

02_Numpy 활용

• Reshape

- 배열의 shape을 변형해주는 함수
- 배열의 요소 개수를 유지하며 형태만 변경
- 변형 전 size와 변형 후 size가 동일해야 함

> array.reshape(size)

```
arr = np.array([[1, 2, 3], [4, 5, 6]]) # 2x3 형태의 2차원 배열  
print(arr.reshape(3,2)) # 3x2 형태의 배열로 변형
```

```
Out : [[1 2]  
       [3 4]  
       [5 6]]
```

Numpy 활용

- Reshape

> 6 x 1 배열로 변형

```
print(arr.reshape(6, 1))
```

Out : [[1]
[2]
[3]
[4]
[5]
[6]]

> 1 x 6 배열로 변형

```
print(arr.reshape(1, 6))
```

Out : [[1 2 3 4 5 6]]

Numpy 활용

- Reshape

> 3 x 1 x 2 배열로 변형

```
print(arr.reshape(3, 1, 2))
```

Out :
[[[1 2]]
[[3 4]]
[[5 6]]]

> ? x 2 배열로 변형

```
print(arr.reshape(-1, 2))
```

Out :
[[1 2]
[3 4]
[5 6]]

Numpy 기초

연습문제

- Reshape

- > [문제1]

- range() 함수를 사용해 1부터 36까지의 숫자가 나열된
tensor 배열을 생성하고 (3, 3, 4) 형태의 3차원 배열로
변형해봅시다.

- > [문제2]

- tensor 배열을 2차원 배열로 변형시켜봅시다. * 열의 개수는 9

Numpy 기초

연습문제

- Reshape

> [문제1]

range() 함수를 사용해 1부터 36까지의 숫자가 나열된
tensor 배열을 생성하고 (3, 3, 4) 형태의 3차원 배열로
변형해봅시다.

```
tensor = np.array(range(1, 37)).reshape(3, 3, 4)
```

Out :

```
[[[ 1  2  3  4]
   [ 5  6  7  8]
   [ 9 10 11 12]]
 ...
 [[25 26 27 28]
 [29 30 31 32]
 [33 34 35 36]]]
```

Numpy 기초

연습문제

- Reshape

> [문제2]

tensor 배열을 2차원 배열로 변형시켜봅니다. * 열의 개수는 9

```
print(tensor.reshape(-1, 9))
```

Out :

```
[[ 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 32 33 34 35 36]]
```

- Flatten

- 다차원 배열을 1차원 배열로 변형

> shape(2, 3) -> shape (6,)

```
print(arr.flatten())
```

Out : [1 2 3 4 5 6]

> shape(1, 6) -> shape (6,)

```
print(arr.reshape(1, 6).flatten())
```

Out : [1 2 3 4 5 6]

> shape(3, 1, 2) -> shape (6,)

```
print(arr.reshape(3, 1, 2).flatten())
```

Out : [1 2 3 4 5 6]

- Flatten

- 다차원 배열을 1차원 배열로 변형

> shape(2, 3) -> shape (6,)

```
print(arr.flatten())
```

Out : [1 2 3 4 5 6]

> shape(1, 6) -> shape (6,)

```
print(arr.reshape(1, 6).flatten())
```

Out : [1 2 3 4 5 6]

> shape(3, 1, 2) -> shape (6,)

```
print(arr.reshape(3, 1, 2).flatten())
```

Out : [1 2 3 4 5 6]

Numpy 활용

- Arange

- range 함수와 비슷한 기능
- 값의 범위를 지정하여 값이 채워져 있는 배열을 생성
- 특정한 규칙에 따라 증가하는 값을 넣는 것도 가능

> np.arange(size)

```
print(np.arange(30))
```

Out : [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]

Numpy 활용

- Arange

011

> 간격 지정 : np.arange(start, end, step)

```
print(np.arange(0, 10, 2))
```

Out : [0 2 4 6 8]

```
print(np.arange(0, 2, 0.2))
```

Out : [0. 0.2 0.4 0.6 0.8 1. 1.2 1.4 1.6 1.8]

