



1 파이썬 데이터분석 특화 데이터 형태: [Numpy](#)

- **Numpy:** Numerical Python의 약자로 수치를 다루는 분야를 위한 파이썬 패키지
 - 데이터분석 패키지 중 하나로 고차원 데이터 연산(선형대수학)과 컴퓨팅 연산(자료구조)에 특화
 - 대표적으로 사용되는 표준 수학 함수기능 지원
 - 저차원 및 고차원 데이터 입출력 및 산술연산
 - Array 또는 Matrix와 같은 자료구조 수치를 다루는 Python 데이터형태
 - **Array:** numpy.array함수로 변환된 다차원의 데이터 형태(모든 값은 같은 자료형)

1.1 설치 및 호출

```
In [1]: # 주피터 노트북에서 Numpy 설치
# Anaconda 설치하면 가설되어 있어서 진행할 필요 없음
!pip install numpy
```

executed in 2.46s, finished 14:09:52 2021-11-18

Requirement already satisfied: numpy in c:\Users\KK\anaconda3\lib\site-packages (1.19.5)

WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.

You should consider upgrading via the 'C:\Users\KK\anaconda3\python.exe -m pip install --upgrade pip' command.

```
In [2]: # Numpy 사용을 위해 패키지를 불러오기
# import 패키지명 as 뒤네임
# 관례적으로 np라는 약자를 많이 사용
import numpy as np
```

executed in 104ms, finished 14:09:52 2021-11-18

1.2 List vs Array

- List를 사용해서 다차원 데이터를 구현할 수 있는데 Numpy ndarray로 다차원을 다루는 이유는?
 - 유사점: List의 다차원이나 Numpy ndarray의 다차원 데이터의 출력은 (거의)같음

```
In [3]: import numpy as np
a = [1,2,3,4,5]
b = np.array(a)
```

executed in 14ms, finished 14:09:52 2021-11-18

```
In [4]: print(a, b)
```

executed in 14ms, finished 14:09:52 2021-11-18

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

```
In [5]: a
```

executed in 14ms, finished 14:09:52 2021-11-18

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

```
In [6]: b
```

executed in 14ms, finished 14:09:52 2021-11-18

Out[6]: array([1, 2, 3, 4, 5])

```
In [7]: print(type(a), type(b))
```

executed in 13ms, finished 14:09:52 2021-11-18

<class 'list'> <class 'numpy.ndarray'>

- 차이점: 고차원이거나 빅데이터의 경우 Numpy array 데이터로 저장된 경우 연산에 유리함

List	Array
여러 Type의 값을 가질 수 있음	동일한 Type의 값만 가질 수 있음
메모리를 많이 점유하고 계산 속도가 느림	메모리를 최적으로 쓰고 계산 속도 향상
각 값마다 연산 필요 -> 반복문 필요	벡터화(값의 묶음)로 연산 가능 -> 반복문 불필요

- array를 사용하면 list에서 사용하지 못했던 콤마(,)기반으로 인덱싱과 슬라이싱 가능
- array를 사용하면 정수 또는 불리언 형식으로 값을 선택가능

```
In [8]: # List 예시
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]*2
a
executed in 14ms, finished 14:09:52 2021-11-18
Out[8]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [9]: # Array 예시
a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])*2
a
executed in 14ms, finished 14:09:52 2021-11-18
Out[9]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])

In [10]: # List 예시
a = [[0, 1, 2], [3, 4, 5]]
a[0][1:]
executed in 14ms, finished 14:09:52 2021-11-18
Out[10]: [1, 2]

In [11]: # Array 예시
a = np.array([[0, 1, 2], [3, 4, 5]])
a[0,1:]
executed in 14ms, finished 14:09:52 2021-11-18
Out[11]: array([1, 2])

In [12]: # List 예시
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
a[True]
executed in 14ms, finished 14:09:52 2021-11-18
Out[12]: 1

In [13]: # Array 예시
a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
a[True]
executed in 14ms, finished 14:09:52 2021-11-18
Out[13]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [14]: # # List 예시
# a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# a[[True, False, True, False, True, False, True, False, True, False]]
executed in 14ms, finished 14:09:53 2021-11-18
-----  

TypeError                                 Traceback (most recent call last)
<ipython-input-45-a0b1051558de> in <module>
      1 a[[True, False, True, False, True, False, True, False, True, False]]
      2
----> 3 TypeError: list indices must be integers or slices, not list
```

```
In [15]: # Array 예시
a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
a[[True, False, True, False, True, False, True, False, True, False]]
executed in 13ms, finished 14:09:53 2021-11-18
Out[15]: array([0, 2, 4, 6, 8])
```

- 정리:

- 브로드캐스팅 기능으로 형태가 다른 저차원과 고차원간 연산 가능
- 행렬(저차원 및 고차원)을 다루는 선형대수 및 변환기능 지원
- 빅데이터 연산에 효율성이 높음
- 다른 프로그래밍 언어에서 존재하지 않음

▼ 1.3 Numpy Array(입출력)

