Numpy Tutorial

https://github.com/dlcjfgmlnasa/Tongmyong_College_Tensorflow_Tutorial

Created by **Choelhui lee**

Python 설치

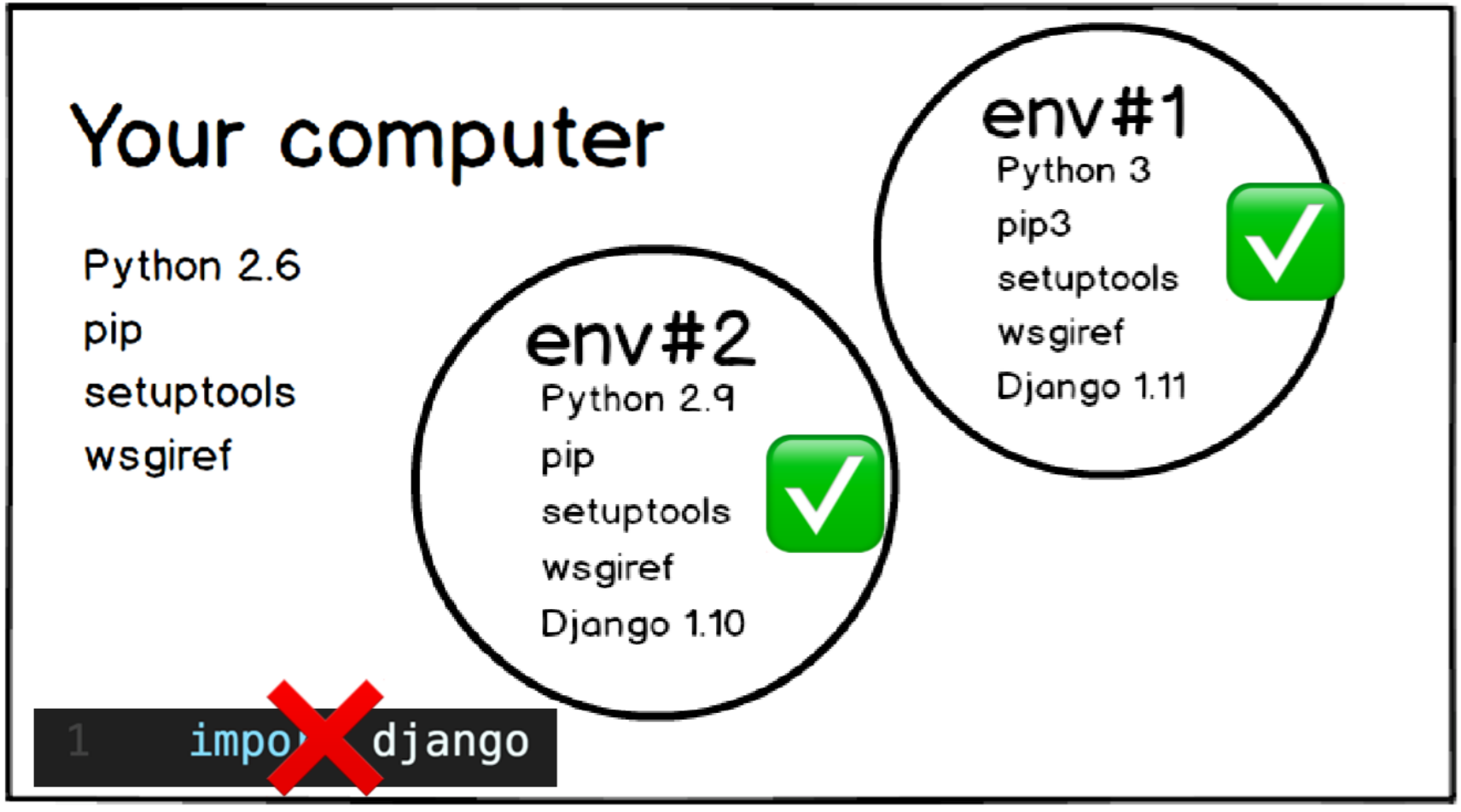
Windows 사용자

- Python 3.6.7 64bit 설치
  - 설치시 열린마음으로 모든 옵션에 체크

Linux 사용자

- `pyenv` 설치하거나 혹은 `sudo apt install python3-pip python3-venv`로 필요한 패키지 선택

python virtualenv



python virtualenv

- 윈도우 사용자의 경우 `PowerShell`을 `CMD`로 변경할 것.
  - `작업 표시줄 설정` > `시작 단추를 마우스 오른쪽[...]` > `Windows PowerShell로 바꾸기` > `끔`

```
C:\workspace> mkdir [파일 이름]
C:\workspace> cd [파일 이름]
C:\workspace\[파일 이름]> python -m venv venv
C:\workspace\[파일 이름]> venv\Scripts\activate

(venv) C:\workspace\[파일 이름]>
```

