NumPy (Numerical Python)

- 수학 및 과학적 연산을 쉽고 빠르게 지원
- 다차원 행렬(Array/Matrix)을 효과적으로 처리
- 일반적으로 같은 데이터 타입 값으로 구성
- https://numpy.org

```
import warnings
warnings.filterwarnings('ignore')
```

I. NumPy Package import ~ as

```
import numpy as np
```

- Version Check

```
np.__version__
```

II. Array 생성

- Python List 구조를 사용

1) Scalar - 0D Array - Rank0 Tensor

```
a0 = np.array(9)
```

```
print(a0)
```

```
9
```

2) Vector - 1D Array - Rank1 Tensor

```
a1 = np.array([1, 3, 5, 7, 9])
```

```
print(a1)
```

```
[1 3 5 7 9]
```

```
a1[2]
```

```
5
```

```
a1[1:3]
```

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

3) Matrix - 2D Array - Rank2 Tensor

```
a2 = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
```

```
print(a2)
```

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

- a2[행]

```
a2[1]
```

```
array([4, 5, 6])
```

- a2[행, 열]

```
a2[1, 1]
```

```
5
```

- a2[행, 열]

```
a2[:, 1]
```

```
array([2, 5, 8])
```

4) Array - 3D Array - Rank3 Tensor

```
a3 = np.array([[[1, 2],
                [3, 4]],
               [[5, 6],
                [7, 8]],
               [[9, 10],
                [11, 12]]])
```

```
print(a3)
```

```
[[[ 1  2]
  [ 3  4]]

 [[ 5  6]
  [ 7  8]]

 [[ 9 10]
  [11 12]]]
```

- a3[축]

```
a3[1]
```

```
array([[5, 6],
       [7, 8]])
```

- a3[축, 행]

```
a3[1, 1]
```

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

- a3[축, 행, 열]

```
a3[1, 1, 1]
```

```
8
```

- a3[축, 행, 열]

```
a3[:, 0, 0]
```

```
array([1, 5, 9])
```

- a3[축, 행, 열]

```
a3[:, :, 0]
```

```
array([[ 1,  3],
       [ 5,  7],
       [ 9, 11]])
```

III. AR.shape and AR.reshape( )

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

```
print(AR)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

1) .shape

- 행렬 크기

```
AR.shape
```

```
(12,)
```

- 행렬 차원

```
AR.ndim
```

```
1
```

- 행렬 원소 개수

```
AR.size
```

```
12
```

2) .reshape(3, 4)

- .reshape(행, 열)

```
AR2 = AR.reshape(3, 4)
```

```
print(AR2)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

- 행렬 크기

```
AR2.shape
```

```
(3, 4)
```

- 행렬 차원

```
AR2.ndim
```

```
2
```

- 행렬 원소 개수

```
AR2.size
```

```
12
```

3) .reshape(3, 2, 2)

- .reshape(축, 행, 열)

```
AR3 = AR.reshape(3, 2, 2)
```

```
print(AR3)
```

```
[[[ 1  2]
  [ 3  4]]

 [[ 5  6]
  [ 7  8]]

 [[ 9 10]
  [11 12]]]
```

- 행렬의 크기

```
AR3.shape
```

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

- 행렬의 차원

```
AR3.ndim
```

```
3
```

- 행렬의 원소 개수

```
AR3.size
```

```
12
```

4) .reshape(-1, 1)

- .reshape(12, 1)

```
AR.reshape(-1, 1)
```

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

- .reshape(1, 12)

```
AR2.reshape(1, -1)
```

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

```
AR2.reshape(12)
```

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

```
AR3.reshape(-1)
```

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

- .flatten( )

```
AR3.flatten()
```

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

IV. 범위 지정(arange) 함수

1) 연속된 10개 값 생성

```
np.arange(10)
```

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

2) 1부터 9까지 1간격으로 생성

```
np.arange(1, 10)
```

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

3) 1부터 9까지 2간격으로 생성

```
np.arange(1, 10, 2)
```

```
array([1, 3, 5, 7, 9])
```

4) Array 생성 후 .reshape( ) 적용

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

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

V. 특별한 형태의 Array 생성

1) 0과 1로만 구성된 Array

- 0으로만 구성

```
np.zeros(9)
```

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

```
np.zeros([3, 4])
```

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

- 1로만 구성

```
np.ones(9)
```

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

```
np.ones([4, 3])
```

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

- 산술연산을 적용하여 '9'로만 구성된 행렬 생성

```
np.zeros([3, 4]) + 9
```

```
array([[9., 9., 9., 9.],
       [9., 9., 9., 9.],
       [9., 9., 9., 9.]])
```

2) 3 x 3 단위행렬

```
np.eye(3)
```

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

3) 난수 Array 생성

- 실수 난수 생성
  - (축, 행, 열)

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

```
array([[[0.64848452, 0.99506736],
        [0.07311662, 0.12909484]],

       [[0.74977182, 0.85048786],
        [0.85663604, 0.37699742]],

       [[0.98785292, 0.54096569],
        [0.41338835, 0.50345139]]])
```