▼ 1.3.1 Array 값 생성하기

- numpy 연산의 기본이 되는 데이터 구조로 Array 또는 배열이라고 함
- 리스트보다 간편하게 만들 수 있으며 연산이 빠른 장점
- 같은 type의 데이터만 저장 가능
- 다른 차원간 연산인 브로드캐스팅 연산 지원
- array 또한 numpy의 기본 함수로서 생성 가능
- 사용법:

```
np.array(배열변환 대상 데이터) # 리스트와 같은 순서가 있는 데이터를 받아 Numpy 배열 데이터 생성
np.arange(start, end, step_forward)
np.linspace(start, end, number)
```

```
In [16]: # 기존 리스트 데이터 구조를 array로 변환
test_list = [1,2,3,4]
test_array = np.array(test_list)
executed in 14ms, finished 14:09:53 2021-11-18
```

```
In [17]: # array 생성 확인
test_array
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[17]: array([1, 2, 3, 4])
```

```
In [18]: # 같은 타입의 데이터만 들어가는지 확인
# 문자열이 들어가면 가장 큰 범주의 데이터타입으로 저장
test_list = [1, 2, 3.14]
executed in 14ms, finished 14:09:53 2021-11-18
```

```
In [19]: np.array(test_list)
executed in 13ms, finished 14:09:53 2021-11-18
```

```
Out[19]: array([1. , 2. , 3.14])
```

```
In [20]: # np.arange 함수로 생성
np.arange(1, 11)
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[20]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [21]: # 범위설정
# (시작포인트, 마지막포인트 + 1, 스텝)
# range() 함수와 동일하게 작동함
# for i in range(1, 100, 10):
#     print(i)
np.arange(1, 100, 10)
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[21]: array([ 1, 11, 21, 31, 41, 51, 61, 71, 81, 91])
```

```
In [22]: # 범위 데이터 생성 (시작포인트, 마지막포인트, 데이터갯수)
np.linspace(0, 10, 50)
executed in 12ms, finished 14:09:53 2021-11-18
```

```
Out[22]: array([ 0.          ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
 1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
 2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
 3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
 4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
 5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
 6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
 7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
 8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
 9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

- 특수한 형태의 array도 numpy로 생성 가능
- 입력 데이터는 정수, 리스트, 튜플만 입력 가능
- ndarray**: n-dimensional array

함수	내용
np.array(데이터)	입력된 데이터를 ndarray로 변환. dtype을 명시하면 자료형을 설정할 수 있다
np.asarray(데이터)	입력 데이터를 ndarray로 변환하나 이미 ndarray일 경우에는 새로 메모리에 ndarray가 생성되지는 않는다
np.arange(데이터)	range 함수와 유사하나 ndarray를 반환
np.ones(데이터)	전달인자로 전달한 dtype과 모양(행,렬)으로 배열을 생성하고 모든 내용을 1로 초기화하여 ndarray를 반환
np.zeros(데이터)	ones와 같으나 초기값이 0이다
np.empty(데이터)	ones와 zeros와 비슷하나 값을 초기화하지는 않는다

```
In [23]: # 벡터 생성
import numpy as np

a1d = np.array([1,2,3])
a1d
executed in 13ms, finished 14:09:53 2021-11-18
```

```
Out[23]: array([1, 2, 3])
```

```
In [24]: # 배열 형태 확인
type(a1d)
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[24]: numpy.ndarray
```

```
In [25]: # 배열 내부 값들의 형태 확인
a1d.dtype
executed in 15ms, finished 14:09:53 2021-11-18
```

```
Out[25]: dtype('int32')
```

```
In [26]: # 2차원 배열(행렬) 생성
# 행렬은 행과 열로 이루어진 2차원 배열이다.
```

```
b2d = np.array([1, 2, 3, 4, 5, 6]) # 2 배열 형태 확인
executed in 14ms, finished 14:09:53 2021-11-18
Out[26]: array([[1, 2, 3],
 [4, 5, 6]])

In [27]: # 배열 형태 확인
type(b2d)
executed in 14ms, finished 14:09:53 2021-11-18
Out[27]: numpy.ndarray

In [28]: # 구조 확인
b2d.shape
executed in 14ms, finished 14:09:53 2021-11-18
Out[28]: (2, 3)

In [29]: c3d = np.array([[1,2,3], [4,5,6]])
c3d
executed in 14ms, finished 14:09:53 2021-11-18
Out[29]: array([[1, 2, 3],
 [4, 5, 6]])

In [30]: # 구조 확인
c3d.shape
executed in 14ms, finished 14:09:53 2021-11-18
Out[30]: (1, 2, 3)

In [31]: c3d = np.array([[1,2,3], [4,5,6],
 [[10,20,30], [40,50,60]]])
c3d
executed in 14ms, finished 14:09:53 2021-11-18
Out[31]: array([[1, 2, 3],
 [4, 5, 6],
 [[10, 20, 30],
 [40, 50, 60]]])

In [32]: # 구조 확인
c3d.shape
executed in 14ms, finished 14:09:53 2021-11-18
Out[32]: (2, 2, 3)

In [33]: # 함수 사용해서 구조(모양)확인
np.shape(c3d)
executed in 14ms, finished 14:09:53 2021-11-18
Out[33]: (2, 2, 3)

In [34]: # 고유값 확인
# len 써워주시면 고유값 갯수까지
len(np.unique(c3d))
executed in 14ms, finished 14:09:53 2021-11-18
Out[34]: 12

In [35]: # 타입변환
test = [10,20,30]
type(test)
executed in 14ms, finished 14:09:53 2021-11-18
Out[35]: list

In [36]: test2 = np.asarray(test)
type(test2)
executed in 14ms, finished 14:09:53 2021-11-18
Out[36]: numpy.ndarray

In [37]: print(test, test2)
executed in 14ms, finished 14:09:53 2021-11-18
[10, 20, 30] [10 20 30]

In [38]: # 연속적인 시퀀스(Sequence) 배열 생성
np.arange(1,20)
executed in 14ms, finished 14:09:53 2021-11-18
Out[38]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
 18, 19])

In [39]: # 0인 값만 보유한 행렬 생성
a2d = np.zeros((3,3))
a2d
executed in 14ms, finished 14:09:53 2021-11-18
Out[39]: array([[0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.]])
```

```
In [40]: # 1인 값만 보유한 행렬 생성
b2d = np.ones((2,3))
b2d
executed in 14ms, finished 14:09:53 2021-11-18
Out[40]: array([[1., 1., 1.],
 [1., 1., 1.]])
```

```
In [41]: # 특정 값만 보유한 행렬 생성
c2d = np.full((3,3), 7)
c2d
executed in 14ms, finished 14:09:53 2021-11-18
Out[41]: array([[7, 7, 7],
 [7, 7, 7],
 [7, 7, 7]])
```