- `OS X`, `*NIX` 사용자는 `terminal`을 사용할 것

```
[.. 위에 부분 동일]
locs@ ~/Workspace/[파일 이름]$ python -m venv venv
locs@ ~/Workspace/[파일 이름]$ source venv/bin/activate
```

Numpy (넘파이)

- `Numpy` 는 C언어로 구현된 파이썬 라이브러리로써, `고성능의 수치계산을 위해 제작`
- `Numpy` 는 `벡터 및 행렬 연산에 있어서 매우 편리한 기능을 제공`
- `pandas` 와 `matplotlib` 의 기반으로 사용

NumPy

Numpy Install

```
pip install numpy
```

Numpy import

```
import numpy as np
```

01. Array 정의

```
data1 = [1, 2, 3, 4, 5]

>> [1, 2, 3, 4, 5]
```

```
data2 = [1, 2, 3, 3.5, 4]

>> [1, 2, 3, 3.5, 4]
```

01. numpy를 이용하여 array 정의

```
arr1 = np.array(data1)
print(arr1)
print(arr1.shape)
```

```
>> array([1, 2, 3, 4, 5])
>> (5,)
```

02. 바로 python list를 넣어 줌으로써 만듦

```
arr2 = np.array([1,2,3,4,5])
print(arr2)
print(arr2.shape)
```

```
>> array([1, 2, 3, 4, 5])
>> (5,)
```

03. array의 자료형을 확인

```
print(arr1.dtype)
print(arr2.dtype)
```

```
>> dtype('int64')
>> dtype('float64')
```

04. 다중 배열 사용하기

```
arr3 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
print(arr3)
print(arr3.shape)
```

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

>> (4, 3)
```

numpy shape

- numpy 에서는 해당 array의 크기 알수 있음
- shape 을 확인함으로써 **몇개의 데이터가 있는지, 몇 차원으로 존재하는지 등을 확인할 수 있음**
- *example*
  - arr1.shape 의 결과는 (5,) | 1차원의 데이터이며 5 라는 크기를 가짐
  - arr3.shape 의 결과는 (4,3) | 2차원의 데이터이며 4 * 3 크기를 가짐

numpy type

- arr1, arr2 => **int64**
- arr3 => **float64**
    - 3.5라는 실수형 데이터를 갖기 때문

Numpy 자료형	의미
int(8, 16, 32, 64)	부호가 있는 정수
uint(8 ,16, 32, 54)	부호가 없는 정수
float(16, 32, 64, 128)	실수
complex(64, 128, 256)	복소수
bool	불리언
string_	문자열
object	파이썬 오프젝트
unicode_	유니코드

np.zeros(), np.ones(), np.arange()

- `numpy에서 array를 정의할 때 사용되는 함수`

`np.zeros()`

- 인자로 받는 크기만큼, 모든요소가 0인 array를 만듦

```
np.zeros(10)

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

```
np.zeros((3,5))

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

`np.ones()`

- 인자로 받는 크기만큼, 모든요소가 1인 array를 만듦

```
np.zeros(10)

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

```
np.zeros((3,5))

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

`np.arange()`

- 인자로 받는 값 만큼 1씩 증가하는 1차원 array를 만듦
- 하나의 인자만 입력하면 0 ~ 입력한 인자, 값 만큼의 크기를 가진다.

