5. Numpy 패키지

패키지(Package)

☐ 패키지(Package)란

- 다른 프로그램 제작에 사용하기 위해 미리 만들어진 프로그램의 집합을 라이브러리(library)라고 하며, 파이썬에서는 라이브러리라는 용어 대신 **패키지(package)**라는 말을 주로 사용
- 파이썬에서 모듈은 하나의 .py 파일을 가리키며, 패키지는 이러한 모듈들을 모은 컬렉션을 가리킴
- 도트(.)를 사용하여 파이썬 모듈을 계층적(디렉터리 구조)으로 관리할 수 있게 해줌
- 예를 들어 모듈 이름이 **A.B**인 경우에 A는 패키지 이름이 되고 B는 A 패키지의 B모듈

☐ 설치된 패키지 확인 및 설치

- 파이썬에서는 pip라고 하는 패키지 관리자를 기본으로 제공
  - `pip list` 명령으로 설치된 패키지 확인 가능

```
In [1]: pip list
```

```
Package                       Version
----------------------------- --------------------
aiohttp                       3.8.1
aiosignal                     1.2.0
alabaster                     0.7.12
anaconda-client               1.9.0
anaconda-navigator            2.1.4
anaconda-project              0.10.2
anyio                         3.5.0
appdirs                       1.4.4
argon2-cffi                   21.3.0
argon2-cffi-bindings          21.2.0
arrow                         1.2.2
astroid                       2.6.6
astropy                       5.0.4
asttokens                     2.0.5
async-timeout                 4.0.1
atomicwrites                  1.4.0
attrs                         21.4.0
```

- pip 패키지 관리자를 이용하여 패키지를 설치하려면 콘솔 창에서 다음과 같이 입력

```
pip install 패키지이름
```

  - 설치 한 후 **사용하기 위해서는 패키지를 import 해야 함**
    - import는 현재 디렉터리에 있는 파일이나 파이썬 라이브러리가 저장된 디렉터리에 있는 모듈만 불러올 수 있음

```
import 모듈이름
```

  - **Numpy 패키지 설치 및 import**

```
pip install numpy #numpy 패키지 설치 import numpy #numpy 불러오기
import numpy as np #np라는 이름으로 numpy 불러오기
```

☐ 데이터 분석을 위한 주요 패키지

- **Numpy (넘파이)**
  - Numpy는 C언어로 구현된 파이썬 라이브러리로써, 고성능의 수치계산을 위해 제작
  - Numerical Python의 줄임말이기도 한 Numpy는 벡터 및 행렬 연산에 있어 매우 편리한 기능을 제공
  - 데이터분석을 할 때 사용되는 라이브러리인 pandas와 matplotlib의 기반으로 사용
  - numpy에서는 기본적으로 array라는 단위로 데이터를 관리하며 이에 대해 연산을 수행(행렬 개념)
    - 심볼릭 연산 라이브러리
    - 홈페이지: http://www.sympy.org/
    - 개발: 2006, Ondřej Čertík

- **Pandas (판다스)**
  - 파이썬에서 사용하는 데이터분석 라이브러리로, 행과 열로 이루어진 데이터 객체를 만들어 다룰 수 있게 되며 보다 안정적으로 대용량의 데이터들을 처리하는데 매우 편리한 도구
  - 테이블 형태의 데이터를 다루는 데이터프레임(DataFrame) 자료형을 제공
  - 자료의 탐색이나 정리에 아주 유용하여 데이터 분석에 빠질 수 없는 필수 패키지
  - 2008년도에 Wes McKinney에 의해 프로젝트가 시작되었다. 원래는 R 언어에서 제공하는 데이터프레임 자료형을 파이썬에서 제공할 수 있도록 하는 목적이었으나 더 다양한 기능이 추가됨
    - 데이터 분석 라이브러리. R의 data.frame 자료구조 구현
    - 홈페이지: http://pandas.pydata.org/
    - 개발: 2008, Wes McKinney (AQR Capital Management)
- **Matplotlib (맷플롯리브)**
  - 파이썬에서 각종 그래프나 챠트 등을 그리는 시각화 기능을 제공
  - Tkinter, wxPython, Qt, GTK+ 등의 다양한 그래픽 엔진을 사용할 수 있음
  - 또한, MATLAB의 그래프 기능을 거의 동일하게 사용할 수 있는 pylab이라는 서브패키지를 제공하므로 MATLAB에 익숙한 사람들은 바로 Matplotlib을 사용할 수 있음
    - 시각화 라이브러리, MATLAB 플롯 기능 구현
    - 홈페이지: http://matplotlib.org/
    - 개발: 2002, John D. Hunter
- 기타 패키지
  - SciPy(사이파이) : 고급 수학 함수, 수치적 미적분, 미분 방정식 계산, 최적화, 신호 처리 등에 사용하는 다양한 과학 기술 계산 기능을 제공
  - SymPy(심파이) : 숫자를 더하거나 빼는 수치 연산이 아니라 인수 분해, 미분, 적분 등 심볼릭 연산 기능을 제공
  - Seaborn(시본) : Matplotlib 패키지에서 지원하지 않는 고급 통계 차트를 그리는 통계용 시각화 기능을 제공

