

The Numpy logo consists of a red square with a white border. Inside the square, the word "Numpy" is written in white, bold, sans-serif font.

Numpy

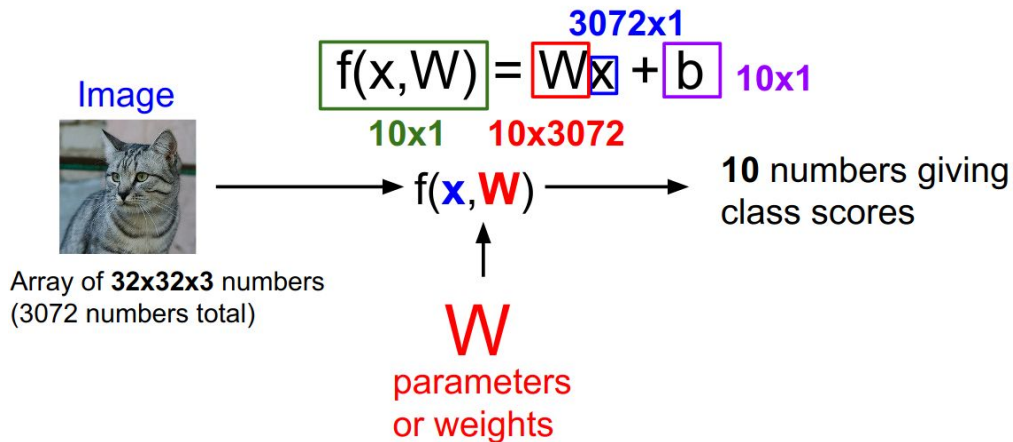
목차

1. Numpy 특징
2. 자료구조 ndarray
3. ndarray 생성
4. handling shape
5. indexing/ slicing
6. 산술 연산, 유니버설 함수, 기본
통계 메서드
7. any, all, where
8. 선형대수

Numpy

- 수치 계산을 위한 라이브러리

Parametric Approach: Linear Classifier



Numpy 핵심 기능

- ndarray (빠르고 메모리를 효율적으로 사용. 브로드 캐스팅)
- 반복문 사용할 필요 없이 전체 데이터 배열에 대해 빠른 연산을 제공하는 표준 수학 함수
- C, C++, 포트란으로 쓰여진 코드를 통합하는 도구
- 선형대수, 푸리에 변환, 난수 발생기 등