```
np.zeros(10)

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

```
np.arange(3,10)

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

02. Array 연산

- 기본적으로 numpy에서 연산을 할때는 크기가 서로 동일한 array 끼리 연산이 진행
- 이때 같은 위치에 있는 요소들 끼리 연산이 진행

```
arr1 = np.array([[1,2,3],[4,5,6]])

>> array([[1, 2, 3], [4, 5, 6]])
```

```
arr2 = np.array([[10,11,12],[13,14,15]])

>> array([[10, 11, 12], [13, 14, 15]])
```

01. array 덧셈

```
arr1 + arr2

>> array([[11, 13, 15], [17, 19, 21]])
```

02. array 뺄셈

```
arr1 - arr2

>> array([[-9, -9, -9], [-9, -9, -9]])
```

행렬의 곱처럼 곱셈이 진행되는 것이 아니라 `각 요소별로 곱셈이 진행`

03. array 곱셉

```
arr1 * arr2

>> array([[10, 22, 36], [52, 70, 90]])
```

04. array 나눗셈

```
arr1 * arr2

>> array([[10, 22, 36], [52, 70, 90]])
```

05. array의 Broadcast

```
arr1              # arr1 정의

>> array([[1, 2, 3], [4, 5, 6]])
```

```
arr1.shape

>> (2, 3)
```

```
arr3 = np.array([10,11,12])      # arr3 정의

>> array([10, 11, 12])
```

- 아래와 같이 서로 크기가 다른 arr1과 arr3의 연산
- **연산결과를 살펴보면 arr3이 [10,11,12] 에서 [[10,11,12], [10,11,12]]로 확장되어 계산되었음을 확인할 수 있음**
- **array에 스칼라 연산도 가능**

```
arr1 + arr3

>> array([[11, 13, 15], [14, 16, 18]])
```

```
arr1 * 10

>> array([[10, 20, 30], [40, 50, 60]])
```

```
arr1 ** 2

>> array([[ 1, 4, 9], [16, 25, 36]])
```

03. Array 인덱싱

- numpy에서 사용되는 인덱싱은 기본적으로 python 인덱싱과 동일
- python에서와 같이 1번째로 시작하는 것이 아니라 0번째로 시작하는 것에 주의

```
arr1 = np.arange(10)

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

- **01. 0번째 요소**

```
arr1[0]

>> 0
```

- **02. 3번째 요소**

```
arr1[3]

>> 3
```

- **03. 3번째 요소부터 8번째 요소**

```
arr1[3:9]

>> array([3, 4, 5, 6, 7, 8])
```

- **04. 전체 출력 요소**

```
arr1[:]

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

1차원이 아닌 그 이상의 차원에서도 인덱싱이 가능

```
arr2 = np.array([[1,2,3,4],
                 [5,6,7,8],
                 [9,10,11,12]])

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

01. 2차원의 array에서 인덱싱을 하기 위해선 2개의 인자를 입력해야 함

```
arr2[0,0]

>> 1
```

02. 2행의 모든 요소 꺼내기

```
arr2[2,:]

>> array([ 9, 10, 11, 12])
```

03. 2행의 3번째 요소 꺼내기

```
arr2[2,3]

>> 12
```

04. 모든 열의 3번째 요소 꺼내기

```
arr2[:,3]

>> array([ 4,  8, 12])
```

04. Array boolean 인덱싱(마스크)

- 위에서 이용한 다차원의 인덱싱을 응용하여 boolean 인덱싱
- 해당 기능은 주로 마스크라고 이야기하는데, boolean인덱싱을 통해 만들어낸 array를 통해 우리가 원하는 행 또는 열의 값만 뽑아낼 수 있음
- **즉, 마스크처럼 우리가 가리고 싶은 부분은 가리고, 원하는 요소만 꺼낼 수 있음**

```
names = np.array(['Beomwoo','Beomwoo','Kim','Joan','Lee',
                  'Beomwoo','Park','Beomwoo'])

>> array(['Beomwoo', 'Beomwoo', 'Kim', 'Joan', 'Lee',
>>        'Beomwoo', 'Park', 'Beomwoo'], dtype='<U7')
```

```
names.shape

>> (8,)
```

- **np.random.randn()** : 기대값이 0 이고, 표준편차가 1 인 가우시안 정규 분포 를 따르는 **난수를 발생시키는 함수**

```
data = np.random.randn(8,4)

>> array([[-1.07099572, -0.85382063, -1.42474621, -0.0599846],
>>        [ 1.93097843, -1.8885167 , 1.99767454, 0.3152498],
>>        [-0.22633642, -0.76385264, 0.16368804, 0.9120438],
>>        [ 1.34321923, 1.54532121, 0.28814921, 0.5068776],
>>        [ 0.4126606 , -0.52356522, 0.27124037, -0.6383264],
>>        [ 0.88575452, -1.39205929, 0.91019739, -1.4676349],
>>        [-0.05673648, -1.63408607, -2.29844338, -0.662913 ],
>>        [ 0.45963024, 0.35662128, 0.18307525, 1.4692167]])
```

```
data.shape

>> (8, 4)
```