Numpy 패키지

☐ Numpy 패키지 설치 및 import

```
In [1]: # numpy패키지 설치

In [3]: pip install numpy  # pip을 사용하여 numpy 설치

        Requirement already satisfied: numpy in c:₩users₩parkmj₩anaconda3₩lib₩site-packages (1.18.1)
        Note: you may need to restart the kernel to use updated packages.

In [7]: import numpy as np #np라는 이름으로 numpy를 사용
```

- 많은 숫자 데이터를 하나의 변수에 넣고 관리 할 때 **리스트는 속도가 느리고 메모리를 많이 차지하는 단점**이 있음
    - **배열(array)을 사용하면 적은 메모리로 많은 데이터를 빠르게 처리**할 수 있음
    - 배열은 리스트와 비슷하지만 다음과 같은 점에서 다름
        1. **모든 원소가 같은 자료형**이어야 한다.(리스트는 각가의 원소가 다른 자료형이 될수 있음)
        2. 원소의 갯수를 바꿀 수 없다.
    - 파이썬은 **자체적으로 배열 자료형을 제공하지 않음**. 따라서 배열을 구현한 다른 패키지를 임포트해야 함. 파이썬에서 **배열을 사용하기 위한 표준 패키지는 넘파이(NumPy)임**
    - 배열은 위와 같은 제약사항이 있는 대신 원소에 대한 접근과 반복문 실행이 빨라짐

☐ 배열(Array) 만들기

- **numpy shape** : numpy에서는 해당 array의 크기를 알 수 있음. shape 을 확인함으로써 몇 개의 데이터가 있는지, 몇 차원으로 존재하는지 등을 확인
- numpy에서 사용되는 **자료형**(자료형 뒤에 붙는 숫자는 몇 비트 크기인지를 의미)

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

- 1차원 배열 만들기
  - 넘파이의 `array` 함수에 리스트를 넣으면 `ndarray` 클래스 객체 즉, 배열로 변환시켜
  - **arr1.shape**의 결과는 (5,) 으로써, 1차원의 데이터이며 총 5라는 크기를 갖고 있음
    - **numpy 자료형(dtype함수)** : arr1이나 arr2는 int32라는 자료형을 갖는 것에 반해 arr3는 float32라는 자료형을 가짐
    - 이는 arr3내부 데이터를 살펴보면 3.5라는 실수형 데이터를 갖기 때문

In [8]:
```
#Array 정의하기 및 사용
```

In [9]:
```
data1 = [1,2,3,4,5]
data1
```

Out[9]:
```
[1, 2, 3, 4, 5]
```

In [10]:
```
data2 = [1,2,3,3.5,4]
data2
```

Out[10]:
```
[1, 2, 3, 3.5, 4]
```

In [11]:
```
# numpy를 이용하여 array 정의하기
# 1. 위에서 만든 python list를 이용
arr1 = np.array(data1)
arr1
```

Out[11]:
```
array([1, 2, 3, 4, 5])
```

In [12]:
```
# array의 형태(크기)를 확인할 수 있다.
arr1.shape
```

Out[12]:
```
(5,)
```

In [13]:
```
# 2. 바로 python list를 넣어 줌으로써 만들기
arr2 = np.array([1,2,3,4,5])
arr2
```

Out[13]:
```
array([1, 2, 3, 4, 5])
```

In [14]:
```
arr2.shape
```

Out[14]:
```
(5,)
```

In [15]:
```
# array의 자료형을 확인할 수 있다.
arr2.dtype
```

Out[15]:
```
dtype('int32')
```

In [16]:
```
arr3 = np.array(data2)
arr3
```

Out[16]:
```
array([1. , 2. , 3. , 3.5, 4. ])
```

In [17]:
```
arr3.shape
```

Out[17]:
```
(5,)
```

In [18]:
```
arr3.dtype
```

Out[18]:
```
dtype('float64')
```

- 2차원 배열 만들기
  - **arr4.shape**의 결과는 (4,3) 으로써, 2차원의 데이터이며 4 * 3 크기를 갖고 있는 array

```
In [19]: arr4 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
         arr4
Out[19]: array([[ 1,  2,  3],
                [ 4,  5,  6],
                [ 7,  8,  9],
                [10, 11, 12]])

In [20]: arr4.shape
Out[20]: (4, 3)
```

- 3차원 배열 만들기
  - **arr5.shape**의 결과는 (2,4,3) 으로써, 3차원의 데이터이며 2*4 * 3 크기를 갖고 있는 array

```
In [24]: arr5=np.array([[[1,2,3],[4,5,6],[7,8,9],[10,11,12]],
                        [[13,14,15],[16,17,18],[19,20,21],[22,23,24]]])

In [25]: type(arr5)
Out[25]: numpy.ndarray

In [26]: arr5.shape
Out[26]: (2, 4, 3)
```