- Arange + Reshape

- 배열 요소를 순서대로 가지는 다차원 배열 생성 가능

> arange + reshape

```
print(np.arange(1, 26).reshape(5,5))
```

Out : [[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]]

- Arange + Reshape

- 배열 요소를 순서대로 가지는 다차원 배열 생성 가능

> arange + reshape

```
print(np.arange(0, 6, 0.2).reshape(3, 2, 5))
```

Out :
[[[0. 0.2 0.4 0.6 0.8]
 [1. 1.2 1.4 1.6 1.8]]

[[2. 2.2 2.4 2.6 2.8]
 [3. 3.2 3.4 3.6 3.8]]

[[4. 4.2 4.4 4.6 4.8]
 [5. 5.2 5.4 5.6 5.8]]]

Numpy 기초

연습문제

- Arange + Reshape

014

- > [문제3]

arange, reshape 함수를 사용해 100부터 190까지의 숫자가 10 간격으로 나열된 배열을 생성하고 (2, 5) 형태의 2차원 배열로 변형해봅시다.

- > [문제4]

arange, reshape 함수를 사용해 -2부터 2까지의 숫자가 0.5 간격으로 나열된 배열을 생성하고 3차원 배열로 변형해봅시다.

Numpy 기초

연습문제

- Arange + Reshape

015

> [문제3]

arange, reshape 함수를 사용해 100부터 190까지의 숫자가 10 간격으로 나열된 배열을 생성하고 (2, 5) 형태의 2차원 배열로 변형해봅시다.

```
print(np.arange(100,191,10).reshape(2,5))
```

Out :

```
[[100 110 120 130 140]
 [150 160 170 180 190]]
```

Numpy 기초

연습문제

- Arange + Reshape

016

> [문제4]

arange, reshape 함수를 사용해 -2부터 2까지의 숫자가
0.5 간격으로 나열된 배열을 생성하고 (3, 3) 형태의 2차원 배열로
변형해봅시다.

```
print(np.arange(-2, 2.1, 0.5).reshape(3,3))
```

Out :
[[-2. -1.5 -1.]
 [-0.5 0. 0.5]
 [1. 1.5 2.]]

- Zeros

- 0으로 채워진 배열 생성
- shape 지정이 가능

> np.zeros(shape, dtype)

```
print(np.zeros(shape=(5, ), dtype=np.int8))
```

Out : [0 0 0 0 0]

```
print(np.zeros((5), int))
```

Out : [0 0 0 0 0]

```
print(np.zeros((2,3)))
```

Out : [[0. 0. 0.]
[0. 0. 0.]]

Numpy 활용

- Ones

- 1으로 채워진 배열 생성
- shape 지정이 가능

> np.ones(shape, dtype)

```
print(np.ones(shape=(5, ), dtype=np.int8))
```

Out : [1 1 1 1 1]

```
print(np.ones((5), int))
```

Out : [1 1 1 1 1]

```
print(np.ones((2,3)))
```

Out : [[1. 1. 1.]
[1. 1. 1.]]

Numpy 활용

- Empty

- shape만 주어지고 비어있는 배열 생성
- 배열을 생성만 하고 값을 주지 않아 메모리에 저장되어 있던 기존 값이 저장될 수 있음

> np.empty(shape, dtype)

```
print(np.empty(shape = (10,), dtype = np.int32))
```

Out : [0 0 0 0 0 0 0 0 0 0]

```
print(np.empty((2,5), np.int8))
```

Out : [[0 0 0 0 0]
[0 0 0 0 0]]

Numpy 활용

- Zeros_like / Ones_like / Empty_like

- 입력 받은 배열과 같은 shape, dtype의 배열 생성

> np.zeros_like(ndarray)

```
matrix = np.arange(24).reshape(4, 6)
print(np.zeros_like(matrix))
```

Out :

```
[[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.ones_like(ndarray)

```
matrix = np.arange(-2, 2, 0.2).reshape(2, 10)
print(np.ones_like(matrix))
```

Out :

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
```