- 위와 같은 `names` 와 `data` 라는 `array` 가 있다.
- `이때, names의 각 요소가 data의 각 행과 연결된다고 가정해보자`
- **`이때, names가 Beomwoo인 행의 data만 보고 싶을 때 다음과 같이 마스크를 사용`**

```
names_Beomwoo_mask = (names == 'Beomwoo')

>> array([True, True, False, False, False, True, False, True])
```

```
data[names_Beomwoo_mask,:]

>> array([[-1.07099572, -0.85382063, -1.42474621, -0.05992846],
>>        [ 1.93097843, -1.8885167 , 1.99767454, 0.31524498],
>>        [ 0.88575452, -1.39205929, 0.91019739, -1.04676349],
>>        [ 0.45963024, 0.35662128, 0.18307525, 1.46992167]])
```

- 위의 결과를 보면, 요소가 Beomwoo인 것은 0번째, 1번째, 5번째, 7번째 이므로 data에서 0,1,5,7행의 모든 요소를 꺼내와야 한다.
- 이를 위해 요소가 Beomwoo인 것에 대한 boolean 값을 가지는 mask를 만들었고 마스크를 인덱싱에 응용하여 data의 0,1,5,7행을 꺼냈다.

01. 요소가 Kim인 행의 데이터만 꺼내기

```
data[names == 'Kim',:]

>> array([[-0.22633642, -0.76385264, 0.16368804, 0.91204438]])
```

02. 논리 연산을 응용하여, 요소가 Kim 또는 Park인 행의 데이터만 꺼내기

```
data[(names == 'Kim') | (names == 'Park'),:]

>> array([[-0.22633642, -0.76385264, 0.16368804, 0.91204438],
>>        [-0.05673648, -1.63408607, -2.29844338, -0.3662913 ]]
```

data array 자체적으로도 마스크를 만들고, 이를 응용하여 인덱싱이 가능

01. data array에서 0번째 열의 값이 0보다 작은 행을 구해보자.

```
data[:,0] < 0

>> array([ True, False, True, False, False, False, True, False]
```

02. 위에서 만든 마스크를 이용하여 0번째 열의 값이 0보다 작은 행을 구함

```
data[data[:,0]<0,:]

>> array([[-1.07099572, -0.85382063, -1.42474621, -0.05992846],
>>        [-0.22633642, -0.76385264, 0.16368804, 0.91204438],
>>        [-0.05673648, -1.63408607, -2.29844338, -0.3662913 ]]
```

이를 통해 특정 위치에만 우리가 원하는 값을 대입할 수 있다.

01. 위에서 얻은, 0번째 열의 값이 0보다 작은 행의 2,3번째 열값에 0을 대입해보자.

- 0번째 열의 값이 0보다 작은 행의 2,3번째 열 값

```
data[data[:,0]<0,2:4]

>> array([[-1.42474621, -0.05992846],
>>        [ 0.16368804, 0.91204438],
>>        [-2.29844338, -0.3662913 ]])
```

```
data[data[:,0]<0,2:4] = 0

>> array([[-1.07099572, -0.85382063, 0. , 0. ],
>>        [ 1.93097843, -1.8885167 , 1.99767454, 0.31524498],
>>        [-0.22633642, -0.76385264, 0. , 0. ],
>>        [ 1.34321923, 1.54532121, 0.28814921, 0.50688776],
>>        [ 0.4126606 , -0.52356522, 0.27124037, -0.66383264],
>>        [ 0.88575452, -1.39205929, 0.91019739, -1.04676349],
>>        [-0.05673648, -1.63408607, 0. , 0. ],
>>        [ 0.45963024, 0.35662128, 0.18307525, 1.46992167]])
```

05. Numpy 함수

numpy에서는 array에 적용되는 다양한 함수가 있다.

```
arr1 = np.random.randn(5,3)

>> array([[-1.28394941, -1.38235479, 0.3676742 ],
>>        [ 0.91707237, 0.45364032, 0.00683315],
>>        [ 0.51191795, 0.39014894, -0.15396686],
>>        [ 0.75541648, -3.0457677 , 0.83785171],
>>        [ 0.36609986, 1.2300834 , 0.51764117]])
```