```
In [42]: # 대각행렬이라고도 하는데 대각선으로 1인 값만 존재
d2d = np.eye(3)
d2d
executed in 14ms, finished 14:09:53 2021-11-18
Out[42]: array([[1., 0., 0.],
 [0., 1., 0.],
 [0., 0., 1.]])
```

```
In [43]: # 임의의 값으로 행렬을 생성
e2d = np.random.random((2,2))
e2d
executed in 14ms, finished 14:09:53 2021-11-18
Out[43]: array([0.44469493, 0.86127287],
 [0.8149375 , 0.81034536])
```

```
In [44]: # 임의의 값으로 행렬을 생성하는데 값이 고정된 경우
f2d = np.empty((4, 5), dtype=int)
f2d
executed in 13ms, finished 14:09:53 2021-11-18
Out[44]: array([[ 35,     32, 48176, 50676,     32],
 [54805, 53468,     32, 54869, 51064],
 [   10,    116,    121,    112,    101],
 [   40,     98,     50,    100,     41]])
```

```
In [45]: # 세부 값의 type 변환
np.array(np.random.random((2,2)), dtype=np.float64)
executed in 14ms, finished 14:09:53 2021-11-18
Out[45]: array([0.74679353, 0.41958635],
 [0.32025615, 0.33536631])
```

```
In [46]: # 세부 값의 type 변환
np.array(np.random.random((2,2)), dtype=np.complex)
executed in 14ms, finished 14:09:53 2021-11-18
Out[46]: array([[0.9930886 +0.j, 0.04065359+0.j],
 [0.25772073+0.j, 0.9417365 +0.j]])
```

```
In [47]: # 세부 값의 type 변환
np.array(np.random.random((2,2)), dtype=np.int)
executed in 14ms, finished 14:09:53 2021-11-18
Out[47]: array([[0, 0],
 [0, 0]])
```

```
In [48]: # 세부 값의 type 변환
np.array(np.random.random((2,2)), dtype=np.bool)
executed in 14ms, finished 14:09:53 2021-11-18
Out[48]: array([[ True,  True],
 [ True,  True]])
```

```
In [49]: # 세부 값의 type 변환
np.array(np.random.random((2,2)), dtype=np.object)
executed in 14ms, finished 14:09:53 2021-11-18
Out[49]: array([[0.7609211206738068, 0.9254126131263817],
 [0.13369668111295585, 0.6822665518761526]], dtype=object)
```

1.3.2 Array 값 다루기

"데이터 분석을 하다보면 데이터 각각의 값들에 접근해야 하는 경우 종종 발생"

- 인덱싱(Indexing): 데이터 값 하나에 접근하는 방법으로 인덱스는 값의 순번
- 슬라이싱(Slicing): 인덱싱과 유사하여 데이터 값을 여러가지 방식으로 접근할 수 있는 유니버설 방법

- 다차원 데이터의 경우 배열의 각 차원에 대해 슬라이싱 사용
- 슬라이싱으로 값을 선택하면 결과의 차원은 원래의 차원보다 같거나 작음

```
In [50]: # 시퀀스 '벡터' 생성
a = np.arange(10)
a
```

```
executed in 14ms, finished 14:09:53 2021-11-18
Out[50]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [51]: # 5번째 값을 반환(0부터 시작해서 5번째)
a[5]
executed in 14ms, finished 14:09:53 2021-11-18
Out[51]: 5

In [52]: # 변수[시작인덱스 이상: 끝인덱스 미만:간격]
a[5:8]
executed in 14ms, finished 14:09:53 2021-11-18
Out[52]: array([5, 6, 7])

In [53]: # 시작인덱스 생략시 처음부터 지정
a[:4]
executed in 14ms, finished 14:09:53 2021-11-18
Out[53]: array([0, 1, 2, 3])

In [54]: # 끝인덱스 생략시 마지막까지 지정
a[4:]
executed in 14ms, finished 14:09:53 2021-11-18
Out[54]: array([4, 5, 6, 7, 8, 9])

In [55]: # 시작인덱스와 끝인덱스 생략시 자동으로 처음과 마지막을 지정
# 간격은 생략시 자동으로 1임
a[:1]
executed in 13ms, finished 14:09:53 2021-11-18
Out[55]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [56]: a[:3]
executed in 14ms, finished 14:09:53 2021-11-18
Out[56]: array([0, 3, 6, 9])

In [57]: # 간격을 음수를 취하면 뒤에서부터 출력
a[-3]
executed in 15ms, finished 14:09:53 2021-11-18
Out[57]: array([9, 6, 3, 0])

In [58]: # 시작인덱스가 음수이면 뒤에서부터 출력
# a[0, 1, 2, 3]
# a[-4, -3, -2, -1]
a[-4:]
executed in 14ms, finished 14:09:53 2021-11-18
Out[58]: array([6, 7, 8, 9])

In [59]: # 뒤에서 4번째 값부터 2간격으로
a[-4::2]
executed in 14ms, finished 14:09:53 2021-11-18
Out[59]: array([6, 8])

In [60]: a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
a
executed in 14ms, finished 14:09:53 2021-11-18
Out[60]: array([[1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 10, 11, 12]])

In [61]: np.array(range(1,13))
executed in 14ms, finished 14:09:53 2021-11-18
Out[61]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])

In [62]: # reshape을 사용해서 데이터 생성 가능
a = np.array(range(1,13)).reshape(3,4)
a
executed in 14ms, finished 14:09:53 2021-11-18
Out[62]: array([[1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 10, 11, 12]])

In [63]: # 빅터의 경우와 달리 행과 열 각각의 인덱싱 시행
# 1번째 행과 전체 열 출력
a[1,:]
executed in 14ms, finished 14:09:53 2021-11-18
Out[63]: array([5, 6, 7, 8])