- 기타 배열 만드는 함수
  - **np.zeros()** : 인자로 받는 크기만큼, **모든요소가 0인** array를 만듬
  - **np.ones()** : 인자로 받는 크기만큼, **모든요소가 1인** array를 만듬
  - **np.arange()** : 인자로 받는 값 만큼 **1씩 증가하는 1차원** array를 만듬
  - **np.full()** : (N,N) shape을 특정값으로 채움
  - **np.eye()** : (N,N) shape의 단위 행렬을 생성
  - **np.empty()** : 초기화되지 않은 값으로 zeros나 ones와 마찬가지로 배열을 생성

```
In [21]: # np.zeros() 함수는 인자로 받는 크기만큼, 모든요소가 0인 array를 만듬

In [22]: np.zeros(10)
Out[22]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

In [23]: np.zeros((3,5))
Out[23]: array([[0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.]])

In [24]: # np.ones() 함수는 인자로 받는 크기만큼, 모든요소가 1인 array를 만듬

In [25]: np.ones(9)
Out[25]: array([1., 1., 1., 1., 1., 1., 1., 1., 1.])

In [26]: np.ones((2,10))
Out[26]: array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])

In [27]: # np.arange() 함수는 인자로 받는 값 만큼 1씩 증가하는 1차원 array를 만듬
         # 이때 하나의 인자만 입력하면 0 ~ 입력한 인자, 값 만큼의 크기를 가짐

In [28]: np.arange(10)
Out[28]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [29]: np.arange(3,10)
Out[29]: array([3, 4, 5, 6, 7, 8, 9])
```

In [148]:
```
# (N, N) shape을 특정값으로 채움, (2,2)를 7로 채움
np.full((2,2),7)
```

Out[148]:
```
array([[7, 7],
       [7, 7]])
```

In [149]:
```
#(N, N) shape의 단위 행렬(Unit Matrix)을 생성
np.eye(4)
```

Out[149]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [150]:
```
#초기화되지 않은 값으로 zeros나 ones와 마찬가지로 배열을 생성
#주의해야 할 것은 메모리도 초기화되지 않기 때문에 예상하지 못한 쓰레기 값이 들어가 있음
np.empty((4,2))
```

Out[150]:
```
array([[0.94679596, 1.22371697],
       [0.10052805, 0.26680367],
       [2.8825159 , 0.31909186],
       [0.75582999, 0.48346625]])
```

- np.zeros_like(), np.ones_like(), np.empty_like() 함수는 이미 있는 array와 동일한 모양과 데이터 형태를 유지한 상태에서 각 각 '0', '1', '빈 배열'을 반환

```
In [151]: # arange로 1부터 10 미만의 범위에서 1씩 증가하는 배열 생성
          # 배열의 shape을 (3, 3)으로 지정
          a = np.arange(1, 10).reshape(3, 3)

In [152]: a

Out[152]: array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])

In [154]: np.zeros_like(a)

Out[154]: array([[0, 0, 0],
                 [0, 0, 0],
                 [0, 0, 0]])

In [155]: np.ones_like(a)

Out[155]: array([[1, 1, 1],
                 [1, 1, 1],
                 [1, 1, 1]])

In [156]: np.empty_like(a)

Out[156]: array([[   77668480,         596,    48801232],
                 [        596,           0,           0],
                 [-2147483648,         596,  2131941552]])
```

☐ Array 연산

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

```
In [30]: #Array 연산

In [31]: arr1 = np.array([[1,2,3],[4,5,6]])
         arr1

Out[31]: array([[1, 2, 3],
                [4, 5, 6]])

In [32]: arr1.shape

Out[32]: (2, 3)

In [33]: arr2 = np.array([[10,11,12],[13,14,15]])
         arr2

Out[33]: array([[10, 11, 12],
                [13, 14, 15]])

In [34]: arr2.shape

Out[34]: (2, 3)

In [36]: #Array 덧셈
         arr1 + arr2

Out[36]: array([[11, 13, 15],
                [17, 19, 21]])

In [37]: #Array 뺄셈
         arr1 - arr2

Out[37]: array([[-9, -9, -9],
                [-9, -9, -9]])

In [38]: #Array 곱셈
         arr1 * arr2

Out[38]: array([[10, 22, 36],
                [52, 70, 90]])

In [39]: #Array 나눗셈
         arr1 / arr2

Out[39]: array([[0.1       , 0.18181818, 0.25      ],
                [0.30769231, 0.35714286, 0.4       ]])
```

- **array의 브로드 캐스트** : 위에서는 array가 같은 크기를 가져야 서로 연산이 가능하다고 했지만, numpy에서는 브로드캐스트라는 기능을 제공
  - 브로드캐스트 : **서로 크기가 다른 array가 연산**이 가능하게 함

```
In [40]: #arrar의 브로드 캐스트

In [52]: arr1
Out[52]: array([[1, 2, 3],
                [4, 5, 6]])

In [53]: arr1.shape
Out[53]: (2, 3)

In [54]: arr3 = np.array([10,11,12])
         arr3
Out[54]: array([10, 11, 12])

In [55]: arr3.shape
Out[55]: (3,)

In [56]: arr1 + arr3 #서로 다른 크기 연산가능. [10,11,12],[10,11,12]로 확장되어 계산
Out[56]: array([[11, 13, 15],
                [14, 16, 18]])

In [57]: arr1 * 10
Out[57]: array([[10, 20, 30],
                [40, 50, 60]])

In [58]: # 요소에 대해 제곱처리
         arr1 ** 2
Out[58]: array([[ 1,  4,  9],
                [16, 25, 36]], dtype=int32)
```

