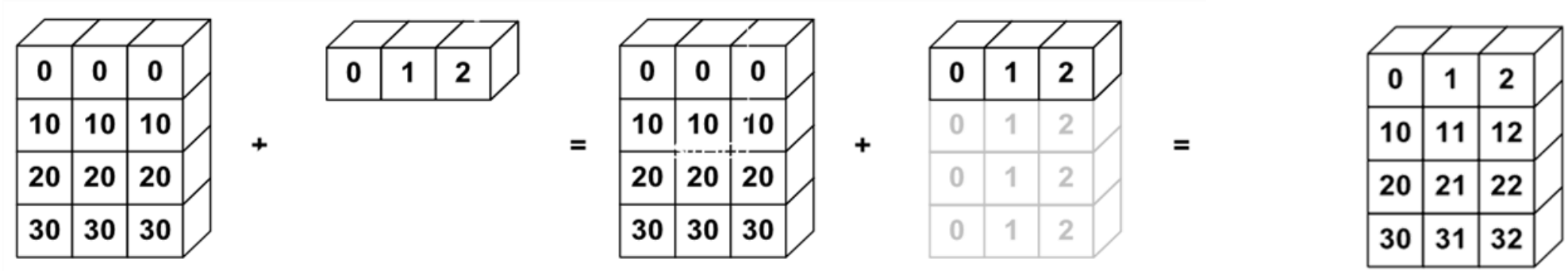

```
arr1 = np.array([[0,0,0],[10,10,10],[20,20,20],[30,30,30]])
arr2 = np.array([[0,1,2],[0,1,2],[0,1,2],[0,1,2]])
arr1.shape, arr2.shape
```

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

```
arr1 + arr2          # (4x3)  +  (4x3)
```

```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

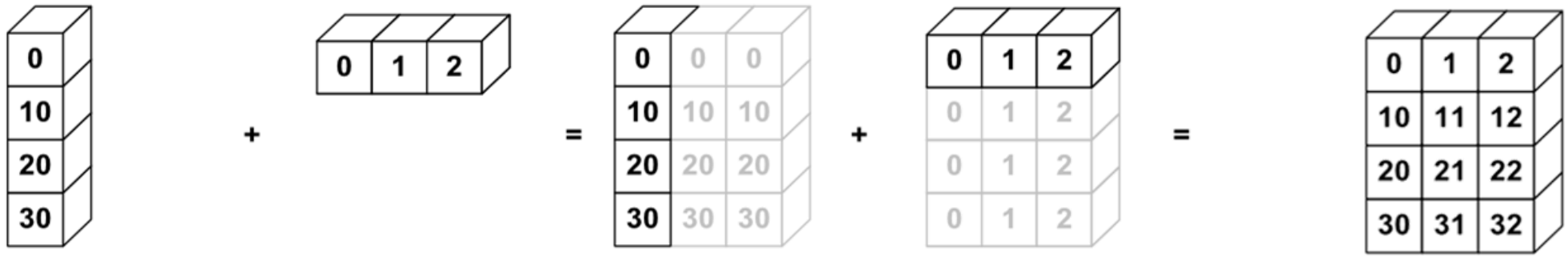
numpy 기본 4. 고급 Broadcasting




```
arr1 + np.array([0,1,2])      # (4x3) + (1x3)
```

```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

numpy 기본 4. 고급 Broadcasting




```
#  (4x1) + (1x3)
np.array([[0],[10],[20],[30]]) + np.array([0,1,2])
```

```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

Q & A

THANK YOU :)