Numpy 활용

- Zeros_like / Ones_like / Empty_like

- 입력 받은 배열과 같은 shape, dtype의 배열 생성

```
> np.empty_like(ndarray)
```

```
matrix = np.arange(30).reshape(5, 6)
```

```
print(np.empty_like(matrix))
```

```
Out: [[ 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]]
```

- Identity

022

- 단위 행렬 - 주대각선의 값이 1이고 나머지는 0인 정사각 행렬

> np.identity(size, dtype)

```
print(np.identity(5, dtype=np.int8))
```

Out : $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

```
print(np.identity(2))
```

Out : $\begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$

Numpy 활용

- Eye

- 대각선이 1로 채워지는 행렬
- 대각선의 시작 위치 지정 가능

> `np.eye(size, M, k, dtype)`

```
print(np.eye(5, dtype=np.int8))
```

Out : `[[1 0 0 0 0]
[0 1 0 0 0]
[0 0 1 0 0]
[0 0 0 1 0]
[0 0 0 0 1]]`

- Eye

- k : 대각선의 시작 위치를 입력 받은 수 만큼 오른쪽으로 이동
- M : 보여지는 열의 개수를 지정

> np.eye(size, M, k, dtype)

```
print(np.eye(5, k = 2))
```

Out :

```
[[0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
```


- Eye

- k : 대각선의 시작 위치를 입력 받은 수 만큼 오른쪽으로 이동
- M : 보여지는 열의 개수를 지정

> np.eye(size, M, k, dtype)

```
print(np.eye(5, M = 10, k = -1))
```

Out :

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
```

Numpy 활용

- Full

- 입력 값으로 채워지는 행렬

> np.full((shape), value)

```
print(np.full((2, 5), 3))
```

Out :

```
[[3 3 3 3 3]
 [3 3 3 3 3]]
```

```
print(np.full((5, 4), 'a'))
```

Out :

```
['a' 'a' 'a' 'a']
['a' 'a' 'a' 'a']
['a' 'a' 'a' 'a']
['a' 'a' 'a' 'a']
['a' 'a' 'a' 'a']
```

- Random.randint

- 주어진 범위 안의 랜덤한 값을 뽑는 함수

> np.random.randint(range)

```
# 0~5 까지의 정수 중 하나 뽑기  
print(np.random.randint(6))
```

Out : 5

```
# 1~19 까지의 정수 중 하나 뽑기  
print(np.random.randint(1,20))
```

Out : 11

- Random.rand

- 랜덤한 값으로 채워지는 배열
- 표준정규분포 난수를 shape 형태의 배열로 생성
- 0 ~ 1 사이의 값을 가짐

> np.random.rand(shape)

```
print(np.random.rand(3))
```

Out : [0.57905605 0.00533085 0.12858647]

```
print(np.random.rand(3,2))
```

Out : [[0.68689508 0.32713029]
[0.6481698 0.35997786]
[0.57364588 0.79425818]]

- Random.randn

029

- 랜덤한 값으로 채워지는 배열
- 평균 0, 표준편차 1의 가우시안 표준정규분포 난수를 shape 형태의 배열로 생성

> np.random.randn(shape)

```
print(np.random.randn(3))
```

Out : [-0.01259062 -1.0451196 1.35525447]

```
print(np.random.randn(3,2))
```

Out : [[0.69510343 -2.16322795]
 [-1.13541398 -1.63182969]
 [-0.48371222 0.77480652]]

• 연산함수

- 배열의 요소값들을 이용하여 연산을 할 수 있게 해주는 함수

sum (합)	mean (평균)	max (최대값)	min (최소값)
log (로그)	sqrt (제곱근)	std (표준편차)	exp (지수)
sin (삼각함수)	cos (삼각함수)	tan (삼각함수)	abs / fabs (절대값)
ceil (올림)	floor (버림)	round (반올림)	mod (나머지)
add (덧셈)	subtract (뺄셈)	multiply (곱셈)	divide (나눗셈)
power (제곱)	sort, median, var, ...		

- 합 (sum)

- 요소들의 합을 구해주는 함수

- ndarray.sum()

> 2차원 배열 합

