



# Day44; 20221108

📅 날짜	@2022년 11월 8일
👤 유형	
☰ 태그	

GitHub - u8yes/AI

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<https://github.com/u8yes/ai>

**u8yes/AI**



1 Contributor 0 Issues 0 Stars 0 Forks

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/7bdde243-00d2-40da-826b-a99526bdf9f8/01\\_%EB%A8%B8%EC%8B%A0%EB%9F%A%C%EB%8B%9D\\_%EA%B0%9C%EB%85%90.pdf](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/7bdde243-00d2-40da-826b-a99526bdf9f8/01_%EB%A8%B8%EC%8B%A0%EB%9F%A%C%EB%8B%9D_%EA%B0%9C%EB%85%90.pdf)

처음부터 끝까지 한번에 실행해주는 아이콘



## Numpy ndarray 개요(nd: n차원, 예) 1차원, 2차원..)

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
```

### - 배열 생성

```
In [2]: # - 1차원 배열(Vector) 정의
array1 = np.array([1,2,3]) # series로 보여주는 것일뿐. 행렬구조인 matrix가 아니다. 절대 vector 1차원이다.
print(array1) # [1 2 3] # ', '가 없이 보여주는 것이 numpy
print('array1.type:', type(array1))
print('array1 array 형태:', array1.shape) # 행렬로 오해하지 말고 1차원이라 튜플 자료형 (3,) 으로 나온다. # 괄 3열을 의미
# shape은 차원을 결과로
[1 2 3]
array1.type: <class 'numpy.ndarray'>
array1 array 형태: (3,)
```

```
In [3]: # cf) 리스트 자료형
list = [1,2,3]
print(list)
print('list.type:', type(list)) # 파이썬에서 알려주는 list
[1, 2, 3]
list.type: <class 'list'>
```

```
In [4]: # - 2차원 배열(Matrix) 정의
array2 = np.array([[1,2,3],
                  [4,5,6]]) # 2[1행 3[,,]열
print(array2)
print('array2.type:', type(array2))
print('array2 array 형태:', array2.shape) # (2, 3) 2행 3열
[[1 2 3]
 [4 5 6]]
array2.type: <class 'numpy.ndarray'>
array2 array 형태: (2, 3)
```

```
In [5]: # - 2차원 배열(Matrix) 정의
array3 = np.array([[1,2,3]]) # 2차원 1행 3열 구조이다. 절대 1차원이 아니다. [] 갯수가 2개.
print(array3)
print('array3.type:', type(array3))
print('array3 array 형태:', array3.shape) # (1, 3) 1행 3열
[[1 2 3]]
array3.type: <class 'numpy.ndarray'>
array3 array 형태: (1, 3)
```

```
In [6]: # - 3차원 배열(Array) 정의
array4 = np.array([[[1,2,3,4], # 1면 시작, 1행
                   [5,6,7,8], # 2행
                   [9,10,11,12]], # 3행, 1면 끝
                   [[13,14,15,16], # 2면 시작
                   [17,18,19,20],
                   [21,22,23,24]]) # R은 행, 열, 면 Python은 면, 행, 열 순서!! 2면 3행 4열
print(array4) #
print('array4.type:', type(array4))
print('array4 array 형태:', array4.shape) # (2, 3, 4) 2면 3행 4열
[[[1 2 3 4]
  [5 6 7 8]
  [9 10 11 12]]

 [[13 14 15 16]
  [17 18 19 20]
  [21 22 23 24]]]
array4.type: <class 'numpy.ndarray'>
array4 array 형태: (2, 3, 4)
```

```
In [9]: print(f'array1: {array1.ndim}차원, array2: {array2.ndim}차원, array3: {array3.ndim}차원, array4: {array4.ndim}차원')
array1: 1차원, array2: 2차원, array3: 2차원, array4: 3차원
```

```
In [8]: list1 = [1,2,3]
print(type(list1))

array1 = np.array(list1)
print(type(array1))

print(array1, array1.dtype) # 할당되는 빙트크기 기준으로 int32 = 정수형(4byte=32bit) # int64 = long형