자료구조 ndarray

Ndarray

Numpy의 배열 클래스

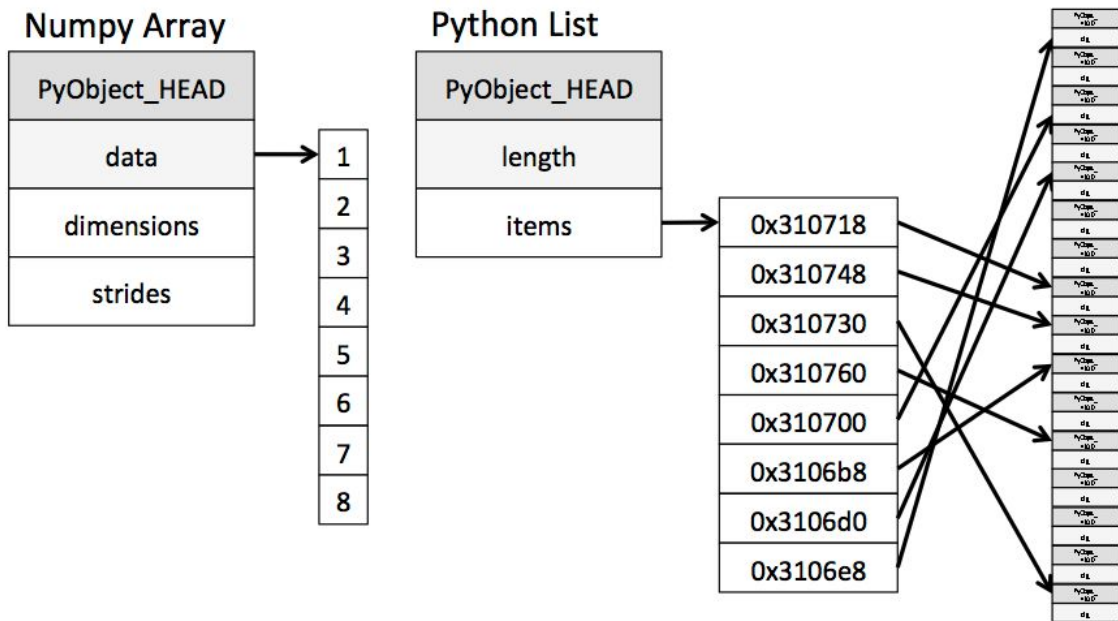
N차원의 배열 객체

같은 종류의 데이터를 담는 포괄적인 다차원 배열.

연속된 메모리 블록에 저장. 대규모 데이터 집합을 담을 수 있는 자료구조.

```
array([[ -10,   -9,   -8,   -7,   -6],  
       [  -5,   -4,   -3,   -2,   -1],  
       [   0,    1,    2,    3,    4],  
       [   5,    6,    7,    8,    9]])
```

ndarray vs list



ndarray attributes

- **dtype**
 - 배열에 담긴 원소의 자료형
 - int8, float16, complex32, bool, object, string_, unicode_ 등 (참고)

```
arr
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
arr.dtype
```

```
dtype('int64')
```

ndarray attributes

- **itemsizes**
 - 배열에 있는 각 원소의 크기(byte)

```
arr
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
arr.itemsize
```

```
8
```


ndarray attributes

- size
 - 배열에 있는 원소의 개수

```
arr
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
arr.size
```

```
8
```

ndarray attributes

- shape
 - 배열의 각 차원의 크기

```
arr
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
arr.shape
```

```
(2, 4)
```

ndarray attributes

- **ndim**
 - 배열의 차원

```
arr
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
arr.ndim
```

```
2
```

ndarray attributes

1 D array



```
array([1, 2, 3, 5])
```

- size: ?
- shape: ?
- ndim: ?