☐ Array의 인덱싱

- numpy에서 사용되는 인덱싱은 기본적으로 **python 인덱싱과 동일**
- **python에서와 같이 1번째로 시작**하는 것이 아니라 0번째로 시작

In [59]:
```
#Array 인덱싱
```

In [60]:
```
arr1 = np.arange(10)
arr1
```

Out [60]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [61]:
```
# 0번째 요소
arr1[0]
```

Out [61]:
```
0
```

In [62]:
```
# 3번째 요소
arr1[3]
```

Out [62]:
```
3
```

In [63]:
```
# 3번째 요소부터 8번째 요소
arr1[3:9]
```

Out [63]:
```
array([3, 4, 5, 6, 7, 8])
```

In [64]:
```
arr1[:]
```

Out [64]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [65]:
```
arr2 = np.array([[1,2,3,4],
                 [5,6,7,8],
                 [9,10,11,12]])
arr2
```

Out [65]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [66]:
```
# 2차원의 array에서 인덱싱을 하기 위해선 2개의 인자를 입력해야 합니다.
arr2[0,0]
```

Out [66]:
```
1
```

In [67]:
```
# 2행의 모든 요소 꺼내기
arr2[2,:]
```

Out [67]:
```
array([ 9, 10, 11, 12])
```

In [68]:
```
# 2행의 3번째 요소 꺼내기
arr2[2,3]
```

Out [68]:
```
12
```

In [69]:
```
# 모든 열의 3번째 요소 꺼내기
arr2[:,3]
```

Out [69]:
```
array([ 4,  8, 12])
```

- **Array boolean 인덱싱(마스크)**
  - 위에서 이용한 다차원의 인덱싱을 응용하여 **boolean 인덱싱**을 할 수 있음
  - 해당 기능은 주로 마스크라고 이야기하는데, boolean 인덱싱을 통해 만들어 낸 array를 통해 **원하는 행 또는 열의 값만 뽑아낼 수 있음(TRUE 인 부분만 출력됨)**
  - 즉, 마스크처럼 가리고 싶은 부분은 가리고, 원하는 요소만 꺼낼 수 있음

```
In [76]: # 아래에서 사용되는 np.random.randn() 함수는 기대값이 0이고, 표준편차가 1인 가우시안 정규 분포를 따르는 난수를 발생시키는 함수
         # 이 외에도 0~1의 난수를 발생시키는 np.random.rand() 함수도 존재한다.
         data = np.random.randn(8,4) #8행 4열로 난수 채움
         data

Out[76]: array([[-1.14732163e+00, -2.18328472e-01, -7.87242484e-01,
                  1.62500130e+00],
                [ 2.88785562e-01, -5.33439904e-01,  9.00057858e-01,
                  6.90885726e-01],
                [ 9.46795962e-01,  1.22371697e+00,  1.00528049e-01,
                 -2.66803667e-01],
                [-7.53136519e-01,  2.60148313e-01, -1.13804398e+00,
                 -1.39448170e-03],
                [ 6.19639434e-01,  1.39801411e+00, -9.27462443e-01,
                  1.01073757e+00],
                [ 8.12272632e-01,  9.35877230e-01,  1.52829414e-01,
                 -6.72128098e-01],
                [ 2.88251590e+00,  3.19091861e-01,  7.55829988e-01,
                  4.83466253e-01],
                [-6.12833154e-02, -2.20989893e-01, -8.57481126e-01,
                 -9.55433638e-01]])

In [77]: data.shape

Out[77]: (8, 4)
```

  - data array 자체적으로도 마스크를 만들고, 이를 응용하여 인덱싱이 가능
    - data array에서 **0번째 열의 값이 0보다 작은 행**을 구하기
    - data array에서 **0번째 열의 값이 0보다 작은 행의 2,3번째 열값에 0을 대입**

```
In [82]: # 먼저 마스크를 만든다.
         # data array에서 0번째 열이 0보다 작은 요소의 boolean 값은 다음과 같다.
         data[:,0] < 0

Out[82]: array([ True, False, False,  True, False, False, False,  True])
```

```
In [83]: # 위에서 만든 마스크를 이용하여 0번째 열의 값이 0보다 작은 행을 구한다.
         data[data[:,0]<0,:]

Out[83]: array([[-1.14732163e+00, -2.18328472e-01, -7.87242484e-01,
                 -1.62500130e+00],
                [-7.53136519e-01,  2.60148313e-01, -1.13804398e+00,
                 -1.39448170e-03],
                [-6.12833154e-02, -2.20989893e-01, -8.57481126e-01,
                 -9.55433638e-01]])
```