<class 'list'>
<class 'numpy.ndarray'>
[1 2 3] int32
```

```
In [9]: list2 = [1.,2.,3.]
print(type(list2))

array2 = np.array(list2)
print(type(array2))

print(array2, array2.dtype) # float64 = 실수형(double형) = 빙트크기 64비트

<class 'list'>
<class 'numpy.ndarray'>
[1. 2. 3.] float64
```

```
In [11]: list3 = [1, 2, 'test'] # 리스트는 어떤 자료형이라도 상관없다.
print(type(list3))

array3 = np.array(list3)
print(type(array3))

print(array3, array3.dtype) # ['1' '2' 'test'] 자료형을 강제로 문자열로 강제형변환시킴. # <U11 유니코드 타입

<class 'list'>
<class 'numpy.ndarray'>
['1' '2' 'test'] <U11
```

```
In [ ]:
```


---

```
In [12]: list4 = [1,2,3.5] # 0일 경우에는 '3.' 이라고 써도 3.0으로 인식됨
array4 = np.array(list4)
print(array4, array4.dtype) # [1. 2. 3.5] float64 # 전부 실수형으로 강제형변환 출력됨.

[1. 2. 3.5] float64
```

```
In [20]: # 정수 -> 실수(강제형변환)
array_int = np.array([1,2,3]) # int32
array_float = array_int.astype('float64') # as가 끝으면 강제형변환
print(array_float, array_float.dtype) # [1. 2. 3.] float64 실수형으로 출력
print(array_int, array_int.dtype) # [1 2 3] int32

# 실수 -> 정수(강제형변환)
array_int1 = array_float.astype('int32')
print(array_int1, array_int1.dtype)

# 실수 -> 정수(강제형변환)
array_float1 = np.array([1.1, 2.5, 3.7])
array_int2 = array_float1.astype('int32') # 데이터가 적은 타입으로 강제형변환을 하면 데이터 손실 가능성이 크다.
print(array_int2, array_int2.dtype) # [1 2 3] int32

[1. 2. 3.] float64
[1 2 3] int32
[1 2 3] int32
[1 2 3] int32
```

## ndarray를 편리하게 생성하기

```
In [27]: # zeros((행, 열)) : 0으로 채우는 함수 # Matrix 행렬구조로 만들어줌  
zero_array = np.zeros((3,4), dtype='int32') # (3,4)tuple로 채워줄 # int32로 하지 않으면 실수로 채워지게 됨  
print(zero_array)  
print(zero_array.dtype, zero_array.shape) # int32 (3, 4) 2차원 배열  
  
[[0 0 0 0]  
 [0 0 0 0]  
 [0 0 0 0]]  
int32 (3, 4)
```

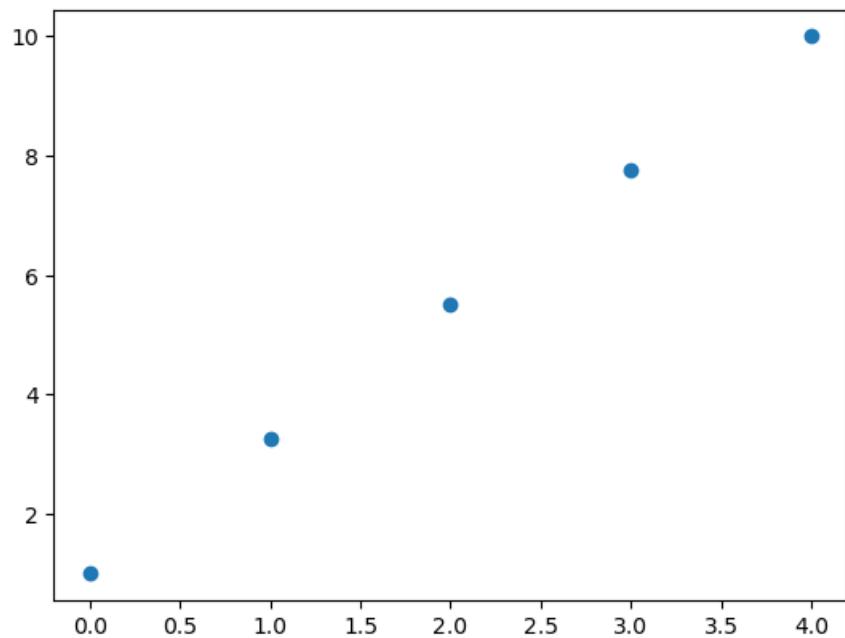
```
In [29]: # ones((행, 열)) : 1로 채우는 함수 # Matrix 행렬구조로 만들어줌  
one_array = np.ones((3,4)) # default값은 float64  
print(one_array)  
print(one_array.dtype, one_array.shape)  
  