01. 각 성분의 절대값 계산하기

```
np.abs(arr1)

>> array([[1.28394941, 1.38235479, 0.3676742 ],
>>        [0.91707237, 0.45364032, 0.00683315],
>>        [0.51191795, 0.39014894, 0.15396686],
>>        [0.75541648, 3.0457677 , 0.83785171],
>>        [0.36609986, 1.2300834 , 0.51764117]])
```

02. 각 성분의 제곱근 계산하기 (== array ** 0.5)

```
np.sqrt(arr1)

>> array([[ nan, nan, 0.60636144],
>>                [0.95763896, 0.67352826, 0.08266285],
>>                [0.71548442, 0.62461903, nan],
>>                [0.86914699, nan, 0.9153424 ],
>>                [0.60506187, 1.10909125, 0.71947284]])
```

03. 각 성분을 무리수 e의 지수로 삼은 값을 계산하기

```
np.exp(arr1)

>> array([[0.27694138, 0.25098684, 1.44437138],
>>        [2.50195487, 1.57403175, 1.00685655],
>>        [1.66848821, 1.47720079, 0.85730043],
>>        [2.12849782, 0.04755979, 2.3113961 ],
>>        [1.44209925, 3.42151487, 1.6780647 ]])
```

04. 각 성분을 자연로그, 상용로그, 밑이 2인 로그를 씌운 값을 계산하기

```
np.log(arr1)

>> array([[ nan, nan, -1.00055807],
>>        [-0.08656889, -0.79045064, -4.98596986],
>>        [-0.66959092, -0.94122672, nan],
>>        [-0.28048605, nan, -0.17691415],
>>        [-1.00484914, 0.20708197, -0.65847301]])
```

```
np.log10(arr1)

>> array([[ nan, nan, -0.43453685],
>>        [-0.03759639, -0.34328835, -2.1653792 ],
>>        [-0.29079964, -0.40876957, nan],
>>        [-0.12181354, nan, -0.07683284],
>>        [-0.43640043, 0.08993456, -0.28597119]])
```

```
np.log2(arr1)
>> array([[ nan, nan, -1.44350016],
>>        [-0.1248925 , -1.14037921, -7.193234 ],
>>        [-0.9660155 , -1.35790312, nan],
>>        [-0.40465583, nan, -0.25523316],
>>        [-1.44969086, 0.29875613, -0.94997574]])
```

05. 각 성분에 대해 삼각함수 값을 계산하기(cos, cosh, sin, sinh, tan, tanh)

```
np.cos(arr1)

>> array([[ 0.28292939, 0.18732825, 0.93316587],
>>        [ 0.60814678, 0.89885772, 0.99997665],
>>        [ 0.8718066 , 0.92485242, 0.9881705 ],
>>        [ 0.72798607, -0.9954123 , 0.66906099], [ 0.9337306 ,
```

```
np.tanh(arr1)

>> array([[-0.85753362, -0.88147747, 0.35195568],
>>        [ 0.72450949, 0.42488675, 0.00683304],
>>        [ 0.47143823, 0.37148862, -0.15276165],
>>        [ 0.63836924, -0.99548634, 0.68466951], [ 0.3505756 ,
```

06. 두 개의 array에 적용되는 함수

```
arr1 = np.random.randn(5,3)

>> array([[-1.28394941, -1.38235479, 0.3676742 ],
>>        [ 0.91707237, 0.45364032, 0.00683315],
>>        [ 0.51191795, 0.39014894, -0.15396686],
>>        [ 0.75541648, -3.0457677 , 0.83785171],
>>         [ 0.36609986, 1.2300834 , 0.51764117]])
```

```
arr2 = np.random.randn(5,3)

>> array([[-1.08072661, 0.49305711, -1.2341793 ],
>>        [ 0.72539264, -0.17482108, -2.29144412],
>>        [ 0.33676285, -0.44206124, 0.68359426],
>>        [-1.3367298 , -0.62530265, 0.04842608], [ 0.46567612,
```

01. 두 개의 array에 대해 동일한 위치의 성분끼리 연산 값을 계산하기 (add, subtract, multiply, divide)

```
np.multiply(arr1,arr2)

>> array([[ 1.3875983 , -0.68157986, -0.45377588],
>>        [ 0.66523755, -0.07930589, -0.01565778],
>>        [ 0.17239495, -0.17246972, -0.10525086],
>>        [-1.00978772, 1.90452663, 0.04057388],
>>        [ 0.17048396, -3.11221075, 0.41810686]])
```