- names의 각 요소가 data의 각 행과 연결된다고 가정
  - names가 **'Beomwoo'인 행의 data만 보고 싶을 때** 다음과 같이 마스크를 사용
  - 요소가 Beomwoo인 것은 0번째, 1번째, 5번째, 7번째 이므로 data에서 0,1,5,7행의 모든 요소를 꺼내와야 함
  - 이를 위해 **요소가 Beomwoo인 것에 대한 boolean 값을 가지는 mask를 만들었고 마스크를 인덱싱에 응용하여** data의 0,1,5,7행을 추출

```
In [84]: # 0번째 열의 값이 0보다 작은 행의 2,3번째 열 값
         data[data[:,0]<0,2:4]

Out[84]: array([[-7.87242484e-01, -1.62500130e+00],
                [-1.13804398e+00, -1.39448170e-03],
                [-8.57481126e-01, -9.55433638e-01]])
```

```
In [70]: #Array Boolean 인덱싱

In [71]: names = np.array(['Beomwoo','Beomwoo','Kim','Joan','Lee','Beomwoo','Park','Beomwoo'])
         names

Out[71]: array(['Beomwoo', 'Beomwoo', 'Kim', 'Joan', 'Lee', 'Beomwoo', 'Park',
                'Beomwoo'], dtype='<U7')

In [72]: names.shape

Out[72]: (8,)

In [74]: names[0,]

Out[74]: 'Beomwoo'

In [75]: names[4,]

Out[75]: 'Lee'
```

```
In [78]: # 요소가 Beomwoo인 항목에 대한 mask 생성
         names_Beomwoo_mask = (names == 'Beomwoo')
         names_Beomwoo_mask

Out[78]: array([ True,  True, False, False, False,  True, False,  True])
```

```
In [79]: data[names_Beomwoo_mask,:]

Out[79]: array([[-1.14732163, -0.21832847, -0.78724248,  1.6250013 ],
                [ 0.28878556, -0.5334399 ,  0.90005786,  0.69088573],
                [ 0.81227263,  0.93587723,  0.15282941, -0.6721281 ],
                [-0.06128332, -0.22098989, -0.85748113, -0.95543364]])
```

```
In [80]: # 요소가 Kim인 행의 데이터만 꺼내기
         data[names == 'Kim',:]

Out[80]: array([[ 0.94679596,  1.22371697,  0.10052805, -0.26680367]])
```

```
In [81]: # 논리 연산을 응용하여, 요소가 Kim 또는 Park인 행의 데이터만 꺼내기
         data[(names == 'Kim') | (names == 'Park'),:]

Out[81]: array([[ 0.94679596,  1.22371697,  0.10052805, -0.26680367],
                [ 2.8825159 ,  0.31909186,  0.75582999,  0.48346625]])
```

- 다차원의 배열 인덱싱

```
In [27]: # 배열 인덱싱은 다차원 배열의 각 차원에 대해 가능
         a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
         a

Out[27]: array([[ 1,  2,  3,  4],
                [ 5,  6,  7,  8],
                [ 9, 10, 11, 12]])

In [28]: a[:, [True, False, False, True]]

Out[28]: array([[ 1,  4],
                [ 5,  8],
                [ 9, 12]])

In [29]: a[[2, 0, 1], :]

Out[29]: array([[ 9, 10, 11, 12],
                [ 1,  2,  3,  4],
                [ 5,  6,  7,  8]])
```

☐ Numpy 함수

- 하나의 array에 적용되는 함수

함수	설명	함수	설명
np.abs	각 성분 절대값 계산	np.sign	각 성분의 부호 계산하기 (+인 경우 1, -인 경우 -1, 0인 경우 0)
np.sqrt	각 성분 제곱근 계산	np.ceil	각 성분의 소수 첫 번째 자리에서 올림한 값을 계산
np.square	각 성분 제곱 계산	np.floor	각 성분의 소수 첫 번째 자리에서 내림한 값을 계산
np.exp	e의 지수로 삼은 값을 계산	np.isnan	각 성분이 NaN인 경우 True를, 아닌 경우 False를 반환하기
np.log	각 성분 자연로그를 씌운 값을 계산	np.isinf	각 성분이 무한대인 경우 True를, 아닌 경우 False를 반환
np.log10	각 성분 상용로그를 씌운 값을 계산	np.cos	각 성분에 대해 삼각함수 값을 계산하기 (cos, cosh, sin, sinh, tan, tanh)
np.log2	각 성분 밑이 2인로그를 씌운 값을 계산		

In [87]:
```
#하나의 arrary에 적용되는 함스
```

In [88]:
```
arr1 = np.random.randn(5,3)
arr1
```

Out[88]:
```
array([[ 0.24556261,  1.43475166, -0.8663763 ],
       [ 0.31975127,  0.2188742 ,  1.34225246],
       [-0.43051703,  0.00477715,  0.87518836],
       [ 0.45073559, -0.46197827, -0.15606305],
       [ 1.01944772, -0.04231902,  1.96402074]])
```