ndarray attributes

1 D array

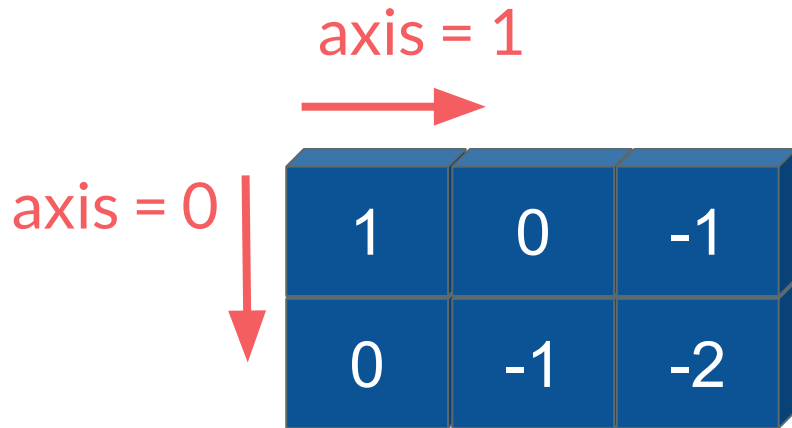


```
array([1, 2, 3, 5])
```

- size: 4
- shape: (4,)
- ndim: 1

ndarray attributes

2 D array

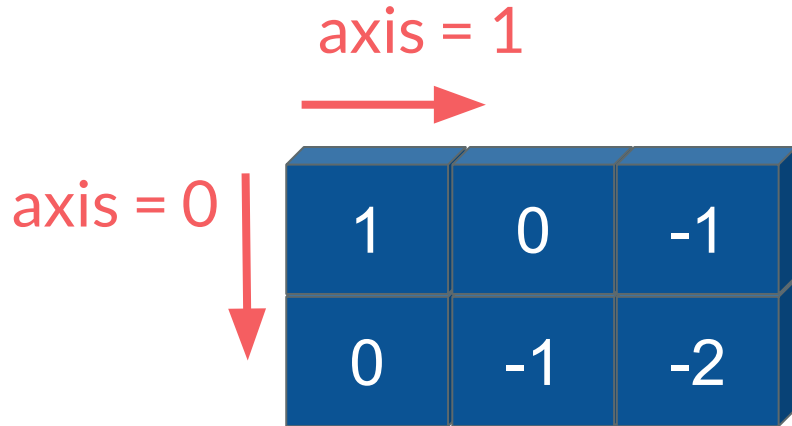


```
array([[ 1,  0, -1],  
       [ 0, -1, -2]])
```

- size: ?
- shape: ?
- ndim: ?

ndarray attributes

2 D array

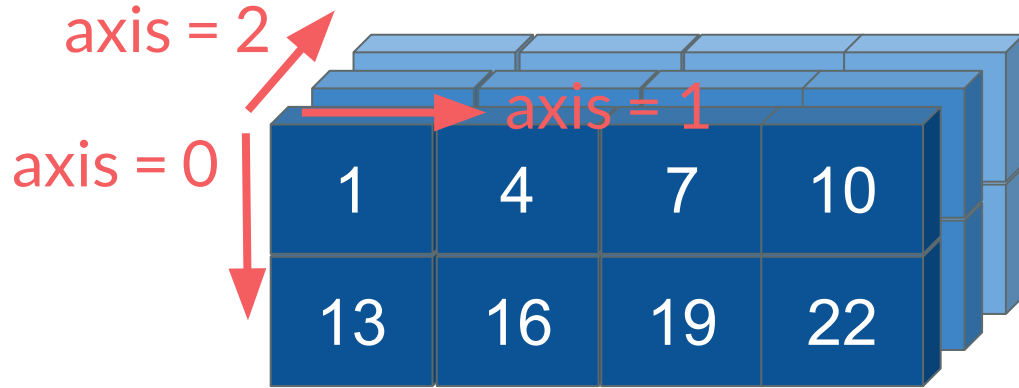


```
array([[ 1,  0, -1],  
       [ 0, -1, -2]])
```

- size: 6
- shape: (2, 3)
- ndim: 2

ndarray attributes

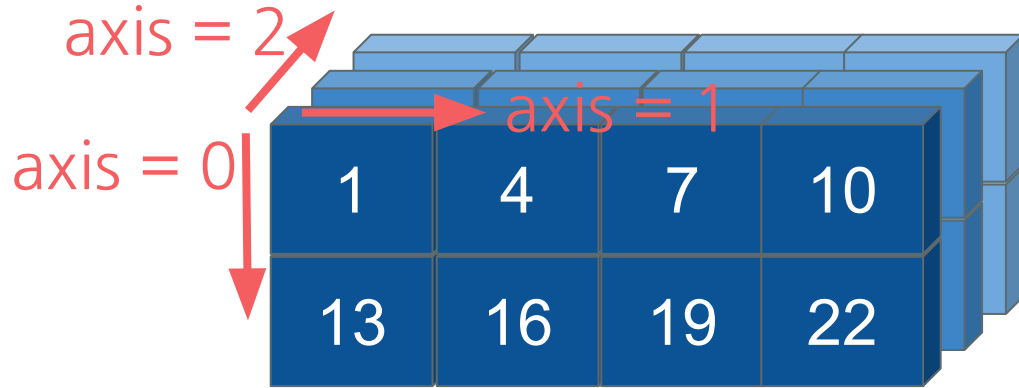
3 D array



```
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]]])
```