```
matrix = np.arange(1, 7).reshape(2, 3)
print(matrix.sum())
```

Out : 21

```
print(matrix.sum(axis=0)) # 행끼리 더하기(위 아래)
print(matrix.sum(axis=1)) # 열끼리 더하기(좌 우)
```

Out : [5 7 9]
[6 15]

Numpy 활용

- 합 (sum)

> 3차원 배열 합

```
tensor = np.arange(1, 19).reshape(3, 2, 3)
print(tensor)
print(tensor.sum())
```

Out :

```
[[[ 1  2  3]
    [ 4  5  6]
    [ 7  8  9]
    [10 11 12]
    [13 14 15]
    [16 17 18]]]
171
```

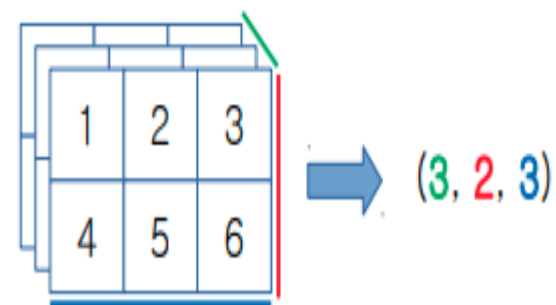

Numpy 활용

- 합 (sum)

> 3차원 배열 합

```
print(tensor.sum(axis=0))  
print(tensor.sum(axis=1))  
print(tensor.sum(axis=2))
```

Out :
[[21 24 27]
 [30 33 36]]
[[5 7 9]
 [17 19 21]
 [29 31 33]]
[[6 15]
 [24 33]
 [42 51]]



Numpy 활용

- 평균 (mean)

- 합과 동일

> 2차원 배열 평균

```
print(matrix.mean())          # 전체 평균  
print(matrix.mean(axis=0))    # 행끼리 평균(위 아래)  
print(matrix.mean(axis=1))    # 열끼리 평균(좌 우)
```

Out : 3.5

[2.5 3.5 4.5]

[2. 5.]

- 표준 편차 (std)

- 합, 평균과 동일

> 2차원 배열 표준 편차

```
print(matrix.std())          # 전체 표준 편차  
print(matrix.std(axis=0))    # 행끼리 표준 편차(위 아래)  
print(matrix.std(axis=1))    # 열끼리 표준 편차(좌 우)
```

Out : 1.707825127659933

[1.5 1.5 1.5]

[0.81649658 0.81649658]

- 그 외 수학 연산자

036

> 지수 (exp)

```
print(np.exp(matrix))
```

```
Out: [[ 2.71828183  7.3890561 20.08553692]
      [54.59815003 148.4131591 403.42879349]]
```

> 로그 (log)

```
print(np.log(matrix))
```

```
Out: [[0.      0.69314718 1.09861229]
      [1.38629436 1.60943791 1.79175947]]
```

- 그 외 수학 연산자

037

> 제곱근 (sqrt)

```
print(np.sqrt(matrix))
```

```
Out: [[1.      1.41421356 1.73205081]
      [2.      2.23606798 2.44948974]]
```

> 삼각함수 (sin)

```
print(np.sin(matrix))
```

```
Out: [[ 0.84147098  0.90929743  0.14112001]
      [-0.7568025  -0.95892427 -0.2794155 ]]
```

- 그 외 수학 연산자

038

> 삼각함수 (cos)

```
print(np.cos(matrix))
```

Out : $\begin{bmatrix} 0.54030231 & -0.41614684 & -0.9899925 \\ -0.65364362 & 0.28366219 & 0.96017029 \end{bmatrix}$

> 삼각함수 (tan)

```
print(np.tan(matrix))
```

Out : $\begin{bmatrix} 1.55740772 & -2.18503986 & -0.14254654 \\ 1.15782128 & -3.38051501 & -0.29100619 \end{bmatrix}$

- 배열 정렬 (Sort)

- > 1차원 배열 정렬