In [89]:
```
# 각 성분의 절대값 계산하기
np.abs(arr1)
```

Out[89]:
```
array([[0.24556261, 1.43475166, 0.8663763 ],
       [0.31975127, 0.2188742 , 1.34225246],
       [0.43051703, 0.00477715, 0.87518836],
       [0.45073559, 0.46197827, 0.15606305],
       [1.01944772, 0.04231902, 1.96402074]])
```

In [90]:
```
# 각 성분의 제곱근 계산하기 ( == array ** 0.5)
np.sqrt(arr1)
```

```
C:₩Users₩ParkMJ₩anaconda3₩lib₩site-packages₩ipykernel_launcher.py:2: RuntimeWarning: invalid v
```

Out[90]:
```
array([[0.49554274, 1.1978112 ,        nan],
       [0.56546553, 0.46783993, 1.1585562 ],
       [       nan, 0.06911693, 0.93551502],
       [0.67136845,        nan,        nan],
       [1.00967704,        nan, 1.40143524]])
```

In [91]:
```
# 각 성분의 제곱 계산하기
np.square(arr1)
```

Out[91]:
```
array([[6.03009941e-02, 2.05851234e+00, 7.50607889e-01],
       [1.02240873e-01, 4.79059172e-02, 1.80164167e+00],
       [1.85344916e-01, 2.28211663e-05, 7.65954660e-01],
       [2.03162576e-01, 2.13423919e-01, 2.43556767e-02],
       [1.03927365e+00, 1.79089973e-03, 3.85737745e+00]])
```

In [92]:
```
# 각 성분을 무리수 e의 지수로 삼은 값을 계산하기
np.exp(arr1)
```

Out[92]:
```
array([[1.27834031, 4.19860222, 0.42047246],
       [1.37678527, 1.24467469, 3.82765545],
       [0.65017285, 1.00478858, 2.39932718],
       [1.56946625, 0.63003603, 0.85550525],
       [2.77166361, 0.95856393, 7.12792905]])
```

In [93]:
```
# 각 성분을 자연로그, 상용로그, 밑이 2인 로그를 씌운 값을 계산하기
np.log(arr1)
```

```
C:₩Users₩ParkMJ₩anaconda3₩lib₩site-packages₩ipykernel_launcher.py:2: RuntimeWarning: invalid
```

Out [93]:
```
array([[-1.40420334,  0.36099178,         nan],
       [-1.14021188, -1.51925812,  0.29434914],
       [        nan, -5.34391105, -0.13331615],
       [-0.79687438,         nan,         nan],
       [ 0.01926103,         nan,  0.67499377]])
```

In [94]:
```
np.log10(arr1)
```

```
C:₩Users₩ParkMJ₩anaconda3₩lib₩site-packages₩ipykernel_launcher.py:1: RuntimeWarning: invalid
  """Entry point for launching an IPython kernel.
```

Out [94]:
```
array([[-0.60983776,  0.15677674,         nan],
       [-0.49518773, -0.65980542,  0.12783421],
       [        nan, -2.32083108, -0.05789847],
       [-0.34607814,         nan,         nan],
       [ 0.00836496,         nan,  0.29314607]])
```

In [95]:
```
np.log2(arr1)
```

```
C:₩Users₩ParkMJ₩anaconda3₩lib₩site-packages₩ipykernel_launcher.py:1: RuntimeWarning: invalid
  """Entry point for launching an IPython kernel.
```

Out [95]:
```
array([[-2.0258372 ,  0.52080105,         nan],
       [-1.64497802, -2.19182616,  0.42465605],
       [        nan, -7.70963397, -0.19233455],
       [-1.14954671,         nan,         nan],
       [ 0.02778779,         nan,  0.97381016]])
```

In [96]:
```
# 각 성분의 부호 계산하기(+인 경우 1, -인 경우 -1, 0인 경우 0)
np.sign(arr1)
```

Out [96]:
```
array([[ 1.,  1., -1.],
       [ 1.,  1.,  1.],
       [-1.,  1.,  1.],
       [ 1., -1., -1.],
       [ 1., -1.,  1.]])
```