ndarray attributes

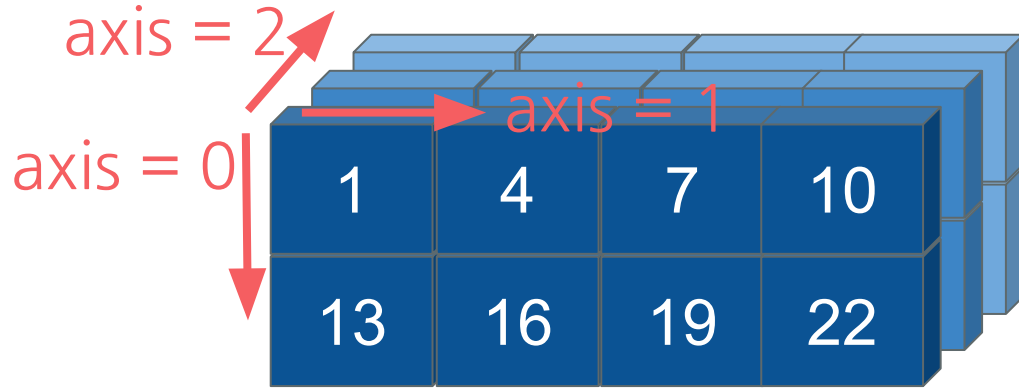
3 D array



- size: ?
- shape: ?
- ndim: ?

ndarray attributes

3 D array



- **size:** 24
- **shape:** (2, 4, 3)
- **ndim:** 3

ndarray attributes

`arr.shape`

(2, 4, 3)

3 D array

```
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]]])
```

```
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]]])
```

```
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]]])
```

ndarray

ndarray 특징

dtype

itemsize

size

shape

ndim

—

ndarray 생성

1. 기존 데이터로 새로운 배열 생성 - array 함수 이용

- `np.array(데이터)`

```
arr = np.array([10, 20, 30])  
arr
```

```
array([10, 20, 30])
```

```
arr = np.array([10, 20, 30], dtype=np.float16)  
arr
```

```
array([10., 20., 30.], dtype=float16)
```

ndarray 생성

2. 배열 생성 함수

```
np.zeros((2, 10), dtype=np.int8)
```

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int8)
```

```
np.full((10), 5)
```

```
array([5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
```

ndarray 생성

2. 배열 생성 함수

```
np.identity(4, dtype=int)
```

```
array([[1, 0, 0, 0],  
       [0, 1, 0, 0],  
       [0, 0, 1, 0],  
       [0, 0, 0, 1]])
```

```
np.arange(10)
```

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

ndarray 생성

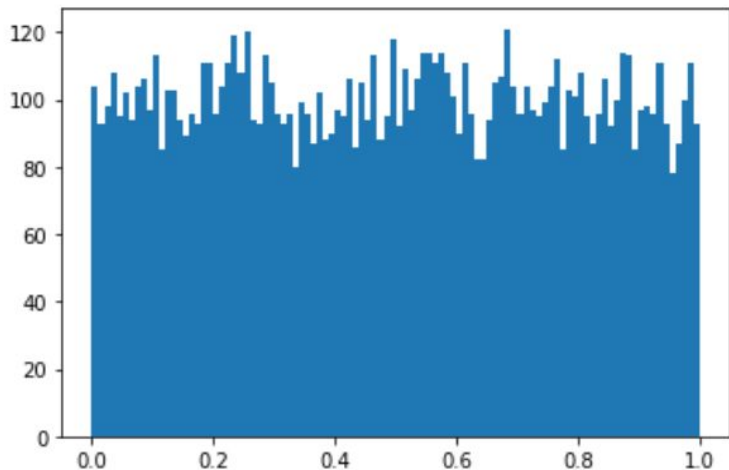
2. 배열 생성 함수

- zeros, ones, full
- identity, eye
- zeros_like, ones_like, full_like
- arange, linspace

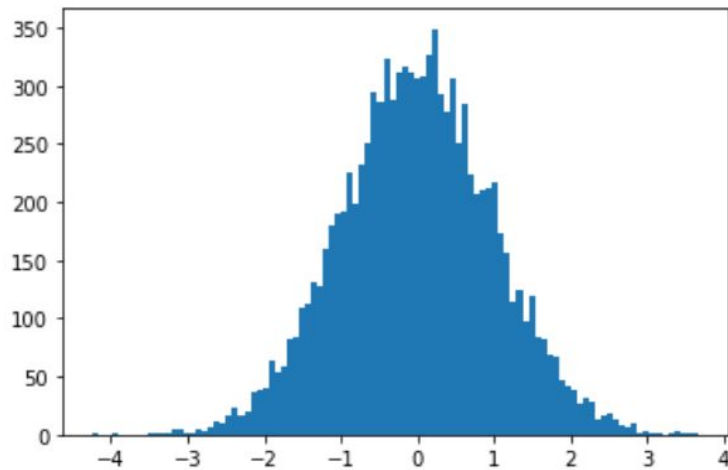
ndarray 생성

3. 난수 생성

