배열의 축소

배열의 차원을 축소하면 정보의 처리를 더 간단하게 할 수 있습니다. 다음과 같이 3X3 배열을 만들었다고 합시다.

```
a33 = np.array([[1,2,3],
                [4,5,6],
                [7,8,9]])
a33
```

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

위 배열을 1차원으로 축소하려면 아래와 같이 flatten 함수를 사용할 수 있습니다.

```
a33.flatten()
```

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

flatten 함수는 2차원 배열뿐만 아니라 다차원 배열도 1차원으로 축소합니다.

행렬의 종류

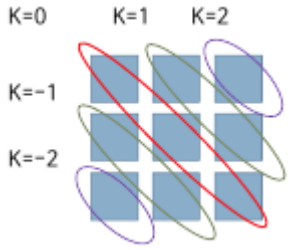
대각행렬(Diagonal Matrix)

정사각행렬에서 대각선의 원소를 제외한 모든 원소가 0으로 구성된 행렬을 대각행렬이라고 합니다.

```
mdiag33 = np.array([[1,0,0],
                    [0,3,0],
                    [0,0,5]])
mdiag33
```

```
array([[1, 0, 0],
       [0, 3, 0],
       [0, 0, 5]])
```

대각행렬 구성시, 주대각선이 아닐 경우에도 구성이 가능한데요, 아래 그림처럼 k = 0 이외에도 위쪽은 k = 1, k = 2 등으로 만들 수 있고, 아래쪽



은 k = -1 ...로 구성할 수 있습니다.

단위행렬(Unit Matrix) 또는 항등행렬(Identity Matrix)

대각행렬 중 주 대각선의 원소가 모두 1인 행렬을 단위행렬이라고 합니다. 보통 단위행렬은 대문자 I로 표시합니다.

삼각행렬(Triangular Matrix)

정사각행렬에서 주대각선을 기준으로 위나 아래의 원소가 모두 0인 행렬을 삼각행렬이라고 합니다. 대각선 위쪽만 0이 아닌 원소로 구성되면 상삼각행렬이되고, 아래쪽만 0이 아닌 원소이면 하삼각행렬이 됩니다.

```
trilo = np.array([[1,0,0],
                  [4,5,0],
                  [7,8,9]])
trilo
```

```
array([[1, 0, 0],
       [4, 5, 0],
       [7, 8, 9]])
```

삼각행렬도 주대각선을 위쪽 혹은 아래쪽으로 이동할 수 있습니다.

```
trilo_1d = np.array([[0,0,0],
                     [4,0,0],
                     [7,8,0]])
trilo_1d
```

```
array([[0, 0, 0],
       [4, 0, 0],
       [7, 8, 0]])
```

삼각행렬의 전치관계

삼각행렬을 전치하여 하삼각행렬은 상감각행렬로, 상삼각행렬은 하삼각행렬로 변경할 수 있습니다.

```
trilo.T
```

```
array([[1, 4, 7],
       [0, 5, 8],
       [0, 0, 9]])
```

삼각행렬의 곱셈은 일반 행렬의 곱셈과는 달리 동일한 원소의 제곱을 출력합니다.

```
trilo * trilo
```

```
array([[ 1,  0,  0],
       [16, 25,  0],
       [49, 64, 81]])
```

상삼각행렬과 하삼각행렬의 곱은 주대각선의 제곱을 제외하고 나머지는 모두 0이 됩니다.

영행렬(Null Matrix, Zero Matrix)

모든 원소가 0인 행렬을 영행렬이라고 합니다.

대칭행렬(Symmetric Matrix)

대칭행렬은 주대각선을 기준으로 아래의 원소와 위의 원소가 동일한 값을 가진 행렬을 말합니다. 이때 주대각선은 행과 열의 인덱스가 동일한 원소를 기준으로 합니다.

```
sm = np.array([[1,2,3],
               [2,3,5],
               [3,5,6]])
sm
```

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

대칭행렬과 이 행렬의 전치행렬을 더하면 원소들의 값이 두배로 증가하지만, 대칭행렬을 유지합니다.

```
sm + sm.T
```

```
array([[ 2,  4,  6],
       [ 4,  6, 10],
       [ 6, 10, 12]])
```

대칭행렬과 이 대칭행렬의 전치행렬을 곱해도 대칭행렬을 유지합니다.

```
np.dot(sm, sm.T)
```

```
array([[14, 23, 31],
       [23, 38, 51],
       [31, 51, 70]])
```

반대칭행렬(Skew Symmetric Matrix)

대각성분을 기준으로 값은 같으나 부호가 반대인 원소를 가진 행렬을 반대칭행렬이라고 합니다.

```
ssm = np.array([[0,2,-3],
                [-2,0,5],
                [3,-5,0]])
ssm
```

```
array([[ 0,  2, -3],
       [-2,  0,  5],
       [ 3, -5,  0]])
```

이 행렬과 전치행렬을 더하면 모든 원소가 0이 됩니다.

```
ssm + ssm.T
```

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

반대칭행렬과 이것의 전치행렬간 뺄셈은 반대칭행렬을 그대로 유지합니다.

```
ssm - ssm.T
```

```
array([[  0,   4,  -6],
       [ -4,   0,  10],
       [  6, -10,   0]])
```

반대칭행렬과 이것의 전치행렬간 행렬곱은 대칭행렬이 됩니다.

```
np.dot(ssm,ssm.T)
```

```
array([[ 13, -15, -10],
       [-15,  29,  -6],
       [-10,  -6,  34]])
```

치환행렬(Permutation Matrix)

단위행렬이나 행렬의 행을 교환하여 재구성하거나 특정 상수를 곱해서 행을 교환한 것을 치환행렬이라고 합니다.

9개의 원소를 가진 3행3열의 행렬을 만듭니다. 단위행렬도 하나 만들겠습니다.

```
pm = np.arange(1,10).reshape(3,3)
pm
```

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

```
pmu = np.eye(3, dtype='int')
pmu
```

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

위 행렬과 단위행렬을 곱하면 변경되는 항목없이 그대로입니다.

```
np.dot(pm,pmu)
```

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

```
pmu21 = np.array([[0,1,0],
                  [1,0,0],
                  [0,0,1]])
np.dot(pm,pmu21)
```

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

행변환이 일어난 단위행렬을 하나 더 만들어서 치환행렬을 만듭니다. 이 치환행렬에 행렬곱하면 변환된 것을 볼 수 있습니다.

직교행렬(Orthogonal Matrix)

직교행렬은 행렬 자신과 그 전치행렬의 행렬곱은 단위행렬이 되는 행렬입니다.

전치행렬(Transpose Matrix)

전치행렬은 행과 열을 서로 바꾸어서 만듭니다.

```
tmorg = np.arange(1,13).reshape(4,3)
tmorg
```

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

```
tmorg.T
```

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

다차원 배열의 검색

색인검색

```
v = np.array([1,2,3,4,5])
```

```
v[0]
```

```
1
```

```
v[-1]
```

```
5
```

```
m = np.array([[1,2,3,4,5],
              [6,7,8,9,10]])
```

```
m[-1]
```

```
array([ 6,  7,  8,  9, 10])
```

```
m[0,1]
```

```
2
```

```
m[-1,-1]
```

```
10
```