In [97]:
```
# 각 성분의 소수 첫 번째 자리에서 올림한 값을 계산하기
np.ceil(arr1)
```

Out [97]:
```
array([[ 1.,  2., -0.],
       [ 1.,  1.,  2.],
       [-0.,  1.,  1.],
       [ 1., -0., -0.],
       [ 2., -0.,  2.]])
```

```
In [99]: # 각 성분의 소수 첫 번째 자리에서 내림한 값을 계산하기
         np.floor(arr1)

Out[99]: array([[ 0.,  1., -1.],
                [ 0.,  0.,  1.],
                [-1.,  0.,  0.],
                [ 0., -1., -1.],
                [ 1., -1.,  1.]])

In [100]: # 각 성분이 NaN인 경우 True를, 아닌 경우 False를 반환하기
          np.isnan(arr1)

Out[100]: array([[False, False, False],
                 [False, False, False],
                 [False, False, False],
                 [False, False, False],
                 [False, False, False]])

In [101]: np.isnan(np.log(arr1))

C:\Users\ParkMJ\anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: invalid
  """Entry point for launching an IPython kernel.

Out[101]: array([[False, False,  True],
                 [False, False, False],
                 [ True, False, False],
                 [False,  True,  True],
                 [False,  True, False]])

In [102]: # 각 성분이 무한대인 경우 True를, 아닌 경우 False를 반환하기
          np.isinf(arr1)

Out[102]: array([[False, False, False],
                 [False, False, False],
                 [False, False, False],
                 [False, False, False],
                 [False, False, False]])

In [103]: # 각 성분에 대해 삼각함수 값을 계산하기(cos, cosh, sin, sinh, tan, tanh)
          np.cos(arr1)

Out[103]: array([[ 0.97000071,  0.13562539,  0.64759201],
                 [ 0.94931363,  0.97614251,  0.22655949],
                 [ 0.90875009,  0.99998859,  0.64085227],
                 [ 0.9001269 ,  0.8951725 ,  0.98784686],
                 [ 0.52383647,  0.99910468, -0.38316872]])
```

- 두 개의 array에 적용되는 함수

함수	설명	함수	설명
np.add(A,B)	두 개의 array에 대해 동일한 위치의 성분끼리 더하기를 계산	np.divide(A,B)	두 개의 array에 대해 동일한 위치의 성분끼리 나누기를 계산
np.subtract(A,B)	두 개의 array에 대해 동일한 위치의 성분끼리 빼기를 계산	np.maximum(A,B)	두 개의 array에 대해 동일한 위치의 성분끼리 비교하여 최대값을 계산
np.multiply(A,B)	두 개의 array에 대해 동일한 위치의 성분끼리 곱하기를 계산	np.minimum(A,B)	두 개의 array에 대해 동일한 위치의 성분끼리 비교하여 최소값을 계산

In [104]:
```
#두 개의 array에 적용되는 함수
```

In [105]:
```
arr1
```

Out[105]:
```
array([[ 0.24556261,  1.43475166, -0.8663763 ],
       [ 0.31975127,  0.2188742 ,  1.34225246],
       [-0.43051703,  0.00477715,  0.87518836],
       [ 0.45073559, -0.46197827, -0.15606305],
       [ 1.01944772, -0.04231902,  1.96402074]])
```

In [106]:
```
arr2 = np.random.randn(5,3)
arr2
```

Out[106]:
```
array([[-1.04693206e+00,  9.33189448e-02,  9.02289412e-01],
       [ 1.23255318e+00,  2.28817230e-01, -3.88844236e-01],
       [ 7.55978597e-01, -1.95894253e+00, -3.43284061e-01],
       [-8.85400371e-01,  9.40806397e-01,  8.92513855e-01],
       [-1.59755687e-03,  1.56123535e+00, -6.06908228e-02]])
```

In [107]:
```
# 두 개의 array에 대해 동일한 위치의 성분끼리 연산 값을 계산하기(add, subtract, multiply, divide)
np.multiply(arr1,arr2)
```

Out[107]:
```
array([[-0.25708737,  0.13388951, -0.78172216],
       [ 0.39411044,  0.05008219, -0.52192713],
       [-0.32546166, -0.00935816, -0.30043821],
       [-0.39908146, -0.43463211, -0.13928844],
       [-0.00162863, -0.06606995, -0.11919803]])
```

In [108]:
```
# 두 개의 array에 대해 동일한 위치의 성분끼리 비교하여 최대값 또는 최소값 계산하기(maximum, minimum)
np.maximum(arr1,arr2)
```

Out[108]:
```
array([[0.24556261, 1.43475166, 0.90228941],
       [1.23255318, 0.22881723, 1.34225246],
       [0.7559786 , 0.00477715, 0.87518836],
       [0.45073559, 0.9408064 , 0.89251385],
       [1.01944772, 1.56123535, 1.96402074]])
```

- 통계 함수
  - 통계 함수를 통해 array의 합이나 평균 등을 구할 때, 추가로 axis라는 인자에 대한 값을 지정하여 열 또는 행의 합 또는 평균 등을 구할 수 있음
  - **axis = 0 : 행, axis = 1 : 열**