```
data = np.random.rand(10000)  
plt.hist(data, bins=100)  
plt.show()
```



```
data = np.random.randn(10000)  
plt.hist(data, bins=100)  
plt.show()
```



ndarray 생성

3. 난수 생성

- random.
 - rand, randn
 - randint
 - binomial, beta, gamma, chisquare, dirichlet

ndarray 생성

1. array 함수
2. 배열 생성 함수
3. 난수 생성

Handling shape

1. flatten

n dim \rightarrow 1 dim

1	2	3
4	5	6
7	8	9



1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

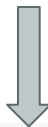
Handling shape

1. flatten

n dim \rightarrow 1 dim

```
arr
```

```
array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 0., 0.]],  
      [[0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 0., 0.]])
```



```
arr.flatten()
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
       0., 0., 0., 0., 0., 0., 0.])
```

Handling shape

2. reshape

```
arr = np.arange(20)  
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,  
       17, 18, 19])
```

```
arr.reshape(2, 10)
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]])
```

Handling shape

2. reshape

```
arr = np.arange(20)  
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,  
       17, 18, 19])
```

```
arr.reshape(3, 7)
```

ValueError

Traceback (most recent call last)

[<ipython-input-98-9781375aa0a8>](#) in <module>()
----> 1 arr.reshape(3, 7)

ValueError: cannot reshape array of size 20 into shape (3,7)

Handling shape

2. reshape

```
arr = np.arange(20)  
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,  
       17, 18, 19])
```

```
arr.reshape(-1, 4)
```

```
array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11],  
       [12, 13, 14, 15],  
       [16, 17, 18, 19]])
```


Handling shape

3. transpose, T

1	2	3
4	5	6
7	8	9

A



1	4	7
2	5	8
3	6	9

A^T

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Handling shape

3. T

```
x = np.arange(6).reshape((-1,3))  
x
```

```
array([[0, 1, 2],  
       [3, 4, 5]])
```

```
x.T
```

```
array([[0, 3],  
       [1, 4],  
       [2, 5]])
```

Handling shape

3. transpose

```
arr = np.arange(10).reshape(2, 5)  
arr
```

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

```
print(arr.transpose().shape)  
arr.transpose()
```

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

handling shape

1. flatten
2. reshape
3. transpose, T

indexing/ slicing

1. indexing

```
arr1d = np.arange(8)  
arr1d
```

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

```
arr1d[1]
```

```
1
```

```
arr2d = np.arange(20).reshape(2, -1)  
arr2d
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]])
```

```
arr2d[1][0]
```

```
10
```

```
arr2d[1, 0]
```

```
10
```

indexing/ slicing

1. indexing

```
arr3d = np.arange(20).reshape(2, 2, -1)  
arr3d
```

```
array([[[ 0,  1,  2,  3,  4],  
        [ 5,  6,  7,  8,  9]],  
       [[10, 11, 12, 13, 14],  
        [15, 16, 17, 18, 19]]])
```

```
arr3d[0, 1, 2]
```

indexing/ slicing

2. slicing

```
arr1d = np.arange(8)  
arr1d
```

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

```
arr1d[:4]
```

```
array([0, 1, 2, 3])
```

```
arr1d[3:6] = 100 # 브로드캐스팅
```

```
arr1d
```

```
array([ 0,  1,  2, 100, 100, 100,  6,  7])
```

indexing/ slicing

2. slicing

```
arr2d = np.arange(20).reshape(2, -1)  
arr2d
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]])
```

