파이썬 라이브러리를 활용한 데이터 분석

3장 numpy 기본: 배열과 벡터 연산

배열 결합: concatenate

- **배열 합치기, 기본은 0 축(세로)으로**
  - np.concatenate((a1, a2, ...), axis=0)
  - a1, a2.....: 배열

In [2]:
```
# 데모 배열
a = np.arange(1, 7).reshape((2, 3))
pprint(a)
b = np.arange(7, 13).reshape((2, 3))
pprint(b)
```

```
shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[1 2 3]
[4 5 6]]
shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[ 7  8  9]
[10 11 12]]
```

In [3]:
```
# axis=0 방향으로 두 배열 결합, axis 기본값=0
result = np.concatenate((a, b))
result
```

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

In [4]:
```
# axis=0 방향으로 두 배열 결합, 결과 동일
result = np.concatenate((a, b), axis=0)
result
```

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

In [5]:
```
# axis=1 방향으로 두 배열 결합, 결과 동일
result = np.concatenate((a, b), axis=1)
result
```

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

배열 결합: vstack

- **수직 방향 배열 결합**
- **np.vstack(tup)**
  - tup: 튜플
  - 튜플로 설정된 여러 배열을 수직 방향으로 연결 (axis=0 방향, 세로)
  - np.concatenate(tup, axis=0)와 동일

```
In [6]: # 데모 배열
        a = np.arange(1, 7).reshape((2, 3))
        pprint(a)
        b = np.arange(7, 13).reshape((2, 3))
        pprint(b)

shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[1 2 3]
[4 5 6]]
shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[ 7  8  9]
[10 11 12]]

In [7]: np.vstack((a, b))

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

In [8]: # 4개 배열을 튜플로 설정
        np.vstack((a, b, a, b))

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

배열 결합: hstack

- **수평 방향 배열 결합**
- **np.hstack(tup)**
  - tup: 튜플
  - 튜플로 설정된 여러 배열을 수평 방향으로 연결 (axis=1 방향, 가로)
  - np.concatenate(tup, axis=1)와 동일

```
In [9]: # 데모 배열
        a = np.arange(1, 7).reshape((2, 3))
        pprint(a)
        b = np.arange(7, 13).reshape((2, 3))
        pprint(b)

shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[1 2 3]
[4 5 6]]
shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[ 7  8  9]
[10 11 12]]

In [10]: np.hstack((a, b))

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

In [11]: np.hstack((a, b, a, b))

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

배열 분리: hsplit

- **np.hsplit(ary, indices_or_sections)**
  - 지정한 배열을 수평(행) 방향으로 분할

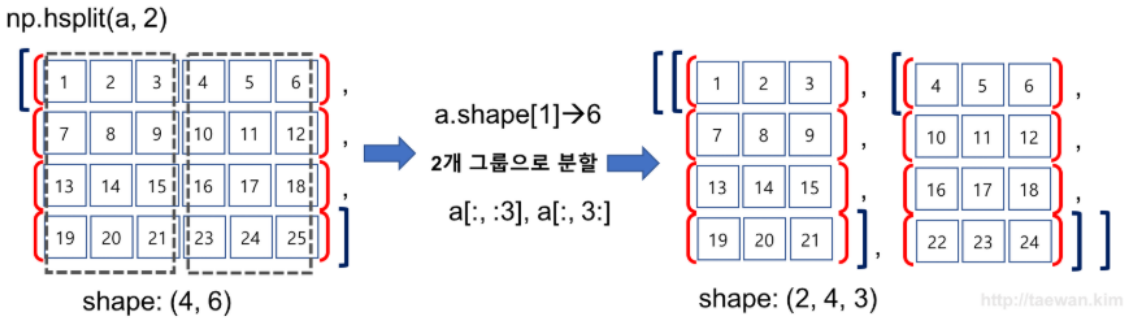
배열 분리

- **hsplit**
  - 결과는 배열의 리스트

In [2]:
```
# 분할 대상 배열 생성
a = np.arange(1, 25).reshape((4, 6))
pprint(a)
```

```
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]]
```