[[1. 1. 1. 1.]  
 [1. 1. 1. 1.]  
 [1. 1. 1. 1.]]  
float64 (3, 4)
```

```
In [35]: # full((행, 열), 값) : 값으로 채우는 함수.  
arr_full = np.full((3,3), 8)  
print(arr_full)  
print(arr_full.dtype, arr_full.shape)  
  
[[8 8 8]  
 [8 8 8]  
 [8 8 8]]  
int32 (3, 3)
```

```
In [40]: # eye(N) : (N, N)의 단위 행렬 생성  
arr_eye = np.eye(3) # 3행 3열의 단위 행렬 출력  
print(arr_eye)  
[[1, 0, 0],  
 [0, 1, 0],  
 [0, 0, 1]]  
  
In [45]: # empty((행, 열)) : 초기화 없이 기존 메모리 값이 들어감  
arr_empty = np.empty((3,3))  
print(arr_empty)  
[[0.00000000e+000 0.00000000e+000 0.00000000e+000],  
 [0.00000000e+000 0.00000000e+000 6.99596955e-321],  
 [1.78019082e-306 1.37959740e-306 2.29178686e-312]]  
  
In [46]: # _like(배열) : 지정한 배열과 동일한 shape의 행렬을 만듦.  
# 종류 : np.zeros_like(), np.ones_like(), np.full_like(), np.empty_like()  
  
arr_sample = np.array([[1,2,3],  
                      [4,5,6]])  
arr_like = np.ones_like(arr_sample)  
print(arr_like)  
# [[1 1 1]]  
# [[1 1 1]]  
  
[[1 1 1]  
 [1 1 1]]
```

```
In [50]: # 배열 데이터 생성 할수  
# - np.linspace(시작, 종료, 개수) : 개수에 맞개끔 시작과 종료 사이에 균등하게 분배  
  
arr_linspace = np.linspace(1, 10, 5)  
print(arr_linspace) # [ 1.  3.25  5.5  7.75 10. ]
```

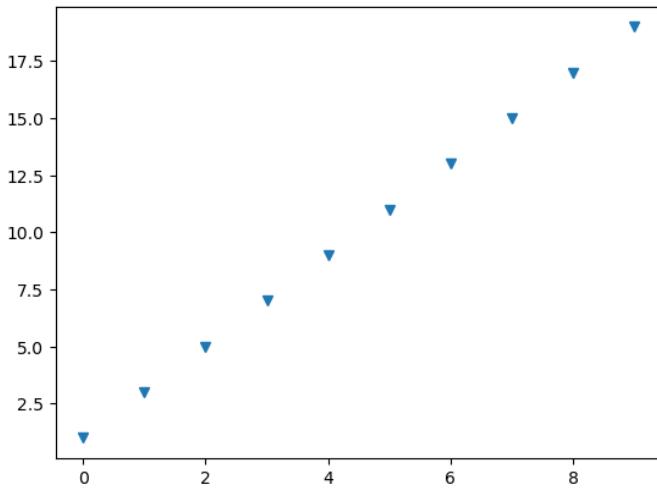
```
In [59]: plt.plot(arr_linspace, 'o')  
# 그래프를 그려주는 함수 마커를 원('o')으로 만든 그래프를 보여줄  
plt.show()
```



```
In [64]: # np.arange(시작, 종료, 스텝) : 시작과 종료 사이에 스텝 간격으로 생성() # array용으로 range를 사용하겠다 해서 'r' 하나가 빠짐  
# 파이썬 range() 함수와 유사  
  
sequence_array = np.arange(1,20,2) # 1차원 tuple로 돌아옴  
print(sequence_array)  
print(sequence_array.dtype, sequence_array.shape) # int32 (10,)
```

```
[ 1  3  5  7  9 11 13 15 17 19]  
int32 (10,)
```

```
In [79]: plt.plot(sequence_array, 'v')  
plt.show()
```



## ndarray의 차원과 크기를 변경하는 reshape()