```
arr2d[:, :2]
```

```
array([[ 0,  1],  
       [10, 11]])
```


indexing/ slicing

2. slicing

```
arr2d
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14],  
       [15, 16, 17, 18, 19]])
```

```
arr2d[:3][:2]
```

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

```
arr2d[:3, :2]
```

```
array([[ 0,  1],  
       [ 5,  6],  
       [10, 11]])
```

indexing/ slicing

3. boolean indexing

```
arr = np.array([0, 1, 2, 3, 4], int)
arr[[True, False, True, False, True]]

array([0, 2, 4])
```

indexing/ slicing

4. fancy indexing:

Numpy에서 정수 배열을 인덱스로 사용하여 인덱싱 하는 것.

```
arr = np.arange(10)*10+5  
arr
```

```
array([ 5, 15, 25, 35, 45, 55, 65, 75, 85, 95])
```

```
arr[[0, 2, 4, 6]]
```

```
array([ 5, 25, 45, 65])
```

indexing slicing

1. indexing
2. slicing
3. boolean indexing
4. fancy indexing

산술연산

Numpy 배열의 중요한 특징:

for 문을 작성하지 않고 데이터를 일괄 처리할 수 있음(벡터화)

```
arr = np.array([[1, 2], [3, 4]])  
arr
```

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

```
arr + arr
```

```
array([[2, 4],  
       [6, 8]])
```

```
arr*3
```

```
array([[ 3,  6],  
       [ 9, 12]])
```

```
arr/3
```

```
array([[0.33333333, 0.66666667],  
       [1.          , 1.33333333]])
```

산술연산

브로드 캐스팅 규칙

1. 차원이 작은 쪽의 왼쪽이 1로 채워짐
2. 차원은 같은데 shape이 다르고, 그게 1이면 브로드 캐스팅 일어남
3. 차원은 같은데 shape이 다르고, 그 값이 다르면 오류

산술연산

arr

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

arr2

```
array([100, 200, 300])
```

arr + arr2

```
array([[101, 202, 303],  
       [104, 205, 306]])
```

arr

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

arr3

```
array([[100],  
       [200]])
```

arr + arr3

```
array([[101, 102, 103],  
       [204, 205, 206]])
```

유니버설 함수

ndarray 안에 있는

데이터 원소별로(element-wise)

연산을 수행하는 함수

```
arr = np.arange(0, 5, 0.5).reshape(2, -1)  
arr
```

```
array([[0. , 0.5, 1. , 1.5, 2. ],  
       [2.5, 3. , 3.5, 4. , 4.5]])
```

```
np.ceil(arr)
```

```
array([[0., 1., 1., 2., 2.],  
       [3., 3., 4., 4., 5.]])
```

```
np.floor(arr)
```

```
array([[0., 0., 1., 1., 2.],  
       [2., 3., 3., 4., 4.]])
```


유니버설 함수

Function	설명
abs, fabs	각 원소의 절대값. 복소수가 아닌 경우 fabs를 쓰면 빠른 연산이 가능
sqrt	각 원소의 제곱근 계산($arr^{**0.5}$)
square	각 원소의 제곱을 계산(arr^{**2})
exp	각 원소에 지수 e^x 계산
log, log10, log2, log1p	자연로그, 밑10인 로그, 밑2인 로그, $\log(1+x)$
sign	각 원소의 부호(양수 1, 영 0, 음수 -1)
ceil	각 원소의 값보다 같거나 큰 정수 중 가장 작은 값
floor	각 원소의 값보다 같거나 작은 정수 중 가장 큰 값
rint	각 원소의 소수자리를 반올림
modf	각 원소의 몫과 나머지를 각각의 배열로 반환
isfinite, isinf	각 원소가 유한한지, 무한한지. 불리언 배열로 반환
cos, cosh, sin, sinh, tan, tanh	일반 삼각함수, 쌍곡삼각함수
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	역삼각함수

유니버설 함수

```
arr1 = np.arange(10).reshape(2, 5)
arr2 = np.arange(-50, 50, 10).reshape(2, 5)
print(arr1)
print(arr2)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
[[-50 -40 -30 -20 -10]
 [ 0 10 20 30 40]]
```