수평 방향으로 배열을 두 그룹으로 분할

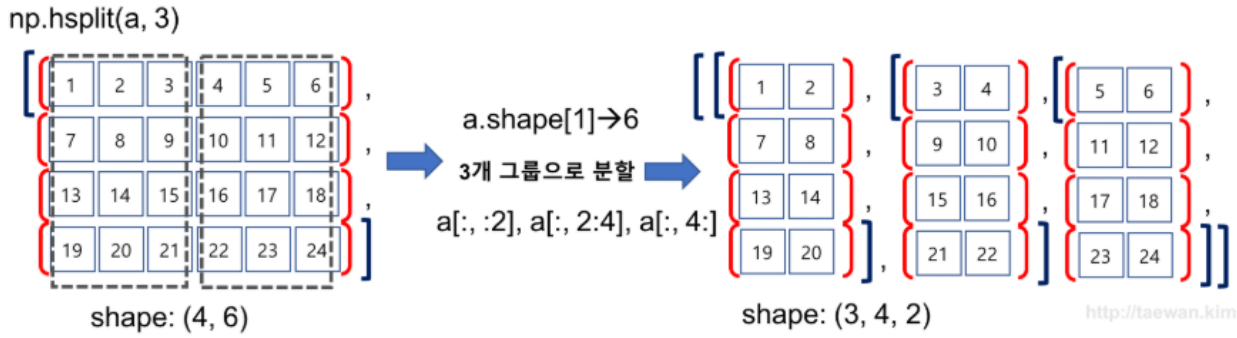


In [3]:
```
# 수평으로 두 그룹으로 분할하는 함수
result = np.hsplit(a, 2)
result
```

Out[3]:
```
[array([[ 1,  2,  3],
        [ 7,  8,  9],
        [13, 14, 15],
        [19, 20, 21]]), array([[ 4,  5,  6],
        [10, 11, 12],
        [16, 17, 18],
        [22, 23, 24]])]
```

배열 분리: hsplit

수평 방향으로 배열을 세 그룹으로 분할



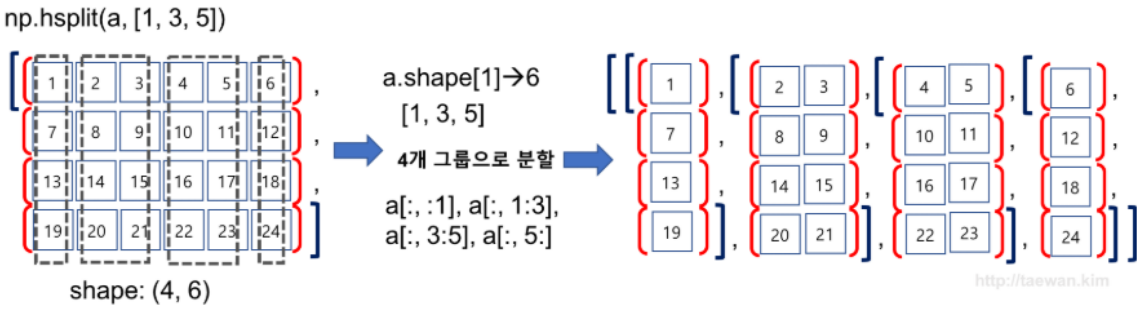
In [4]:
```
# 수평으로 두 그룹으로 분할하는 함수
result = np.hsplit(a, 3)
result
```

Out[4]:
```
[array([[ 1,  2],
        [ 7,  8],
        [13, 14],
        [19, 20]]), array([[ 3,  4],
        [ 9, 10],
        [15, 16],
        [21, 22]]), array([[ 5,  6],
        [11, 12],
        [17, 18],
        [23, 24]])]
```

배열 분리: hsplit

수평 방향으로 여러 구간으로 구분

- np.hsplit의 두 번째 파라미터에 구간 설정 배열을 전달하여 여러 배열로 구분합니다.




```
In [5]: np.hsplit(a, [1, 3 ,5])
```

```
Out[5]: [array([[ 1],
                [ 7],
                [13],
                [19]]), array([[ 2,  3],
                [ 8,  9],
                [14, 15],
                [20, 21]]), array([[ 4,  5],
                [10, 11],
                [16, 17],
                [22, 23]]), array([[ 6],
                [12],
                [18],
                [24]])]
```