```
In [86]: array1 = np.arange(10) # 1차원
print('array1:\n', array1)

array2 = array1.reshape(2, 5) # 2차원으로 변경해줌
print('array2:\n', array2)

array2 = array1.reshape(5, 2)
print('array2:\n', array2)
```

```
array1:
[0 1 2 3 4 5 6 7 8 9]
array2:
[[0 1 2 3 4]
 [5 6 7 8 9]]
array2:
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
```

```
In [88]: array1.reshape(4,3) # 지정된 사이즈로 변경이 불가능하면 오류 발생.
```

```
-----  
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6588\3924263243.py in <module>
----> 1 array1.reshape(4,3) # 지정된 사이즈로 변경이 불가능하면 오류 발생.

ValueError: cannot reshape array of size 10 into shape (4,3)
```

```
In [94]: array1 = np.arange(10)
print(array1)

array2 = array1.reshape(-1, 5) # 암수에 포커스를 맞추자. 행의 갯수는 상관없다. 열의 갯수만 5개로 기준으로 해서 행은 자동 생성됨.
print(array2)
print('array2 shape', array2.shape) # 2행 5열 생성

array3 = array1.reshape(5, -1) # 암수에 포커스를 맞추자. 열의 갯수는 상관없다. 행의 갯수만 5개로 기준으로 해서 열은 자동 생성됨.
print(array3)
print('array3 shape', array3.shape) # 5행 2열 생성
```

```
[0 1 2 3 4 5 6 7 8 9]
[[0 1 2 3 4]
 [5 6 7 8 9]]
array2 shape (2, 5)
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
array3 shape (5, 2)
```

```
In [98]: array1 = np.arange(10)
array4 = array1.reshape(-1, 4) # 고정된 4개의 컬럼을 가진 row 변경 불가
```

```
-----  
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6588\2730013190.py in <module>
----> 1 array1 = np.arange(10)
      2 array4 = array1.reshape(-1, 4) # 고정된 4개의 컬럼을 가진 row 변경 불가

ValueError: cannot reshape array of size 10 into shape (4)
```

```
In [107]: array1 = np.arange(8)
array3d = array1.reshape((2,2,2))
print('array3d:\n', array3d.tolist())

# 3차원 ndarray를 2차원 ndarray로 변환 # 2차원만이 알고리즘이 작동할 수 있다.
# 딥러닝 때 중요!
array5 = array3d.reshape(-1, 1) # 열의 나열이 중요한 것이 아니다. 차원을 축소한 것이 중요하다.
print('array5:\n', array5.tolist())
print('array5 shape: ', array5.shape) # (8, 1)

# 1차원 ndarray를 2차원 ndarray로 변환
array6 = array1.reshape(-1, 1)
print('array6:\n', array6.tolist())
print('array6 shape: ', array6.shape)

array3d:
[[[0, 1], [2, 3]], [[4, 5], [6, 7]]]
array5:
[[0], [1], [2], [3], [4], [5], [6], [7]]
array5 shape: (8, 1)
array6:
[[0], [1], [2], [3], [4], [5], [6], [7]]
array6 shape: (8, 1)
```

## indexing - 넘파이 ndarray의 데이터 세트 선택하기\_20221108

- 단일 값 추출

```
In [121]: # 10에서부터 9까지의 1차원 ndarray 생성  
array1 = np.arange(start=1, stop=13)  
print('array1:', array1)  
  
value = array1[2]  
print('value:', value)  
print(type(value))
```

```
array1: [ 1  2  3  4  5  6  7  8  9 10 11 12]  
value: 3  
<class 'numpy.int32'>
```

```
In [125]: print('맨 뒤의 값:', array1[-1], '맨 뒤에서 두번째 값:', array1[-2])  
맨 뒤의 값: 12  
맨 뒤에서 두번째 값: 11
```

```
In [128]: # 데이터 값 수정  
array1[0] = 8  
array1[8] = 0  
print('array1:', array1)  
  
array1: [ 8  2  3  4  5  6  7  8  0 10 11 12]
```

```
In [137]: # 2차원 ndarray에서 단일 값 추출
array1d = np.arange(1, 10)
array2d = array1d.reshape(3,3)
print(array2d)

print('(row=0, col=0) index 가리키는 값:', array2d[0,0])
print('(row=0, col=0) index 가리키는 값:', array2d[0,1])
print('(row=0, col=0) index 가리키는 값:', array2d[1,2])
print('(row=0, col=0) index 가리키는 값:', array2d[2,2])
print('(row=0, col=0) index 가리키는 값:', array2d[3,2])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
(row=0, col=0) index 가리키는 값: 1
(row=0, col=0) index 가리키는 값: 2
(row=0, col=0) index 가리키는 값: 6
(row=0, col=0) index 가리키는 값: 9
```