In [64]: # 1번째 이상 2번째 미만 행과 전체 열 출력
a[1:2,:]
executed in 14ms, finished 14:09:53 2021-11-18
Out[64]: array([[5, 6, 7, 8]])
```

```
In [65]: # 1번째 이상 3번째 미만 행과 현재 열 출력  
a[1:3,:]  
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[65]: array([[ 5,  6,  7,  8],  
                 [ 9, 10, 11, 12]])
```

```
In [66]: # 전체 행과 1번째 이상 3번째 미만 열 출력  
a[:,1:3]  
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[66]: array([[ 2,  3],  
                 [ 6,  7],  
                 [10, 11]])
```

- 정수 배열 인덱싱(Int Array Indexing): 배열의 특정 위치에 있는 값으로 구성할 수 있음
- 부울 배열 인덱싱(Boolean Array indexing): 배열의 특정 값만 선택할 수 있으며, 일부 조건이 참인 경우의 값을 선택하는데 사용

```
In [67]: a = np.array([[1,2], [3, 4], [5, 6]])  
a  
executed in 8ms, finished 14:09:53 2021-11-18
```

```
Out[67]: array([[1, 2],  
                 [3, 4],  
                 [5, 6]])
```

```
In [68]: # 인덱스를 직접 입력해서 값 출력 가능  
# (0,0), (1,1), (2,0) 값 출력  
np.array([a[0, 0], a[1, 1], a[2, 0]])  
executed in 12ms, finished 14:09:53 2021-11-18
```

```
Out[68]: array([1, 4, 5])
```

```
In [69]: # (0,1,2)번째 행에서 (0,1,0)번째 열의 값 출력  
a[[0, 1, 2], [0, 1, 0]]  
executed in 13ms, finished 14:09:53 2021-11-18
```

```
Out[69]: array([1, 4, 5])
```

```
In [70]: a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])  
a  
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[70]: array([[ 1,  2,  3],  
                 [ 4,  5,  6],  
                 [ 7,  8,  9],  
                 [10, 11, 12]])
```

```
In [71]: b = np.array([0, 2, 0, 1])  
b  
executed in 15ms, finished 14:09:53 2021-11-18
```

```
Out[71]: array([0, 2, 0, 1])
```

```
In [72]: # 0~3까지의 행에서, b의 값 위치 열값 추출  
a[np.arange(4), b]  
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[72]: array([ 1,  6,  7, 11])
```

```
In [73]: # 0~3까지의 행에서, b의 값 위치 열값에 모두 10을 더해 출력  
a[np.arange(4), b] + 10  
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[73]: array([11, 16, 17, 21])
```

```
In [74]: a = np.array([[1,2], [3, 4], [5, 6]])  
a  
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[74]: array([[1, 2],  
                 [3, 4],  
                 [5, 6]])
```

```
In [75]: # 각 값들이 2보다 큰지 논리값 반환  
a>2  
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[75]: array([[False, False],  
                 [ True,  True],  
                 [ True,  True]])
```

```
In [76]: # 각 값들이 2보다 큰지 논리값에서 True인 위치의 행렬값 출력  
a[a>2]  
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[76]: array([3, 4, 5, 6])
```

```
In [77]: a = np.arange(10)  
a  
executed in 14ms, finished 14:09:53 2021-11-18
```

```
Out[77]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [78]: # 5~7번째 값을 100으로 치환
a[5:8] = 100
a
executed in 14ms, finished 14:09:53 2021-11-18
Out[78]: array([ 0,  1,  2,  3, 100, 100, 100,   8,   9])

In [79]: b = list(range(10))
b
executed in 14ms, finished 14:09:53 2021-11-18
Out[79]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [80]: # list에서는 동일하게 작동되지 않음
# 바꾸려는 값의 갯수만큼 치환할 값을 할당해야 가능
# b[5:8] = 100
# b
executed in 14ms, finished 14:09:53 2021-11-18
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-2-74d77b409a34> in <module>
      1 b = list(range(10))
      2 b[5:8] = 100
      3 b
  ----> 4
TypeError: can only assign an iterable
```

```
In [81]: names = np.array(['철수', '영희', '말자', '숙자'])
names
executed in 14ms, finished 14:09:54 2021-11-18
Out[81]: array(['철수', '영희', '말자', '숙자'], dtype='|U2')

In [82]: # 각 값이 철수인지 논리값 반환
names == '철수'
executed in 14ms, finished 14:09:54 2021-11-18
Out[82]: array([ True, False, False, False])

In [83]: # 각 값이 철수인지 논리값 기준 True인 값 반환
names[names == '철수']
executed in 14ms, finished 14:09:54 2021-11-18
Out[83]: array(['철수'], dtype='|U2')

In [84]: data = np.random.randn(4,4)
data
executed in 14ms, finished 14:09:54 2021-11-18
Out[84]: array([[-0.97671583,  0.13167163,  1.61856758,  1.61449301],
       [-0.27387963,  0.51219035,  0.50038249,  0.02253376],
       [-0.69342012,  0.92202835,  0.78231882,  0.59115226],
       [-0.09042872, -0.21549662, -0.18362988, -0.61027774]])

In [85]: # 데이터가 0보다 작은지 논리값 기준 True인 값을 0으로 치환
data[data<0] = 0
data
executed in 14ms, finished 14:09:54 2021-11-18
Out[85]: array([[0.          , 0.13167163, 1.61856758, 1.61449301],
       [0.          , 0.51219035, 0.50038249, 0.02253376],
       [0.          , 0.92202835, 0.78231882, 0.59115226],
       [0.          , 0.          , 0.          , 0.          ]])

In [86]: names == '철수'
executed in 14ms, finished 14:09:54 2021-11-18
Out[86]: array([ True, False, False, False])

In [87]: # names가 철수인지 논리값 기준 True인 값의 data값을 반환
data[names == '철수']
executed in 14ms, finished 14:09:54 2021-11-18
Out[87]: array([[0.          , 0.13167163, 1.61856758, 1.61449301]])