배열 분리: vsplit

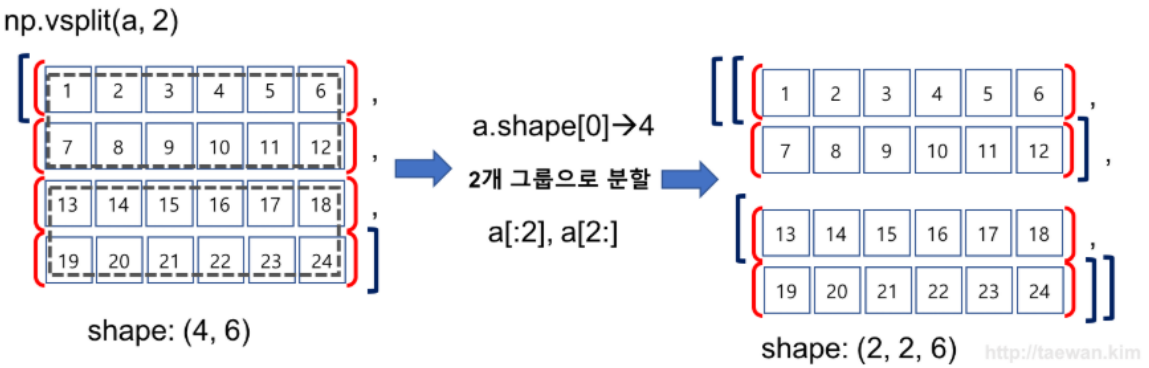
- **배열을 수직 방향(행 방향)으로 분할하는 함수**
  - np.vsplit(ary, indices_or_sections)

배열 분리:

In [6]:
```
# 분할 대상 배열 생성
a = np.arange(1, 25).reshape((4, 6))
pprint(a)
```

```
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]]
```

수직 방향으로 배열을 두 개 그룹으로 분할



In [7]:
```
result=np.vsplit(a, 2)
result
```

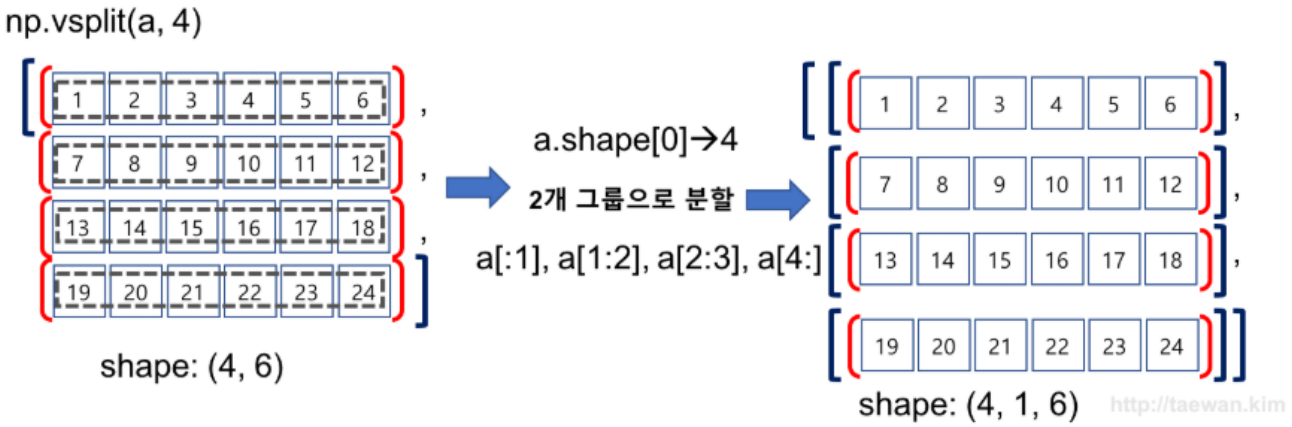
Out[7]:
```
[array([[ 1,  2,  3,  4,  5,  6],
        [ 7,  8,  9, 10, 11, 12]]), array([[13, 14, 15, 16, 17, 18],
        [19, 20, 21, 22, 23, 24]])]
```