```
np.maximum(arr1, arr2)
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5, 10, 20, 30, 40]])
```

유니버설 함수

Function	설명
add	두 배열의 같은 위치의 원소끼리 합함
subtract	첫번째 배열의 원소에서 두번째 배열의 원소를 뺌
multiply	같은 위치의 원소끼리 곱함
divide, floor_divide	첫번째 배열의 원소를 두번째 배열의 원소를 나눔. floor는 몫만 취함
power	첫번째 배열의 원소를 두번째 배열의 원소만큼 제곱함
maximum, fmax	각 배열의 두 원소 중 큰 값을 반환. fmax는 NaN 무시
minimum, fmin	각 배열의 두 원소 중 작은 값을 반환. fmax는 NaN 무시
mod	첫번째 배열의 원소를 두번째 배열의 원소로 나눈 나머지
copysign	첫번째 배열의 원소의 기호를 두번째 배열의 원소의 기호로 바꿈
greater, greater_equal, less, less_equal, less, less_equal,	각 두 원소 간의 비교 연산(<, <=, >, >=, ==, !=)를 불리언 배열로 반환

기본 통계 메서드

```
arr = np.random.randn(1000, 5000)  
arr.shape
```

```
(1000, 5000)
```

```
arr.mean() # ndarray의 메서드 이용
```

```
-2.67739981543579e-05
```

```
np.mean(arr) # numpy의 최상위 함수를 이용
```

```
-2.67739981543579e-05
```

- sum
- mean
- std, var
- min, max
- argmin, argmax
- cumsum
- cumprod
- cov
- corrcoef

기본 통계 메서드

```
arr = np.array([[10, 20, 30, 40], [10, 20, 30, 40]])  
arr
```

```
array([[10, 20, 30, 40],  
       [10, 20, 30, 40]])
```

```
arr.sum()
```

200

```
arr.sum(axis=0)
```

```
array([20, 40, 60, 80])
```

기본 통계 메서드

```
arr = np.array([[10, 20, 30, 40], [10, 20, 30, 40]])  
arr
```

```
array([[10, 20, 30, 40],  
       [10, 20, 30, 40]])
```

```
arr.sum(axis=0)
```

```
array([20, 40, 60, 80])
```

```
arr.sum(axis=1)
```

```
array([100, 100])
```

다양한 수학, 통계 연산

산술연산 - 벡터화

유니버설 함수

기본 통계 메서드

any, all

- **any** 원소 중 하나라도 참인 경우 True, 그렇지 않으면 False
- **all** 모든 원소가 참인 경우 True, 그렇지 않으면 False

```
arr = np.array([True, False, True])  
arr.any(), arr.all()
```

```
(True, False)
```

```
arr = np.array([0, 0, 0])  
arr.any(), arr.all()
```

```
(False, False)
```


where

- where(조건, T, F)

```
xarr = np.array([100, 200, 300, 400])  
yarr = np.array([1, 2, 3, 4])  
cond = np.array([True, False, True, False])
```

```
# 파이썬 기본  
result = [x if c else y  
          for x, y, c in zip(xarr, yarr, cond)]  
result
```

```
[100, 2, 300, 4]
```

```
# numpy 사용 => 큰 배열을 빠르게 처리 가능. 다차원일 때도 간결하게 표현 가능  
result = np.where(cond, xarr, yarr)
```

정렬

```
arr = np.random.randint(1, 100, size=10)  
arr
```

```
array([83, 18,  1, 67, 55, 61, 82, 69, 29, 81])
```

```
arr.sort()
```

```
arr
```

```
array([ 1, 18, 29, 55, 61, 67, 69, 81, 82, 83])
```

any, all 조건절 정렬

산술연산 - 벡터화

유니버설 함수

기본 통계 메서드

집합 관련 함수

선형대수

`np.dot(x, y)`

- dot product

$$\mathbf{a} = (a_1, a_2, \dots, a_n)$$

$$\mathbf{b} = (b_1, b_2, \dots, b_n)$$

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

선형대수

`np.dot(x, y)`