```
IndexError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6588\531286027.py in <module>
  8 print('(row=0, col=0) index 가리키는 값:', array2d[1,2])
  9 print('(row=0, col=0) index 가리키는 값:', array2d[2,2])
--> 10 print('(row=0, col=0) index 가리키는 값:', array2d[3,2])
```

**IndexError:** index 3 is out of bounds for axis 0 with size 3

- slicing

```
: array1 = np.arange(1, 10)
array2 = array1[0:3]
print(array2)
print(type(array2)) # slicing은 ndarray
```

```
[1 2 3]
<class 'numpy.ndarray'>
```

pandas 자격증)

```
In [171]: # [2-59] pivot_table을 사용하여 대륙별(columns), '맥주'와 '와인'의 mean, median값을 구합니다.
# 결과 참조
df.pivot_table(columns='대륙', values=['맥주', '와인'], aggfunc=['mean', 'median'])
```

대륙	mean						median					
	AF	AS	EU	NA	OC	...	AS	EU	NA	OC	SA	
맥주	61.471698	37.045455	193.777778	145.434783	89.6875	...	17.5	219.0	143.0	52.5	162.5	
와인	16.264151	9.068182	142.222222	24.521739	35.6250	...	1.0	128.0	11.0	8.5	12.0	
2 rows × 12 columns												

```
In [147]: # 슬라이싱 기호인 ':' 사이의 시작, 종료 인덱스는 생략 가능.
```

```
array1 = np.arange(1, 10)
array3 = array1[:3] # tuple # R과 동일 컨셉
print(array3)

array4 = array1[3:]
print(array4)

array5 = array1[:] # 전체 데이터
print(array5)
```

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

```
In [156]: # 2차원 ndarray에서 슬라이싱으로 데이터 접근.
```

```
array1d = np.arange(1, 10)
array2d = array1d.reshape(3, 3)
print('array2d:\n', array2d)

print('array2d[0:2, 0:2]\n', array2d[0:2, 0:2]) # 0~1행, 0~1열
print('array2d[1:3, 0:3]\n', array2d[1:3, 0:3]) # 1~2행, 0~2열
print('array2d[1:3, :]\n', array2d[1:3, :])
print('array2d[:, 1:]\n', array2d[:, 1:])
print('array2d[:, 0]\n', array2d[:, 0])
```

```
array2d:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
array2d[0:2, 0:2]
[[1 2]
 [4 5]]
array2d[1:3, 0:3]
[[4 5 6]
 [7 8 9]]
array2d[1:3, :]
[[4 5 6]
 [7 8 9]]
array2d[:, 1:]
[[2 3]
 [5 6]]
array2d[:, 0]
[1 4]
```

```
# 2차원 ndarray에서 뒤에 오는 인덱스를 없애면 1차원 ndarray를 반환.  
print(array2d[0]) # 1차원 1행을 출력 # row축(axis 0)의 첫번째 row ndarray(1차원)를 반환. # 3차원은 열, 행, 열 순으로  
print(array2d[1]) # 1차원 2행을 출력  
print('array2d[0] shape:', array2d[0].shape, 'array2d[1] shape:', array2d[1].shape)  
[1 2 3]  
[4 5 6]  
array2d[0] shape: (3,) array2d[1] shape: (3,)
```

## fancy

미국·영국['fænsi] ↗ 영국식 ↗

(동사)

1 원하다, ...하고 싶다 (=feel like)

Fancy a drink? ↗

한잔 하겠나?

2 (성적으로) 끌리다[반하다]

I think she fancies me. ↗

그녀가 나한테 반한 것 같아.

(명사)

1 공상, 상상 (=fantasy)

night-time fancies that disappear in the morning ↗

아침이면 사라지는 밤 동안의 공상들

2 바람, 욕망 (=whim)

She said she wanted a dog but it was only a passing fancy. ↗

그녀는 개를 갖고 싶다고 했지만 그것은 일시적인 바람일 뿐이었다.

영어사전 다른 뜻 2

- fancy indexing