In [88]: # 테스트 array 생성
pet = np.array(['개', '고양이', '고양이', '햄스터', '개', '햄스터'])
num = np.array([1, 2, 3, 4, 5, 6])
indexing_test = np.arange(36).reshape(6, 6) # 샘플데이터
display(pet, num, indexing_test)
executed in 14ms, finished 14:09:54 2021-11-18
array(['개', '고양이', '고양이', '햄스터', '개', '햄스터'], dtype='|U3')
array([1, 2, 3, 4, 5, 6])
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],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

```
[30, 31, 32, 33, 34, 35]])
```

```
In [89]: # num이 3보다 클지 논리값 출력  
num > 3
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[89]: array([False, False, False, True, True, True])
```

```
In [90]: indexing_test[num > 3]
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[90]: array([[18, 19, 20, 21, 22, 23],  
[24, 25, 26, 27, 28, 29],  
[30, 31, 32, 33, 34, 35]])
```

```
In [91]: # pet가 개인지 논리값 출력  
pet == '개'
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[91]: array([ True, False, False, False, True, False])
```

```
In [92]: indexing_test[pet == '개']
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[92]: array([[ 0,  1,  2,  3,  4,  5],  
[24, 25, 26, 27, 28, 29]])
```

```
In [93]: (pet == '개') | (pet == '햄스터')
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[93]: array([ True, False, False, True, True, True])
```

```
In [94]: # 꽃 array 기준으로 햄스터와 개 데이터를 indexing_test에서 인덱싱해보기  
indexing_test[(pet == '개') | (pet == '햄스터'), 3:]
```

```
executed in 13ms, finished 14:09:54 2021-11-18
```

```
Out[94]: array([[ 3,  4,  5],  
[21, 22, 23],  
[27, 28, 29],  
[33, 34, 35]])
```

```
In [95]: # 불리언(True/False) 값을 숫자로 판단할 시 True=1, False=0으로 계산  
(pet == '개').sum()
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[95]: 2
```

```
In [96]: # 하나라도 참이면 참  
(pet == '개').any()
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[96]: True
```

```
In [97]: # 전체데이터가 참이여야 참  
(pet == '개').all()
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[97]: False
```

▼ 1.4 Numpy Function(처리)

▼ 1.4.1 Array 기본 연산

- 기본 수학 함수는 배열에서 값마다 계산되며 numpy 내장 함수 모두 사용 가능
- 값연산 이외에 벡터와 다차원 연산이 가능하기 때문에 반복문(for, while) 사용없이 직관적 연산 가능
- (고급) 크기가 다른 배열간의 연산도 브로드캐스팅(Broadcasting) 방식으로 연산 가능

1) 단일 배열에 사용하는 함수

함수	설명
abs, fabs	각 원소의 절대값을 구한다. 복소수가 아닌 경우에는 fabs로 빠르게 연산 가능
sqrt	제곱근을 계산 arr ** 0.5와 동일
square	제곱을 계산 arr ** 2와 동일
Exp	각 원소에 지수 ex를 계산
Log, log10, log2, logp	각각 자연로그, 로그10, 로그2, 로그(1+x)
sign	각 원소의 부호를 계산
ceil	각 원소의 소수자리를 올림
floor	각 원소의 소수자리를 반올림. dtype 유지
rint	원소의 몫과 나머지를 각각 배열로 반환
modf	각 원소의 소수자리를 각각 배열로 반환
isnan	각 원소가 숫자인지 아닌지 NaN으로 나타내는 불리언 배열
isfinite, isnan	배열의 각 원소가 유한한지 무한한지 나타내는 불리언 배열
cos, cosh, sin, sinh, tan, tanh	일반 삼각함수와 쌍곡삼각 함수
logical_not	각 원소의 논리 부정(not) 값 계산

2) 서로 다른 배열에 사용하는 함수

함수	설명
add	두 배열에서 같은 위치의 원소끼리 덧셈
subtract	첫번째 배열 원소 - 두번째 배열 원소
multiply	배열의 원소끼리 곱셈
divide	첫번째 배열의 원소에서 두번째 배열의 원소를 나누셈
power	첫번째 배열의 원소에 두번째 배열의 원소만큼 제곱
maximum, fmax	두 원소 중 큰 값을 반환. fmax는 NaN 무시
minimum, fmin	두 원소 중 작은 값 반환. fmin은 NaN 무시
mod	첫번째 배열의 원소에 두번째 배열의 원소를 나눈 나머지
greater, greater_equal, less, less_equal, equal, not_equal	두 원소 간의 >, >=, <, <=, ==, != 비교연산 결과를 불리언 배열로 반환
logical_and, logical_or, logical_xor	각각 두 원소 간의 논리연산 결과를 반환

```
In [98]: a2d = np.array([[1., 2.], [3., 4.]], dtype=np.float64)
b2d = np.array([[5., 6.], [7., 8.]], dtype=np.float64)
display(a2d, b2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

array([[1., 2.],
 [3., 4.]])

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

```
In [99]: # 요소별 합산
a2d + b2d
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[99]: array([[ 6.,  8.],
                 [10., 12.]])
```

```
In [100]: # 요소별 합산
np.add(a2d, b2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[100]: array([[ 6.,  8.],
                  [10., 12.]])
```

```
In [101]: # 요소별 빼기
a2d - b2d
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[101]: array([[-4., -4.],
                  [-4., -4.]])
```

```
In [102]: # 요소별 빼기
np.subtract(a2d, b2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[102]: array([[-4., -4.],
                  [-4., -4.]])
```

```
In [103]: # 요소별 곱셈
a2d * b2d
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[103]: array([[ 5., 12.],
                  [21., 32.]])
```

```
In [104]: # 요소별 곱셈
np.multiply(a2d, b2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[104]: array([[ 5., 12.],
                  [21., 32.]])
```

