NumPy (1)

수치해석용 파이썬 패키지

[NumPy 배열 (array)]

리스트와 차이점

모든 원소가 같은 자료형.

원소의 수 변경 불가능

리스트보다 적은 메모리사용. 빠른 처리

다차원의 배열 자료구조 클래스인 ndarray 클래스를 지원

벡터와 행렬을 사용하는 선형대수 계산에 주로 사용.

```
import numpy as py
```

[1차원 배열 만들기]

NumPy의 array라는 함수에 리스트를 넣으면 배열로 변환

```
ar = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print(  type(ar)  )
```

실행결과 :

```
numpy.ndarray
```

[벡터화 연산] 배열 객체는 배열의 각 원소에 대한 반복 연산을 하나의 명령어로 처리

예1)

```
data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
x = np.array(data)
x = 2 * x
print(x)
```

실행결과 :

[0 2 4 6 8 10 12 14 16 18]

Cf. 일반적인 리스트 객체에 정수를 곱하면 객체의 크기가 정수배 만큼으로 증가

```
data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(2 * data)
```

실행결과 :

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

예2)

```
a = np.array([1, 2, 3])
b = np.array([10, 20, 30])
print( 2 * a + b )
print(a == 2 )
print(b > 10)
print((a == 2) & (b > 10))
```

실행결과 :

```
 [12, 24, 36]
[False  True False]
[False  True  True]
[False  True False]
```

[2차원 배열 만들기]

2차원 배열은 **행렬(matrix)**

가로줄을 **행**(row) 세로줄을 **열**(column)

예) 2 x 3 배열

```
c = np.array([[0, 1, 2], [3, 4, 5]])
print ( len(c) ) # 행의 갯수
print (  c  )
```

실행결과 : 2

```
[[0 1 2]
 [3 4 5]]
```

[3차원 배열 만들기]

2차원 배열은 **행렬(matrix)**

가로줄을 **행**(row) 세로줄을 **열**(column)

예) 2 x 3 x 4 배열

```
d = np.array([[[1, 2, 3, 4],
               [5, 6, 7, 8],
               [9, 10, 11, 12]],
              [[11, 12, 13, 14],
               [15, 16, 17, 18],
```

```
             [19, 20, 21, 22]]])   # 2 x 3 x 4 array
```

[배열의 차원과 크기 알아내기]

dim 속성은 배열의 차원, shape 속성은 배열의 크기를 반환

```
a = np.array([1, 2, 3])
print(a.ndim)
print(a.shape)
```

실행결과 :

```
1
(3,)
```

```
c = np.array([[0, 1, 2], [3, 4, 5]])
print(c.ndim)
print(c.shape)
```

실행결과 :

```
2
(2, 3)
```

```
d = np.array([[[1, 2, 3, 4],
               [5, 6, 7, 8],
               [9, 10, 11, 12]],
              [[11, 12, 13, 14],
               [15, 16, 17, 18],
               [19, 20, 21, 22]]])   # 2 x 3 x 4 array
print(d.ndim)
print(d.shape)
```

실행결과 :

```
3
(2, 3, 4)
```

[배열의 인덱싱]

```
a = np.array([0, 1, 2, 3, 4])
print(a[0],  a[-1] )   ) # 첫번째   ,    마지막번째
c = np.array([[0, 1, 2], [3, 4, 5]])
```

```
print(a[0, 0] )  # 첫번째 행의 첫번째 열
print(a[-1, -1] ) # 마지막 행의 마지막 열
```

[배열 슬라이싱]

```
a = np.array([[0, 1, 2, 3], [4, 5, 6, 7]])

print(a[0, :]  ) # 0 행   모든열
print(a[: , 0]  )  # 모든행  0 열
print(a[0:2, :-1]  ) #0 행~1 행,  처음부터 마지막열까지
```

실행결과 :

```
[0 1 2 3]
[0 4]
[[0 1 2]
 [4 5 6]]
```

```
a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

print( a % 2 )

print(a % 2 == 0 )

print(a[a % 2 == 0])

print(a)

a = a[a % 2 == 0]

print(a)
```

실행 결과 :

```
[0 1 0 1 0 1 0 1 0 1]
[ True  False  True  False  True  False  True  False  True  False]
[0 2 4 6 8]
[0 1 2 3 4 5 6 7 8 9]
[0 2 4 6 8]
```

[전치 연산] 2차원 배열의 전치(transpose) 연산은 행과 열을 바꾸는 것.

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

print(A)

print(A.T)   #전치(transpose) 연산
```

실행결과 :

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


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