```
In [16]: array1d = np.arange(1, 10)
array2d = array1d.reshape(3,3)
print(array2d)

array3 = array2d[[0,1], 2]
print('array2d[[0,1], 2] => ', array3.tolist())

array4 = array2d[[0,1], 0:2]
print('array2d[[0,1], 0:2] => ', array4.tolist())

array5 = array2d[[0,1]]
print('array2d[[0,1]] => ', array5.tolist())

[[1 2 3]
 [4 5 6]
 [7 8 9]]
array2d[[0,1], 2] => [3, 6]
array2d[[0,1], 0:2] => [[1, 2], [4, 5]]
array2d[[0,1]] => [[1, 2, 3], [4, 5, 6]]
```

- Boolean indexing : 조건 필터링과 검색을 동시에 할 수 있기 때문에 자주 사용.

```
In [169]: arrayId = np.arange(1,10) # 1~9까지
array3 = arrayId[arrayId > 5] # []안에 arrayId > 5 : Boolean indexing을 적용
print('arrayId > 5 부울린 인덱싱 결과 값 : ', array3)

arrayId > 5 부울린 인덱싱 결과 값 : [6 7 8 9]

In [172]: arrayId > 5
Out[172]: array([False, False, False, False, False, True, True, True, True])

In [178]: boolean_indexes = np.array([False, False, False, False, False, True, True, True, True])
print(boolean_indexes)

array3 = arrayId[boolean_indexes]
print('불린 인덱스로 필터링 결과:',array3)

[False False False False True True True True]
불린 인덱스로 필터링 결과: [6 7 8 9]

In [179]: indexes = np.array([5,6,7,8])
array4 = arrayId[indexes]
print('일반 인덱스로 필터링 결과:',array4)

일반 인덱스로 필터링 결과: [6 7 8 9]
```

## 행렬의 정렬-sort() argsort()

- 행렬 정렬

```
In [204]: org_array = np.array([3,1,9,5])
print('원본 행렬 : ', org_array)

# np.sort()로 정렬
sort_array1 = np.sort(org_array)
print('np.sort() 호출 후 반환된 정렬 행렬:', sort_array1)
print('np.sort() 이전 원본 행렬:', org_array)

# ndarray.sort()로 정렬
sort_array2 = org_array.sort()
print('org_array.sort() 호출 후 반환된 정렬 행렬:', sort_array2) # 자체적으로 업데이트를 원분까지 해주기 때문에 None으로 나온다.
print('org_array.sort() 호출 전 원본 행렬:', org_array)
```

원본 행렬 : [3 1 9 5]  
np.sort() 호출 후 반환된 정렬 행렬: [1 3 5 9]  
np.sort() 이전 원본 행렬: [3 1 9 5]  
org\_array.sort() 호출 후 반환된 정렬 행렬: None  
org\_array.sort() 호출 전 원본 행렬: [1 3 5 9]

```
In [208]: sort_array1_desc = np.sort(org_array)[::-1]
print('내림차순으로 정렬:', sort_array1_desc)
```

내림차순으로 정렬: [9 5 3 1]

```
In [214]: array2d = np.array([[8,12],
[7,1]])

sort_array2d_axis0 = np.sort(array2d, axis = 0) # 행끼리 비교 # 1행과 2행끼리 비교
print('row 방향으로 정렬:', sort_array2d_axis0)

sort_array2d_axis1 = np.sort(array2d, axis = 1) # 열끼리 비교 # 1열과 2열끼리 비교
print('column 방향으로 정렬:', sort_array2d_axis1)

row 방향으로 정렬:
[[ 7  1]
 [ 8 12]]
column 방향으로 정렬:
[[ 8 12]
 [ 1  7]]
```

- 정렬 행렬의 인덱스 반환

```
In [223]: org_array = np.array([3,1,9,5])
print(org_array)
sort_index = np.argsort(org_array) # 오름차순으로 인덱스 순서를 알려줌.
print('행렬 정렬 시 원본 행렬의 인덱스:', sort_index, type(sort_index))