```
In [105]: # 요소별 나누기
a2d / b2d
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[105]: array([[0.2          , 0.33333333],
                  [0.42857143, 0.5        ]])
```

```
In [106]: # 요소별 나누기
np.divide(a2d, b2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[106]: array([[0.2          , 0.33333333],
                  [0.42857143, 0.5        ]])
```

```
In [107]: # 요소별 square root
np.sqrt(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[107]: array([[1.          , 1.41421356],
                  [1.73205081, 2.          ]])
```

```
In [108]: a2d = np.array([[[-1, 2, 3], [-4, 5, 6], [7.1, 8.4, -9.7]]])
a2d
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[108]: array([[-1.,  2.,  3.],
                 [-4.,  5.,  6.],
                 [ 7.1,  8.4, -9.7]])
```

```
In [109]: # 절대값 반환
np.abs(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[109]: array([[1.,  2.,  3.],
                 [4.,  5.,  6.],
                 [7.1,  8.4,  9.7]])
```

```
In [110]: # square root
np.sqrt(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
<ipython-input-110-3c6e0ef3af9f>:2: RuntimeWarning: invalid value encountered in sqrt
  np.sqrt(a2d)
```

```
Out[110]: array([[      nan,  1.41421356,  1.73205081],
                 [      nan,  2.23606798,  2.44948974],
                 [2.66458252,  2.89827535,         nan]])
```

```
In [111]: # is same as
a2d**0.5
executed in 14ms, finished 14:09:54 2021-11-18
<ipython-input-111-af6341ed5ab5>:2: RuntimeWarning: invalid value encountered in sqrt
  a2d**0.5
```

```
Out[111]: array([[      nan,  1.41421356,  1.73205081],
                 [      nan,  2.23606798,  2.44948974],
                 [2.66458252,  2.89827535,         nan]])
```

```
In [112]: # 제곱 반환
np.square(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[112]: array([[ 1.,  4.,  9.],
                 [16.,  25.,  36.],
                 [50.41, 70.56, 94.09]])
```

```
In [113]: # exp
np.exp(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[113]: array([[3.67879441e-01,  7.38905610e+00,  2.00855369e+01],
                 [1.83156389e-02,  1.48413159e+02,  4.03428793e+02],
                 [1.21196707e+03,  4.44706675e+03,  6.12834951e-05]])
```

```
In [114]: # log
# log + exp는 원래대로 반환
np.log(np.exp(a2d))
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[114]: array([[-1.,  2.,  3.],
                 [-4.,  5.,  6.],
                 [ 7.1,  8.4, -9.7]])
```

```
In [115]: # 부호 반환
np.sign(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[115]: array([[-1.,  1.,  1.],
                 [-1.,  1.,  1.],
                 [ 1.,  1., -1.]])
```

```
In [116]: # 옮김값
np.ceil(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[116]: array([[-1.,  2.,  3.],
                 [-4.,  5.,  6.],
                 [ 8.,  9., -9.]])
```

```
In [117]: # 없이 nan인지 여부
np.isnan(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[117]: array([[False, False, False],
                 [False, False, False],
                 [False, False, False]])
```

```
In [118]: # 없이 infinite인지 여부
np.isinf(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[118]: array([[False, False, False],
                 [False, False, False],
                 [False, False, False]])
```

```
In [119]: # 값이 유한값인지 여부
np.isfinite(a2d)
```

```

executed in 14ms, finished 14:09:54 2021-11-18
Out[119]: array([[ True,  True,  True],
   [ True,  True,  True],
   [ True,  True,  True]])

In [120]: # cosine
np.cos(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
Out[120]: array([[ 0.54030231, -0.41614684, -0.9899925 ],
   [-0.65364362,  0.28366219,  0.96017029],
   [ 0.68454667, -0.51928865, -0.96236488]])

In [121]: # tangent
np.tan(a2d)
executed in 14ms, finished 14:09:54 2021-11-18
Out[121]: array([-1.55740772, -2.18503986, -0.14254654],
   [-1.15782128, -3.38051501, -0.29100619],
   [ 1.06489313, -1.64571073, -0.28238835]])

In [122]: b2d = np.array([[1,2,3],[4,5,6],[7,8,9]])
display(a2d, b2d)
executed in 14ms, finished 14:09:54 2021-11-18
array([[-1.,  2.,  3.],
   [-4.,  5.,  6.],
   [ 7.,  8., -9.]])
array([[1, 2, 3],
   [4, 5, 6],
   [7, 8, 9]])

In [123]: # 2개의 행렬 더하기
np.add(a2d, b2d)
executed in 14ms, finished 14:09:54 2021-11-18
Out[123]: array([[ 0.,  4.,  6.],
   [ 0., 10., 12.],
   [14.1, 16.4, -0.7]])

In [124]: # 2개의 행렬 곱하기
np.multiply(a2d, b2d)
executed in 14ms, finished 14:09:54 2021-11-18
Out[124]: array([[-1.,  4.,  9.],
   [-16., 25., 36.],
   [ 49.7, 67.2, -87.3]])

In [125]: # 2개의 행렬 제곱
np.power(a2d, b2d)
executed in 14ms, finished 14:09:54 2021-11-18
Out[125]: array([[-1.00000000e+00,  4.00000000e+00,  2.70000000e+01],
   [ 2.56000000e+02,  3.12500000e+03,  4.66560000e+04],
   [ 9.09512016e+05,  2.47875891e+07, -7.60231059e+08]])

In [126]: # 2개의 행렬 요소별 비교
np.greater(b2d, a2d)
executed in 14ms, finished 14:09:54 2021-11-18
Out[126]: array([[ True, False, False],
   [ True, False, False],
   [False, False,  True]])

▼ 1.4.2 Array 통계 연산

In [127]: a2d = np.array([[1,2,3],[-4,5,6],[7,8,-9]])
a2d
executed in 14ms, finished 14:09:54 2021-11-18
Out[127]: array([-1,  2,  3],
   [-4,  5,  6],
   [ 7,  8, -9])