- 주어진 정수 범위에서 난수 생성
  - 1 ~ 44 사이
  - 복원 추출
  - (행, 열)

```
np.random.randint(1, 45, size = (5, 6))
```

```
array([[ 8, 27, 41, 43, 10, 27],
       [40, 25, 19, 18, 36, 42],
       [25, 38, 22, 15,  8,  7],
       [11, 14, 38, 28, 33,  2],
       [ 8, 37, 35, 33, 31, 17]])
```

- np.random.seed( )
  - 의사난수(Pseudo Random Number) 생성 초기값 지정
  - 항상 같은 난수 생성
  - 비복원 추출

```
np.random.seed(2045)

np.random.choice(np.arange(1, 46), size = (5 ,6), replace = False)
```

```
array([[ 7, 32,  6, 41,  4, 34],
       [30, 28, 16,  5, 38, 33],
       [31, 13, 25, 23, 43, 12],
       [45,  8, 29, 22, 18,  9],
       [21, 20, 40,  2, 37, 39]])
```

- shuffle( )
  - 원소 섞기

```
TA = np.arange(1, 10)

TA
```

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

```
np.random.shuffle(TA)

TA
```

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

VI. Array 연산

```
A1 = np.array([85, 93, 75, 97, 69])

A2 = np.array([91, 90, 85, 97, 89])

A3 = np.array([[85, 93, 75],
               [90, 84, 97],
               [99, 91, 80]])
```

1) 기본 연산

- 각각의 행과 열의 값을 매칭하여 연산 수행

```
A1 + A2
```

```
array([176, 183, 160, 194, 158])
```

```
A2 - A1
```

```
array([ 6, -3, 10,  0, 20])
```

```
A1 * A2
```

```
array([7735, 8370, 6375, 9409, 6141])
```

```
A2 / A1
```

```
array([1.07058824, 0.96774194, 1.13333333, 1.        , 1.28985507])
```

```
A1 * 3
```

```
array([255, 279, 225, 291, 207])
```

```
A1 ** 2
```

```
array([7225, 8649, 5625, 9409, 4761])
```

2) 통계량 연산

- 총합

```
A1.sum()
```

```
419
```

- 평균

```
A2.mean()
```

```
90.4
```

- 분산
  - 'ddof = 0'

```
A2.var()
```

```
15.040000000000001
```

- 표준 편차
  - 'ddof = 0'

```
A2.std()
```

```
3.8781438859330635
```

- 최소값

```
A2.min()
```

```
85
```

> - A3 전체 최소값

```
A3.min()
```

```
75
```

> - A3 각 열의 최소값

```
A3.min(axis = 0)
```

```
array([85, 84, 75])
```

> - A3 각 행의 최소값

```
A3.min(axis = 1)
```

```
array([75, 84, 80])
```

```
A3.min(axis = 1, keepdims = True)
```

```
array([[75],
       [84],
       [80]])
```

- 최대값

```
A2.max()
```

```
97
```

> - A3 전체 최대값

```
A3.max()
```

```
99
```

> - A3 각 열의 최대값

```
A3.max(axis = 0)
```

```
array([99, 93, 97])
```

> - A3 각 행의 최대값

```
A3.max(axis = 1)
```

```
array([93, 97, 99])
```

```
A3.max(axis = 1, keepdims = True)
```

```
array([[93],
       [97],
       [99]])
```

- 누적(Cumulative)합

```
A1.cumsum()
```

```
array([ 85, 178, 253, 350, 419])
```

- 누적(Cumulative)곱

```
A1.cumprod()
```

```
array([       85,      7905,    592875,  57508875, 3968112375])
```

VII. Matrix 연산

- M1, M2 지정

```
M1 = np.array([2, 4, 6, 8]).reshape(2, 2)
```

```
print(M1)
```

```
[[2 4]
 [6 8]]
```

```
M2 = np.array([3, 5, 7, 9]).reshape(2, 2)
```

```
print(M2)
```

```
[[3 5]
 [7 9]]
```

1) Matrix 곱

- M1 @ M2

```
```

```
M1.dot(M2)
```

```
array([[ 34,  46],
       [ 74, 102]])
```

- M2 @ M1

```
np.dot(M2, M1)
```

```
array([[ 36,  52],
       [ 68, 100]])
```

- Warning: M1 * M2

```
M1 * M2
```

```
array([[ 6, 20],
       [42, 72]])
```

```
M2 * M1
```

```
array([[ 6, 20],
       [42, 72]])
```

2) 전치 행렬

- M1의 전치 행렬

```
np.transpose(M1)
```

```
array([[2, 6],
       [4, 8]])
```

- M2의 전치 행렬

```
M2.transpose()
```

```
array([[3, 7],
       [5, 9]])
```

```
M2.T
```

```
array([[3, 7],
       [5, 9]])
```

#

#

#

The End

#

#

#