```
x = np.random.randint(-5, 5, size=10)  
y = np.random.randint(-10, 10, size=10)  
x.shape, y.shape
```

```
((10,), (10,))
```

```
x.dot(y)
```

```
-39
```

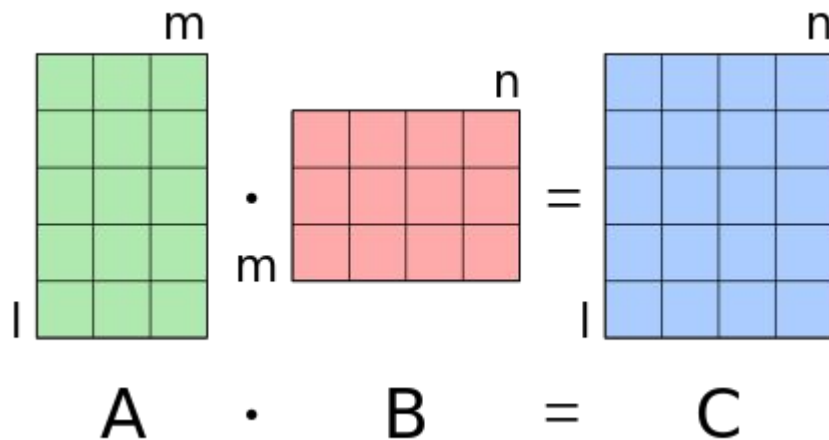
```
np.dot(x, y)
```

```
-39
```

선형대수

`np.matmul(x, y)`

- matrix multiplication



선형대수

`np.matmul(x, y)`

```
a = np.random.randint(-3, 3, 10).reshape(2, 5)
b = np.random.randint(0, 5, 15).reshape(5, 3)
a.shape, b.shape
```

```
((2, 5), (5, 3))
```

```
np.matmul(a, b)
```

```
array([[17,  6, 14],
       [-5,  8,  5]])
```

```
np.matmul(b, a)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-68-af3b88aa2232> in <module>()
----> 1 np.matmul(b, a)
```

```
ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufu
```

선형대수

`np.diag(x)`

- diagonal matrix

주대각선을 제외한 곳의 원소가 모두 0인 정사각행렬

$$\begin{bmatrix} -1 & 0 \\ 0 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 6 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$

선형대수

np.diag(x)

```
matrix = np.arange(25).reshape(5, 5)
```

```
np.diag(matrix)
```

```
array([ 0,  6, 12, 18, 24])
```

```
np.diag([10, 30, 50])
```

```
array([[10,  0,  0],  
       [ 0, 30,  0],  
       [ 0,  0, 50]])
```

선형대수

`np.linalg.det(x)`

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

```
matrix = np.array([[10, 5], [2, 3]])  
matrix
```

```
array([[10,  5],  
       [ 2,  3]])
```

```
np.linalg.det(matrix)
```

```
20.000000000000007
```

선형대수

np.linalg.inv(x)

$$\mathbf{A}^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\det \mathbf{A}} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$= \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

```
matrix = np.array([[10, 20], [30, 40]])  
matrix
```

```
array([[10, 20],  
       [30, 40]])
```

```
inv_matrix = np.linalg.inv(matrix)  
inv_matrix
```

```
array([[ -0.2 ,  0.1 ],  
       [ 0.15, -0.05]])
```

선형대수

`np.linalg.solve(A, b)`

$Ax = b$ 를 만족하는 x

```
A = np.array([[3, 1, -2], [0, -5, 7], [4, 0, 1]])  
b = np.array([-5, 1, -1])  
x = np.linalg.solve(A, b)  
x
```

```
array([-1., 4., 3.])
```

선형대수

dot, matmul

linalg.det

linalg.eig

linalg.inv

linalg.solve

Reference

numpy 공식문서

파이썬 라이브러리를 활용한 데이터 분석(웨스 맥키니)

edwith, 머신러닝을 위한 Python(최성철)