In [8]:
```
np.array(result).shape
```

Out[8]:
```
(2, 2, 6)
```

배열 분리: vsplit

수직 방향으로 배열을 4 개 그룹으로 분할




```
In [9]: result=np.vsplit(a, 4)
        result
```

```
Out[9]: [array([[1, 2, 3, 4, 5, 6]]),
         array([[ 7,  8,  9, 10, 11, 12]]),
         array([[13, 14, 15, 16, 17, 18]]),
         array([[19, 20, 21, 22, 23, 24]])]
```

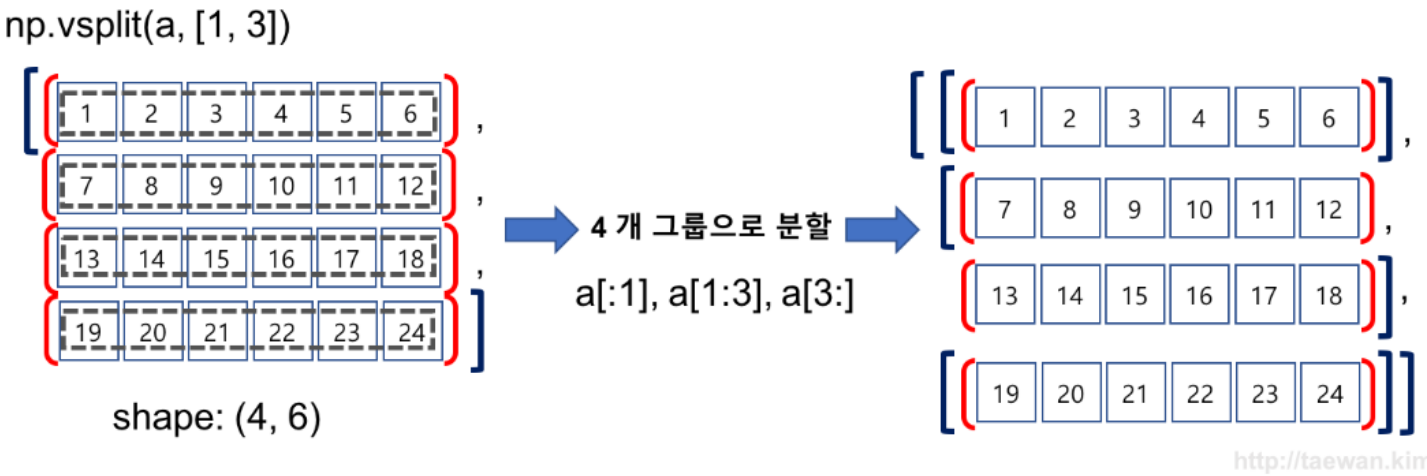
```
In [10]: np.array(result).shape
```

```
Out[10]: (4, 1, 6)
```

배열 분리: vsplit

수직 방향으로 여러 구간으로 구분

- np.hsplit의 두 번째 파라미터에 구간 설정 배열을 전달하여 여러 배열로 구분합니다.




```
In [11]: # row를 1, 2-3, 4번째 라인으로 구분
         np.vsplit(a, [1, 3])

Out[11]: [array([[1, 2, 3, 4, 5, 6]]), array([[ 7,  8,  9, 10, 11, 12],
                 [13, 14, 15, 16, 17, 18]]), array([[19, 20, 21, 22, 23, 24]])]
```

행렬 곱(내적)

- np.dot(a, b)
- a.dot(b)

A B A * B

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 6 & 3 \\ 5 & 2 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 1*6+2*5+3*4 & 1*3+2*2+3*1 \\ 4*6+5*5+6*4 & 4*3+5*2+6*1 \end{pmatrix}$$

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

$$C_{ij} = \sum_k A_{ik} B_{kj} = A_{ik} B_{kj}$$

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