02. 두 개의 array에 대해 동일한 위치의 성분끼리 비교하여 최대값 또는 최소값 계산하기(maximum, minimum)

```
np.maximum(arr1,arr2)

>> array([[-1.08072661, 0.49305711, 0.3676742 ],
>>        [ 0.91707237, 0.45364032, 0.00683315],
>>        [ 0.51191795, 0.39014894, 0.68359426],
>>        [ 0.75541648, -0.62530265, 0.83785171],
>>        [ 0.46567612, 1.2300834 , 0.80771562]])
```

07. 통계 함수

통계 함수를 통해 array의 합이나 평균등을 구할 때, 추가로 axis라는 인자에 대한 값을 지정하여 열 또는 행의 합 또는 평균등을 구할 수 있다.

```
arr1

>> array([[-1.28394941, -1.38235479, 0.3676742 ],
>>        [ 0.91707237, 0.45364032, 0.00683315],
>>        [ 0.51191795, 0.39014894, -0.15396686],
>>        [ 0.75541648, -3.0457677 , 0.83785171],
>>        [ 0.36609986, 1.2300834 , 0.51764117]])
```

01. 전체 성분의 합을 계산

```
np.sum(arr1)

>> 0.4883407866652476
```

02. 열 간의 합을 계산

```
np.sum(arr1, axis=1)

>> array([-2.29863 , 1.37754584, 0.74810003,
>>        -1.45249951, 2.11382443])
```

03. 행 간의 합을 계산

```
np.sum(arr1, axis=0)

>> array([ 1.26655726, -2.35424983, 1.57603336])
```

04. 전체 성분의 평균을 계산

```
np.mean(arr1)

>> 0.03255605244434984 4
```

05. 행 간의 평균을 계산

```
np.mean(arr1, axis=0)

>> array([ 0.25331145, -0.47084997, 0.31520667])
```

06. 전체 성분의 표준편차, 분산, 최소값, 최대값 계산(std, var, min, max)

```
np.std(arr1)

>> 1.0840662273348296
```

```
np.min(arr1, axis=1)

>> array([-1.38235479, 0.00683315, -0.15396686,
>>        -3.0457677 , 0.36609986])
```

07. 전체 성분의 최소값, 최대값이 위치한 인덱스를 반환(argmin, argmax)

```
np.argmin(arr1)

>> 10
```

```
np.argmax(arr1,axis=0)

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

08. 기타 함수

```
arr1

>> array([[-1.28394941, -1.38235479, 0.3676742 ],
>>        [ 0.91707237, 0.45364032, 0.00683315],
>>        [ 0.51191795, 0.39014894, -0.15396686],
>>        [ 0.75541648, -3.0457677 , 0.83785171],
>>        [ 0.36609986, 1.2300834 , 0.51764117]])
```

01. 전체 성분에 대해서 오름차순으로 정렬

```
np.sort(arr1)

>> array([[-1.38235479, -1.28394941, 0.3676742 ],
>>        [ 0.00683315, 0.45364032, 0.91707237],
>>        [-0.15396686, 0.39014894, 0.51191795],
>>        [-3.0457677 , 0.75541648, 0.83785171],
>>        [ 0.36609986, 0.51764117, 1.2300834 ]])
```

02. 전체 성분에 대해서 내림차순으로 정렬

```
np.sort(arr1)[::-1]

>> array([[ 0.36609986, 0.51764117, 1.2300834 ],
>>        [-3.0457677 , 0.75541648, 0.83785171],
>>        [-0.15396686, 0.39014894, 0.51191795],
>>        [ 0.00683315, 0.45364032, 0.91707237],
>>        [-1.38235479, -1.28394941, 0.3676742 ]])
```

03. 행 방향으로 오름차순으로 정렬

```
np.sort(arr1,axis=0)

>> array([[-1.28394941, -3.0457677 , -0.15396686],
>>        [ 0.36609986, -1.38235479, 0.00683315],
>>        [ 0.51191795, 0.39014894, 0.3676742 ],
>>        [ 0.75541648, 0.45364032, 0.51764117],
>>        [ 0.91707237, 1.2300834 , 0.83785171]])
```