In [128]: # 요소별 더하기
a2d + 2
executed in 14ms, finished 14:09:54 2021-11-18
Out[128]: array([[ 1,  4,  5],
   [-2,  7,  8],
   [ 9, 10, -7]])

In [129]: # 요소별 곱하기
a2d * 2
executed in 14ms, finished 14:09:54 2021-11-18
Out[129]: array([[ -2,   4,   6],
   [ -8,  10,  12],
   [ 14,  16, -18]])

In [130]: # 값의 형태 확인
a2d.dtype
executed in 14ms, finished 14:09:54 2021-11-18

```

```
Out[130]: dtype('int32')
```

```
In [131]: # 행렬의 크기 확인
```

```
a2d.shape
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[131]: (3, 3)
```

```
In [132]: # 행렬(데이터) 차원 확인
```

```
# 데이터가 존재하는 축방향이 늘어나면 차원수도 늘어남
```

```
a2d.ndim
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[132]: 2
```

```
In [133]: # 행렬 차원 확인
```

```
# 함수사용하여 확인 가능
```

```
np.ndim(a2d)
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[133]: 2
```

```
In [134]: # 전치행렬: 대각선 기준 회전
```

```
a2d.T
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[134]: array([[-1, -4,  7],
```

```
                 [ 2,  5,  8],
```

```
                 [ 3,  6, -9]])
```

```
In [135]: # 전치행렬: 대각선 기준 회전
```

```
# 함수사용 변환가능
```

```
np.transpose(a2d)
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[135]: array([[-1, -4,  7],
```

```
                 [ 2,  5,  8],
```

```
                 [ 3,  6, -9]])
```

```
In [136]: # 세로로 더하기
```

```
np.sum(a2d, axis=0)
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[136]: array([ 2, 15,  0])
```

```
In [137]: # 가로로 더하기
```

```
np.sum(a2d, axis=1)
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[137]: array([4, 7, 6])
```

```
In [138]: # 전체 더하기
```

```
np.sum(a2d)
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[138]: 17
```

```
In [139]: a2d
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[139]: array([[-1,  2,  3],
```

```
                 [-4,  5,  6],
```

```
                 [ 7,  8, -9]])
```

```
In [140]: # 누적합
```

```
np.cumsum(a2d)
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[140]: array([-1,  1,  4,  0,  5, 11, 18, 26, 17], dtype=int32)
```

```
In [141]: # 곱하기
```

```
np.prod(a2d)
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[141]: -362880
```

```
In [142]: # 세로로 곱하기
```

```
np.prod(a2d, axis=0)
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[142]: array([-28, -80, -162])
```

```
In [143]: # 가로로 곱하기
```

```
np.prod(a2d, axis=1)
```

```
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[143]: array([-6, -120, -504])
```

```
In [144]: # 누적 곱하기
```

```
np.cumprod(a2d)
```

```
executed in 13ms, finished 14:09:54 2021-11-18
```

```
Out[144]: array([-1, -2, -6, 24, 120, 720, 5040, 40320, -362880], dtype=int32)
```

```
In [145]: # 세로로 누적 곱하기
np.cumprod(a2d, axis=0)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[145]: array([[ -1,    2,    3],
   [  4,   10,   18],
   [ 28,   80, -162]], dtype=int32)
```

```
In [146]: # 가로로 누적 곱하기
np.cumprod(a2d, axis=1)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[146]: array([[ -1,   -2,   -6],
   [ -4,  -20, -120],
   [  7,   56, -504]], dtype=int32)
```

```
In [147]: # 세로로 평균
np.mean(a2d, axis=0)
executed in 14ms, finished 14:09:54 2021-11-18
```

```
Out[147]: array([0.66666667, 5.,         , 0.         ])
```

```
In [148]: # 전체 평균
np.mean(a2d)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[148]: 1.8888888888888888
```

```
In [149]: # 세로로 표준편차
np.std(a2d, axis=0)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[149]: array([4.64279609, 2.44948974, 6.4807407 ])
```

```
In [150]: # 전체 표준편차
np.std(a2d)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[150]: 5.30082686305625
```

```
In [151]: # 세로로 분산
np.var(a2d, axis=0)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[151]: array([21.55555556, 6.,         , 42.         ])
```

```
In [152]: # 세로로 최소값
# 실제로 많이 사용
np.min(a2d, axis=0)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[152]: array([-4,  2, -9])
```

```
In [153]: # 세로로 최대값
# 실제로 많이 사용
np.max(a2d, axis=0)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[153]: array([7, 8, 6])
```

```
In [154]: # 전체의 최소값과 최대값
np.min(a2d), np.max(a2d)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[154]: (-9, 8)
```

```
In [155]: # 최소값 인덱스
# argmax보다는 뭘 쓰이지만 자주 보임
# 세로로 가장 작은 값의 위치(인덱스)
np.argmin(a2d, axis=0)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[155]: array([1, 0, 2], dtype=int64)
```

```
In [156]: # 자주 사용합니다.
# 최대값 존재하고 있는 인덱스 넘버를 리턴
# 출력값이 인덱스
# 세로로 가장 큰 값의 위치(인덱스)
np.argmax(a2d, axis=0)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[156]: array([2, 2, 1], dtype=int64)
```

```
In [157]: # 세로로 누적합
np.cumsum(a2d, axis=0)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[157]: array([[-1,   2,    3],
   [-5,   7,    9],
   [ 2,  15,   0]], dtype=int32)
```

```
In [158]: # 세로로 누적곱
np.cumprod(a2d, axis=0)
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[158]: array([[ -1,   2,    3],
   [  4,   10,   18],
   [ 28,   80, -162]], dtype=int32)
```

```
[  4,   10,   18],  
[ 28,   80, -162]], dtype=int32)  
  