```
# randint를 사용하여 1~100 숫자 중 5개를 뽑아 배열 생성
vector = np.array([np.random.randint(1, 100)
                    for n in range(5)])
print(vector)
```

Out : [23 91 34 76 61]

```
print(np.sort(vector))      # 오름차순으로 정렬
```

Out : [23 34 61 76 91]

```
print(np.sort(vector)[::-1]) # 내림차순으로 정렬
```

Out : [91 76 61 34 23]

- 배열 정렬 (Sort)

- > 2차원 배열 정렬

```
# randint를 사용하여 1~100 숫자 중 20개를 뽑아 배열 생성
matrix = np.array([np.random.randint(1, 100)
                    for n in range(20)]).reshape(4, 5)
print(matrix)
```

Out :

```
[[82  5 18 67 91]
 [20 53 46 25 66]
 [59 46 27 92 54]
 [76 62 15 24 44]]
```


- 배열 정렬 (Sort)

- > 2차원 배열 정렬

```
print(np.sort(matrix)) # 열 기준으로 정렬
```

```
Out : [[ 5 18 67 82 91]
       [20 25 46 53 66]
       [27 46 54 59 92]
       [15 24 44 62 76]]
```

```
print(np.sort(matrix, axis = 0)) # 행 기준으로 정렬
```

```
Out : [[20  5 15 24 44]
       [59 46 18 25 54]
       [76 53 27 67 66]
       [82 62 46 92 91]]
```

- 배열 합치기 (Vstack)

- 2개 이상의 배열을 수직으로 합체

```
> np.vstack([arr1, arr2])
```

```
vector1 = np.array([1, 2, 3])
```

```
vector2 = np.array([4, 5, 6])
```

```
print(np.vstack([vector1, vector2]))
```

```
Out : [[1 2 3]
       [4 5 6]]
```

- 배열 합치기 (Hstack)

- 2개 이상의 배열을 수평으로 합체

> np.hstack([arr1, arr2])

```
vector1 = np.array([1, 2, 3]).reshape(3, 1)
```

```
vector2 = np.array([4, 5, 6]).reshape(3, 1)
```

```
print(np.hstack([vector1, vector2]))
```

Out : [[1 4]

[2 5]

[3 6]]

- 배열 합치기 (Concatenate)

- 2개 이상의 배열을 수직, 수평으로 합체

> `np.concatenate(arr1, arr2, axis)`

```
vector1 = np.array([1, 2, 3])  
vector2 = np.array([4, 5, 6])  
print(np.concatenate([vector1, vector2], axis = 0))
```

Out : [1 2 3 4 5 6]

- 배열 합치기 (Concatenate)

- 2개 이상의 배열을 수직, 수평으로 합체

> `np.concatenate(arr1, arr2, axis)`

```
vector1 = np.array([1, 2, 3]).reshape(3, 1)
vector2 = np.array([4, 5, 6]).reshape(3, 1)
print(np.concatenate([vector1, vector2], axis = 1))
```

Out : `[[1 4]`
`[2 5]`
`[3 6]]`

- 배열 합치기 (Concatenate)

- 2개 이상의 배열을 수직, 수평으로 합체

> `np.concatenate(arr1, arr2, axis)`

```
matrix1 = np.arange(1,5).reshape(2,2)
matrix2 = np.arange(5,9).reshape(2,2)
print(np.concatenate([matrix1, matrix2], axis = 0))
```

Out : `[[1 2]`
`[3 4]`
`[5 6]`
`[7 8]]`

- 배열 합치기 (Concatenate)

- 2개 이상의 배열을 수직, 수평으로 합체

> `np.concatenate(arr1, arr2, axis)`

```
matrix1 = np.arange(1,5).reshape(2,2)
matrix2 = np.arange(5,9).reshape(2,2)
print(np.concatenate([matrix1, matrix2], axis = 1))
```

Out : `[[1 2 5 6]`
`[3 4 7 8]]`

- 행 / 열 바꾸기 (transpose)

- 행과 열의 뒤집어 주는 함수

> ndarray.transpose()

> ndarray.T

```
matrix3 = np.array([[5, 6]])  
print(matrix3)
```

Out : [[5 6]]

```
print(matrix3.transpose())  
print(matrix3.T)
```

Out : [[5]
[6]]
[[5]
[6]]

- Transpose + Concatenate

- > 형태를 바꿔 서로 결합할 수 없는 배열을 합칠 수 있다.