[3 1 9 5]
행렬 정렬 시 원본 행렬의 인덱스: [1 0 3 2] <class 'numpy.ndarray'>
```

```
In [224]: org_array = np.array([3,1,9,5])
print(org_array)
sort_index_desc = np.argsort(org_array)[::-1]
print('내림차순 정렬 시 원본 행렬의 인덱스:', sort_index_desc)

[3 1 9 5]
내림차순 정렬 시 원본 행렬의 인덱스: [2 3 0 1]
```

```
In [228]: name_array = np.array(['John', 'Mike', 'Sarah', 'Kate', 'Samuel'])
score_array = np.array([78, 95, 84, 98, 88])

sort_index_asc = np.argsort(score_array)
print('성적 오름차순 정렬 시 score_array의 인덱스:', sort_index_asc)

[0 2 4 1 3]
성적 오름차순 정렬 시 score_array의 인덱스: [0 2 4 1 3]
```

## 선형대수 연산

- list vs ndarray

```
In [233]: x1 = [1,2,3]
x2 = [4,5,6]
x3 = x1 + x2

print(x3)
print(type(x3))
```

```
[1, 2, 3, 4, 5, 6]
<class 'list'>
```

```
In [234]: x4 = np.array([1,2,3])
x5 = np.array([4,5,6])

x6 = x4 + x5 # [5 7 9]

print(x6)
print(type(x6))
```

```
[5 7 9]
<class 'numpy.ndarray'>
```

---

```
In [237]: x = np.array([10,11,12])

for i in np.arange(1,4):
    print(i, '각각 더해줌')
    print(i + x)
```

```
1 각각 더해줌
[11 12 13]
2 각각 더해줌
[12 13 14]
3 각각 더해줌
[13 14 15]
```

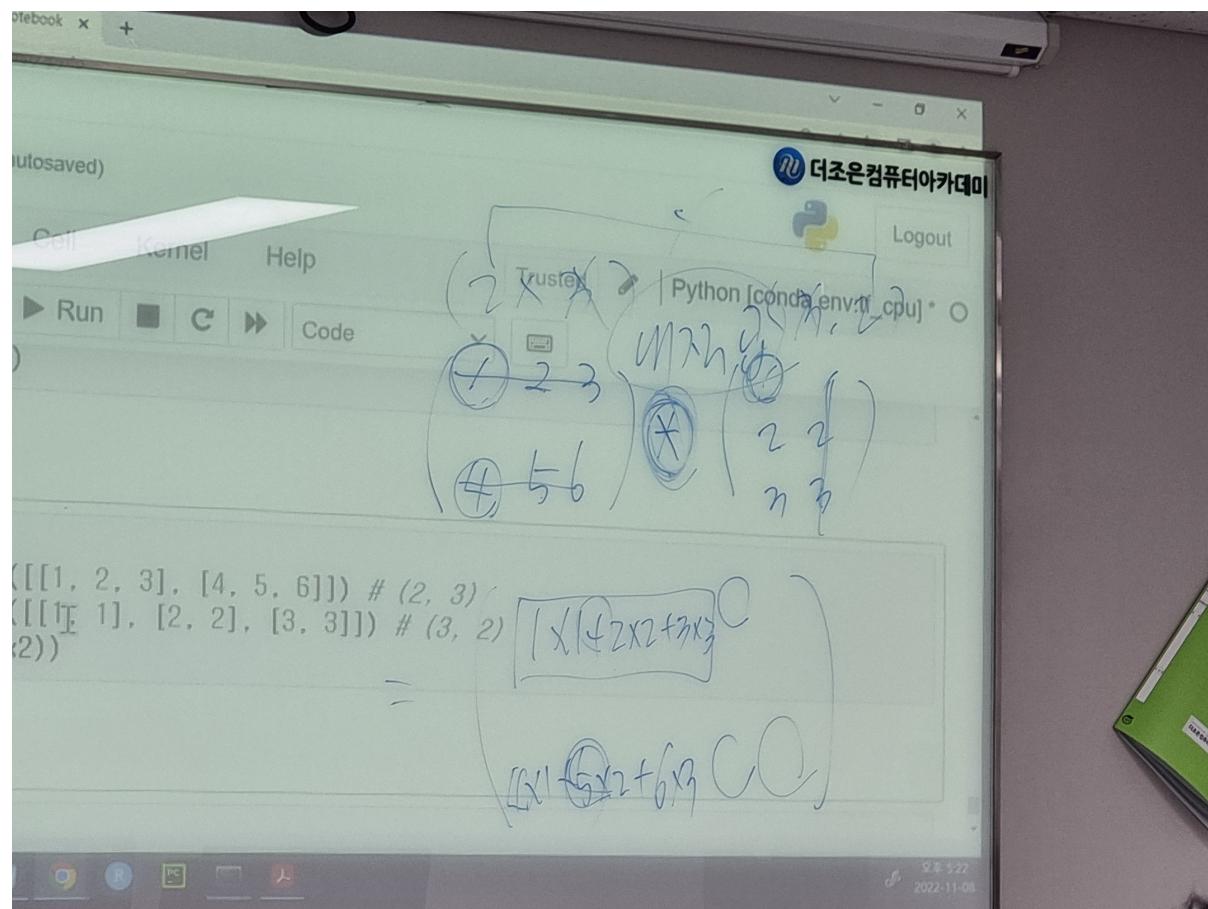
```
In [240]: # 2차원 행렬(numpy.ndarray)의 사칙 연산
# - 덧셈
x1 = np.array([[4,4,4],
               [8,8,8]])
x2 = np.array([[1,1,1],
               [2,2,2]])

