<11주차. Numpy 함수 정리>

손효지 20175074

1. 생성 함수

-기본 행렬 생성 함수들, 행렬 생성시 자주 사용함

1) np.full()

```
np. full((2,2,),7)  # 0 대신 다른 값을 넣는 함수
array( [[7 7],
       [7 7]])
```

2) np.eye()

```
np.eye(2)          # 단위행렬( 주 대각 성분이 1 이고 나머진 0)생성 함수
array([[ 1,   0],
       [0,   1]])
```

3) np.empty()

```
np.empty((4,2))  # ones 와 zeros 와 비슷하나 초기화하지 않음
    array ([[  0.00000000e+000,   6.91240343e-310]
    [  6.91240500e-310,   5.39088070e-317]
    [  5.39084907e-317,   6.91239798e-310]
    [  3.16202013e-322,   6.91239798e-310]])
```

2. 범용함수(올림 혹은 내림)

- unfuncs라고 불리는 유니버설 함수는 ndarray 안에 있는 데이터 원소 별로 연산을 수행하는 함수

- 일종의 래퍼 함수로 간단하게 다른 함수에 약간의 기능을 추가하여 사용하는 함수

```
import numpy as np
a=np.array([-4.62, -2.19, 0, 1.57, 3.40, 4.06])
array([-4.62, -2.19, 0, 1.57, 3.40, 4.06])
np.around(a)  #0.5를 기준으로 올림 혹은 내림
array([-5, -2, 0, 2, 3, 4])
```

3. 지수함수

- $f(x)=a_0\times a^x$ 인 함수
- NumPy 의 np.exp() 함수는 밑(base)이 자연상수 e 인 지수함수 $y=e^x$ 로 변환해줌

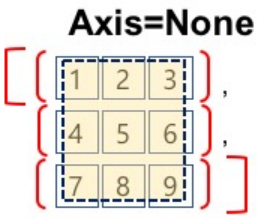
```
import numpy as np
x=np.array([0.00001, 1, 2, 4, 10, 100])
array([  1.00000000e-05,   1.00000000e+00,   2.00000000e+00,
         4.00000000e+00,   1.00000000e+01,   1.00000000e+02])
np.exp(x)
array([  1.00001000e+00,   2.71828183e+00,   7.38905610e+00,
         5.45981500e+01,   2.20264658e+04,   2.68811714e+43])
```

4. 집계 함수

- NumPy 의 모든 집계 함수는 집계 함수는 AXIS 를 기준으로 계산됨
- 집계함수에 AXIS 를 지정하지 않으면 axis=None. axis=None, 0, 1

1) axis=None

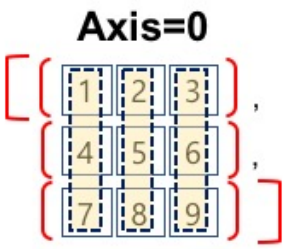
전체 행렬을 하나의 배열로 간주하고 집계 함수의 범위를 전체 행렬로 정의함



2) axis=0

행을 기준으로 각 행의 동일 인덱스의 요소를 그룹으로 함

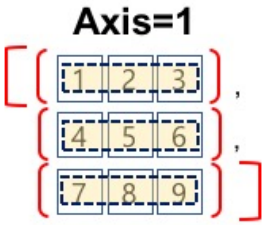
각 그룹을 집계 함수의 범위로 정의



3) axis=1

열을 기준으로 각 열의 요소를 그룹으로 함

각 그룹을 집계 함수의 범위로 정의



5. 특정계산 함수

-1부터 9까지 정수 array를 만들고, np.squrt(arr)메소드를 사용

-np.arrange(n)은 0부터 n-1까지, (1,n)은 1부터 n-1까지 정수 array 생성

-np.squrt(arr) 매소드는 각 array 성분에 제곱근을 입힘

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

In [5]: np.sqrt(arr)
Out[5]: array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
               2.44948974, 2.64575131, 2.82842712, 3.        ])
```

6. 통계함수

- array 전체성분에 대한 각종 통계량을 계산할 수 있는 함수

- np.이 아닌 array.으로 시작하는 것이 특징

- arr.mean()을 이용해서 전체성분의 평균을 구함

```
In [13]: arr
Out[13]: array([[-0.52835747, -1.01455074, -1.01379805,  1.82093666],
                [ 1.34467713,  1.86803527,  1.62364067, -0.84812868],
                [-0.34872064, -1.52630287, -0.02768315, -0.00589635],
                [ 0.94050725, -0.86368574,  0.12584721,  0.15858602],
                [ 0.66943862, -1.13833344, -1.60013155,  0.05314274]])

In [14]: arr.sum()
Out[14]: -0.31077708077800714
```

```
In [15]: arr.mean()
Out[15]: -0.015538854038900356
```

7. 정렬함수

- 데이터상에서 상위-% 등의 데이터를 구하는 방법

- 각 열을 오름, 내림차순으로 정렬(axis=0)하면, 서로 다른 행간에 성분들이 섞여버리게 되기 때문에, 특정 열을 기준으로 행 단위를 오름차순으로 정렬해야 함

ex) 1차원 배열 정렬 # 원래 배열은 그대로, 정렬 결과 복사본 반환

➔ np.sort(x)

배열 자체를 정립

➔ x.sort()

```
In [16]: import numpy as np
         x=np.array([4,2,6,5,1,3,0])
         np.sort(x)
         x.sort()
         x
Out[16]: array([0, 1, 2, 3, 4, 5, 6])
```

8. 논리함수

-참 또는 거짓의 Boolean 값을 변환함

ex) 배열에 nan(Not a Number) 포함 여부 확인 함수

```
In [21]: import numpy as np

         a = np.array([0, 1, 2, np.nan, 4, np.inf, np.NINF, np.PINF])
         print(a)
         [ 0.  1.  2. nan  4. inf -inf  inf]

In [22]: np.isnan(a)
Out[22]: array([False, False, False,  True, False, False, False, False])
```

9. 삼각함수

-시그널 데이터 변환, 벡터 내적 계산에 쓰임

- Python NumPy 의 삼각함수는 radian 을 사용하기 때문에 degree 를 radian 으로 바꿔주기 위해서 degree * np.py/180 을 해줌

```
In [23]: import numpy as np

         np.sin(np.array((0., 30., 45., 60., 90.))*np.pi / 180.)
Out[23]: array([0.        , 0.5       , 0.70710678, 0.8660254 , 1.        ])

In [24]: np.cos(np.array((0., 30., 45., 60., 90.))*np.pi / 180.)
Out[24]: array([1.00000000e+00, 8.66025404e-01, 7.07106781e-01, 5.00000000e-01,
                6.12323400e-17])

In [25]: np.tan(np.array((0., 30., 45., 60., 90.))*np.pi / 180.)
Out[25]: array([0.00000000e+00, 5.77350269e-01, 1.00000000e+00, 1.73205081e+00,
                1.63312394e+16])
```

10. 부호 판별함수

-np.sign(x) # 1(positive), 0(zero), -1(negative) 값 반환

```
In [27]: c=np.array([-2,-1,0,1,2])
         print(c)
         [-2 -1  0  1  2]

In [28]: np.sign(c)
Out[28]: array([-1, -1,  0,  1,  1])
```