In [159]: a2d = np.array([[-1,2,3],[-4,5,6],[7,8,-9]])  
b2d = np.array([[1,2,3],[4,5,6],[7,8,9]])  
display(a2d, b2d)  
executed in 14ms, finished 14:09:55 2021-11-18  
array([[-1,  2,  3],  
       [-4,  5,  6],  
       [ 7,  8, -9]])  
  
array([[1,  2,  3],  
       [4,  5,  6],  
       [7,  8,  9]])
```

```
In [160]: # 세로로 붙이기  
c2d = np.concatenate((a2d, b2d), axis=0)  
c2d  
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[160]: array([[-1,  2,  3],  
                  [-4,  5,  6],  
                  [ 7,  8, -9],  
                  [ 1,  2,  3],  
                  [ 4,  5,  6],  
                  [ 7,  8,  9]])
```

```
In [161]: # 가로로 붙이기  
c2d = np.concatenate((a2d, b2d), axis=1)  
c2d  
executed in 15ms, finished 14:09:55 2021-11-18
```

```
Out[161]: array([[-1,  2,  3,  1,  2,  3],  
                  [-4,  5,  6,  4,  5,  6],  
                  [ 7,  8, -9,  7,  8,  9]])
```

```
In [162]: # 크기/형태 변경  
c2d.reshape((9,2))  
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[162]: array([[-1,  2],  
                  [ 3,  1],  
                  [ 2,  3],  
                  [-4,  5],  
                  [ 6,  4],  
                  [ 5,  6],  
                  [ 7,  8],  
                  [-9,  7],  
                  [ 8,  9]])
```

```
In [163]: # 크기/형태 변경  
# 함수 사용해서 변경 가능  
# 어떤 조건에서 reshape 가능한가? 데이터 내부에 존재하는 속성 갯수가 같아야 함.  
np.reshape(c2d, (9,2))  
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[163]: array([[-1,  2],  
                  [ 3,  1],  
                  [ 2,  3],  
                  [-4,  5],  
                  [ 6,  4],  
                  [ 5,  6],  
                  [ 7,  8],  
                  [-9,  7],  
                  [ 8,  9]])
```

```
In [164]: a2d = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]])  
a2d  
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[164]: array([[ 1,  2,  3,  4],  
                  [ 5,  6,  7,  8],  
                  [ 9, 10, 11, 12],  
                  [13, 14, 15, 16]])
```

```
In [165]: # 세로로 배열 분리하기  
np.split(a2d, 2, axis=0)  
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[165]: [array([[1, 2, 3, 4],  
                  [5, 6, 7, 8]]),  
           array([[9, 10, 11, 12],  
                  [13, 14, 15, 16]])]
```

```
In [166]: # 가로로 배열 분리하기  
np.split(a2d, 2, axis=1)  
executed in 14ms, finished 14:09:55 2021-11-18
```

```
Out[166]: [array([[ 1,  2],  
                  [ 5,  6],  
                  [ 9, 10],  
                  [13, 14]]),  
           array([[ 3,  4],  
                  [ 7,  8],  
                  [11, 12],  
                  [15, 16]])]
```

1.5 Questions

• 데이터 정보

- 파일명: 산업분류별_월별전력사용량.txt
- 파일위치: 'r'Data\Energy\산업분류별_월별전력사용량.txt'
- 포함정보:

- 년월
- 가구수
- 사용량
- 전기요금
- 평균판매단가

```
In [167]: # 데이터 불러오기 실행
import numpy as np
raw = np.loadtxt('r'Data\Energy\산업분류별_월별전력사용량.txt', delimiter='\t')
raw
executed in 43ms, finished 14:09:55 2021-11-18
```

```
Out[167]: array([[2.02001000e+05, 1.30000000e+01, 1.11140000e+04, 1.70977500e+06,
       1.54000000e+02],
      [2.02001000e+05, 5.05200000e+03, 1.66106290e+07, 2.29918341e+09,
       1.38000000e+02],
      [2.02001000e+05, 5.57000000e+03, 6.51106010e+07, 8.23555475e+09,
       1.27000000e+02],
      ...
      [2.02012000e+05, 9.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00],
      [2.02012000e+05, 4.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00],
      [2.02012000e+05, 1.30000000e+01, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00]])
```

```
In [168]: # 데이터의 크기 확인하여 행과 열의 갯수 출력
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [169]: # 결측값의 갯수를 출력
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [170]: # 1) 종복되지 않은 년월 값들 출력
# 2) 1)의 출력값 갯수 출력
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [171]: # 전체 기간의 가구수와 사용량 평균 출력
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [172]: # 사용량 최대값과 최소값 출력
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [173]: # 1) 사용량이 음수인 행 출력
# 2) 사용량이 음수인 행을 모두 양수로 변환 출력
# 3) 사용량이 음수인 행을 필터/삭제하여 새로운 raw_filter 이름으로 저장
# 4) raw_filter의 데이터 크기 확인하여 행과 열의 갯수 출력
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [174]: # raw_filter의 전기요금 최대값과 최소값 출력
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [175]: # raw_filter에서 전기요금으로 1조가 넘는 비용을 지불한 년월 출력
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [176]: # 1) raw_filter에서 전기요금으로 1조가 넘는 비용을 지불한 년월의 가구수 출력
# 2) 1)에서 구한 가구수들 중 최대값과 최대값의 인덱스(순번) 출력
# 3) 2)에서 구한 최대값에 해당되는 년월 출력
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [177]: # raw_filter 데이터에서,
# 각 년월별, 아래 출력결과의 형식과 동일하게 년월 + 전기요금 최대값 + 전기요금 최소값 출력
# 년월은 정수로 출력하고 나머지는 실수로 출력
# 출력결과: '년월: 202001, 전기요금최대값: 100000, 전기요금최소값: 0'
#           '년월: 202002, 전기요금최대값: 110000, 전기요금최소값: 100'
#
#
executed in 14ms, finished 14:09:55 2021-11-18
```

```
In [ ]:
```