함수	설명	함수	설명
np.sum	전체 성분의 합	np.max	전체 성분의 최대값
np.mean	전체 성분의 평균	np.argmin	전체 성분의 최소값이 위치한 인덱스 반환
np.std	전체 성분의 표준편차	np.argmax	전체 성분의 최소값이 위치한 인덱스 반환
np.var	전체 성분의 분산	np.cumsum	맨 처음 성분부터 각 성분까지의 누적합
np.min	전체 성분의 최소값	np.cumprod	맨 처음 성분부터 각 성분까지의 누적곱

```
In [109]: #통계함수

In [110]: arr1

Out[110]: array([[ 0.24556261,  1.43475166, -0.8663763 ],
                 [ 0.31975127,  0.2188742 ,  1.34225246],
                 [-0.43051703,  0.00477715,  0.87518836],
                 [ 0.45073559, -0.46197827, -0.15606305],
                 [ 1.01944772, -0.04231902,  1.96402074]])

In [111]: # 전체 성분의 합을 계산
          np.sum(arr1)

Out[111]: 5.918108089351024

In [112]: # 열 간의 합을 계산
          np.sum(arr1, axis=1)

Out[112]: array([ 0.81393797,  1.88087793,  0.44944847, -0.16730573,  2.94114943])

In [113]: # 행 간의 합을 계산
          np.sum(arr1, axis=0)

Out[113]: array([1.60498016, 1.15410573, 3.1590222 ])

In [114]: # 전체 성분의 평균을 계산
          np.mean(arr1)

Out[114]: 0.3945405392900682

In [115]: # 행 간의 평균을 계산
          np.mean(arr1, axis=0)

Out[115]: array([0.32099603, 0.23082115, 0.63180444])

In [116]: # 전체 성분의 표준편차, 분산, 최소값, 최대값 계산(std, var, min, max)
          np.std(arr1)

Out[116]: 0.7649393310144933
```

In [117]:
```
np.min(arr1, axis=1)
```

- 기타 함수

  - axis = 0 : 행, axis = 1 : 열

함수	설명
np.sort(arr1)	열 축 기준으로 오름차순 정렬
np.sort(arr1,axis=0)[::-1]	행 축 기준으로 내림차순 정렬 (from top to bottom) ※ [::-1] 는 axis = 0 기준으로 2차원 numpy 배열 뒤집기 (mirror 반환하기)
np.sort(arr1,axis=0)	행 축 기준으로 오름차순 정렬 (from top to bottom)
np.sort(arr1,axis=1)	열 축 기준으로 오름차순 정렬 (from left to right)
np.sort(arr1, axis=1)[:,::-1]	열 축 기준으로 내림차순 정렬 (from left to right) ※ [:,::-1] 는 axis = 1 기준으로 2차원 numpy 배열 뒤집기 (mirror 반환하기)

```
In [124]: #기타함수

In [136]: arr1

Out[136]: array([[ 0.24556261,  1.43475166, -0.8663763 ],
                 [ 0.31975127,  0.2188742 ,  1.34225246],
                 [-0.43051703,  0.00477715,  0.87518836],
                 [ 0.45073559, -0.46197827, -0.15606305],
                 [ 1.01944772, -0.04231902,  1.96402074]])

In [137]: # 열 축 기준으로 오름차순 정렬(왼쪽에서 오른쪽으로)
          np.sort(arr1)

Out[137]: array([[-0.8663763 ,  0.24556261,  1.43475166],
                 [ 0.2188742 ,  0.31975127,  1.34225246],
                 [-0.43051703,  0.00477715,  0.87518836],
                 [-0.46197827, -0.15606305,  0.45073559],
                 [-0.04231902,  1.01944772,  1.96402074]])

In [138]: # 행 축 기준으로 오름차순으로 정렬(위에서 아래로)
          np.sort(arr1,axis=0)

Out[138]: array([[-0.43051703, -0.46197827, -0.8663763 ],
                 [ 0.24556261, -0.04231902, -0.15606305],
                 [ 0.31975127,  0.00477715,  0.87518836],
                 [ 0.45073559,  0.2188742 ,  1.34225246],
                 [ 1.01944772,  1.43475166,  1.96402074]])

In [139]: # 행 축 기준으로 내림차순으로 정렬(위에서 아래로)
          np.sort(arr1,axis=0)[::-1]

Out[139]: array([[ 1.01944772,  1.43475166,  1.96402074],
                 [ 0.45073559,  0.2188742 ,  1.34225246],
                 [ 0.31975127,  0.00477715,  0.87518836],
                 [ 0.24556261, -0.04231902, -0.15606305],
                 [-0.43051703, -0.46197827, -0.8663763 ]])
```

In [140]:
```
# 열 축 기준으로 오름차순 정렬(왼쪽에서 오른쪽으로)
np.sort(arr1,axis=1)
```

Out[140]:
```
array([[-0.8663763 ,  0.24556261,  1.43475166],
       [ 0.2188742 ,  0.31975127,  1.34225246],
       [-0.43051703,  0.00477715,  0.87518836],
       [-0.46197827, -0.15606305,  0.45073559],
       [-0.04231902,  1.01944772,  1.96402074]])
```

In [141]:
```
# 열 축 기준으로  내림차순으로 정렬(왼쪽에서 오른쪽으로)
np.sort(arr1,axis=1)[:,::-1]
```

Out[141]:
```
array([[ 1.43475166,  0.24556261, -0.8663763 ],
       [ 1.34225246,  0.31975127,  0.2188742 ],
       [ 0.87518836,  0.00477715, -0.43051703],
       [ 0.45073559, -0.15606305, -0.46197827],
       [ 1.96402074,  1.01944772, -0.04231902]])
```