print(x1 + x2)
```

```
[[ 5  5  5]
 [10 10 10]]
```

```
In [243]: # - 스칼라 x 행렬
x = np.array([[4,4,4],
               [8,8,8]])
scalarm = 10 * x
print(scalar)
```

```
[[40 40 40]
 [80 80 80]]
```



내적곱)  $(1 \times 1 + 2 \times 2 + 3 \times 3) + (4 \times 1 + 5 \times 2 + 6 \times 3) + (1 \times 1 + 2 \times 2 + 3 \times 3) + (4 \times 1 + 5 \times 2 + 6 \times 3) = 14 + 14 + 32 + 32$ 가 나오게 됨.

```
In [247]: # 행렬 x 행렬
x1 = np.array([[1,2],
               [3,4]]) # 2행 2열
x2 = np.array([[1,1],
               [2,2]]) # 2행 2열
print(x1 * x2)
```

[[1 2]  
[6 8]]

```
In [248]: # 행렬 x 행렬
x1 = np.array([[1,2,3],
               [4,5,6]]) # 2행 3열
x2 = np.array([[1,1],
               [2,2],
               [3,3]]) # 3행 2열
print(x1.dot(x2))
```

[[14 14]  
[32 32]]

- 전치 행렬(행렬을 바꾼 것이 전치 행렬)

```
In [25]: A = np.array([[1,2,3],  
                  [4,5,6]])  
  
transpose_mat = np.transpose(A)  
print('A의 전치 행렬:', transpose_mat)
```

A의 전치 행렬:

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

```
In [29]: data = np.array([[51,52],[53,54],[55,56]]) # 3행 2열  
print(data)  
print(data[2,0]) # R의 형태와 똑같이  
print(data[2][0]) # JAVA의 형태와 똑같이 해도 가능해짐
```

```
[[51 52]  
 [53 54]  
 [55 56]]  
55  
55
```

```
In [31]: y = data.flatten() # 1차원으로 만들 # data를 1차원 배열로 변환(평坦화)  
print(y)
```

```
[51 52 53 54 55 56]
```

## - Numpy에서 np.sum 함수의 axis의 이해\_20221108

```
In [34]: arr = np.arange(0,24)
print(len(arr))
print(arr)

24
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]

In [37]: v = arr.reshape([2,3,4]) # 차원변환([z축(depth), x축(row), y축(column)])
v

Out[37]: array([[[ 0,  1,  2,  3],
   [ 4,  5,  6,  7],
   [ 8,  9, 10, 11]],

  [[12, 13, 14, 15],
   [16, 17, 18, 19],
   [20, 21, 22, 23]]])

In [46]: print(v.shape) # 면, 행, 열
print(v.ndim) # 차원
print(v.sum())

(2, 3, 4)
3
276

In [48]: res01 = v.sum(axis = 0) # z축 면을 기준으로 sum해줘라.
print(res01.shape)
print(res01)

(3, 4)
[[12 14 16 18]
 [20 22 24 26]
 [28 30 32 34]]
```