```
print(np.concatenate([matrix1, matrix3]))
```

Out : $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

```
print(np.concatenate([matrix1, matrix3], axis = 1))
```

ValueError Traceback (most recent call last)

<ipython-input-652-2419e8fc4633> in <module>

----> 1 print(np.concatenate([matrix1, matrix3], axis = 1))

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 0, the array at index 0 has size 2 and the array at index 1 has size 1

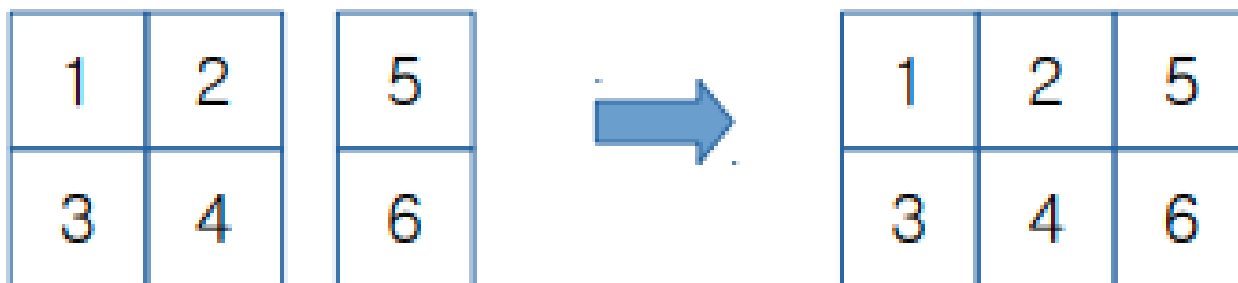
Numpy 활용

- Transpose + Concatenate

> 형태를 바꿔 서로 결합할 수 없는 배열을 합칠 수 있다.

```
print(np.concatenate([matrix1, matrix3.T], axis = 1))
```

Out : $\begin{bmatrix} 1 & 2 & 5 \\ 3 & 4 & 6 \end{bmatrix}$



Numpy 활용

- 인덱스 반환 함수

- > 최대값, 최소값의 인덱스

- argmax, argmin

051

```
a = np.arange(1, 11).reshape(2, 5)
print(a)
```

```
Out : [[ 1  2  3  4  5]
       [ 6  7  8  9 10]]
```

```
print(np.argmax(a), np.argmin(a))
```

```
Out : 9 0
```

```
print(np.argmax(a, axis=0))
```

```
Out : [1 1 1 1 1]
```

Numpy 활용

- 인덱스 반환 함수

> 최대값, 최소값의 인덱스

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

```
Out : [[1 2 3 4]  
       [2 3 4 1]  
       [3 4 1 2]]
```

```
print(np.argmax(b, axis = 1))
```

```
Out : [3 2 1]
```

```
print(np.argmax(b, axis=0))
```

```
Out : [2 2 1 0]
```

Numpy 활용

- 배열 인덱싱(Fancy Index)

- > 인덱스의 값을 가지는 배열을 통해 인덱싱

```
a = np.arange(1, 11)
b = np.array([3, 4, 5, 6])
print(a)
print(b)
```

Out: [1 2 3 4 5 6 7 8 9 10]

[3 4 5 6]

```
print(a[b])
```

Out: [4 5 6 7]

a	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5	6	7	8	9	10

→ a[b]

4	5	6	7
---	---	---	---

b	3	4	5	6
---	---	---	---	---

Numpy 활용

- 배열 인덱싱

- > 인덱스 반환 함수 + fancy index

```
a = np.arange(10, 20)
b = np.argmax(a)
print(a)
print(b)
```

Out : [10 11 12 13 14 15 16 17 18 19]
9

```
print(a[b])
```

Out : 19