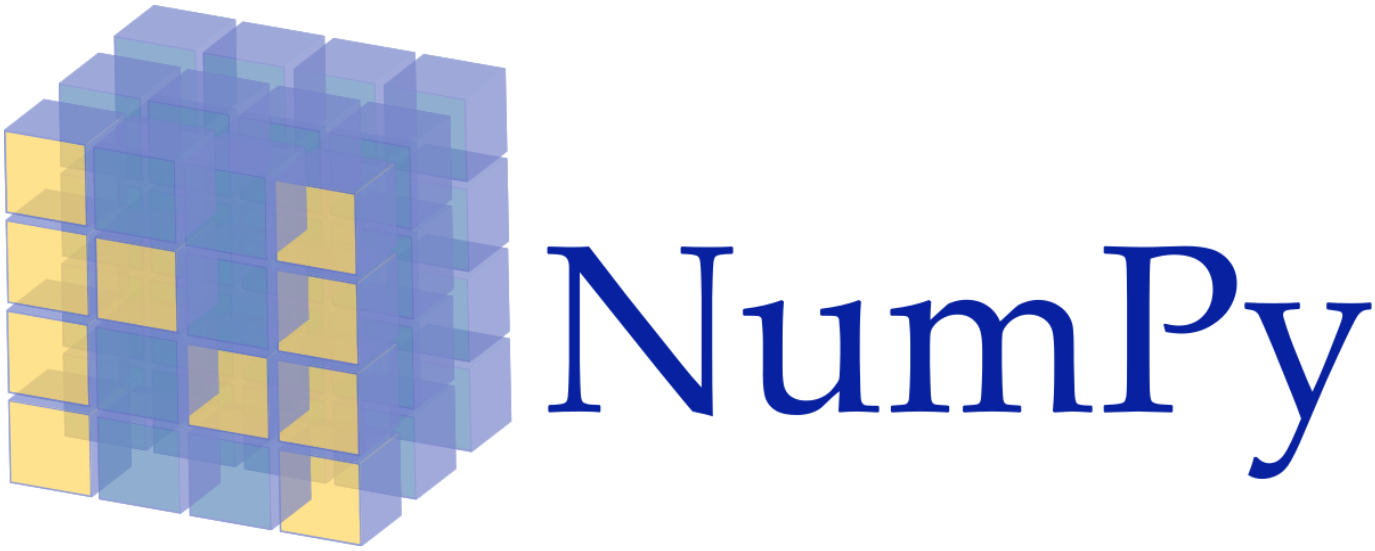
NumPy 한번에 제대로 배우기



NumPy 특징

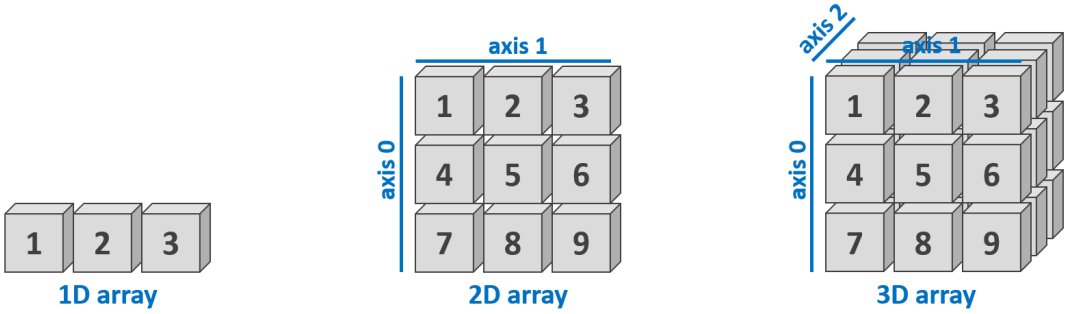
- Numerical Python의 약자
- 고성능 과학 계산용 패키지로 강력한 N차원 배열 객체
- 범용적 데이터 처리에 사용 가능한 다차원 컨테이너
- 정교한 브로드캐스팅(broadcasting) 기능
- 파이썬의 자료형 list와 비슷하지만, 더 빠르고 메모리를 효율적으로 관리
- 반복문 없이 데이터 배열에 대한 처리를 지원하여 빠르고 편리
- 데이터 과학 도구에 대한 생태계의 핵심을 이루고 있음

```
import numpy as np
np.__version__
```

```
'1.18.5'
```

배열 생성

리스트로 배열 만들기




```
a1 = np.array([1, 2, 3, 4, 5])
print(a1)
print(type(a1))
print(a1.shape)
print(a1[0], a1[1], a1[2], a1[3], a1[4])
a1[0] = 4
a1[1] = 5
a1[2] = 6
print(a1)
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
(5,)
1 2 3 4 5
[4 5 6 4 5]
```

```
a2 = np.array([ [1, 2, 3], [4, 5, 6], [7, 8, 9] ])
print(a2)
print(a2.shape)
print(a2[0, 0], a2[1, 1], a2[2, 2])
```

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

```
a3 = np.array([ [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ],
                [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ],
                [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ] ])
print(a3)
print(a3.shape)
```

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

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

 [[1 2 3]
```

```
  [4 5 6]
  [7 8 9]]]
(3, 3, 3)
```

배열 생성 및 초기화

- `zeros()`: 모든 요소를 0으로 초기화

```
np.zeros(10)
```

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

- `ones()`: 모든 요소를 1로 초기화

```
np.ones((3, 3))
```

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

- `full()`: 모든 요소를 지정한 값으로 초기화

```
np.full((3, 3), 1.23)
```

```
array([[1.23, 1.23, 1.23],
       [1.23, 1.23, 1.23],
       [1.23, 1.23, 1.23]])
```

- `eye()`: 단위행렬(identity matrix) 생성
  - 주대각선의 원소가 모두 1이고 나머지 원소는 모두 0인 정사각 행렬

```
np.eye(3)
```

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

- `tri()`: 삼각행렬 생성

```
np.tri(3)
```

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

- `empty()`: 초기화되지 않은 배열 생성
    - 초기화가 없어서 배열 생성비용 저렴하고 빠름
    - 초기화되지 않아서 기존 메모리 위치에 존재하는 값이 있음

```
np.empty(10)
```

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

- `_like()`: 지정된 배열과 shape가 같은 행렬 생성
    - `np.zeros_like()`
    - `np.ones_like()`
    - `np.full_like()`
    - `np.empty_like()`

```
print(a1)
np.zeros_like(a1)
```

```
[4 5 6 4 5]
array([0, 0, 0, 0, 0])
```

```
print(a2)
np.ones_like(a2)
```

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

```
print(a3)
np.full_like(a3, 10)
```

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

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

 [[1 2 3]
  [4 5 6]
  [7 8 9]]]
array([[[10, 10, 10],
        [10, 10, 10],
        [10, 10, 10]],

       [[10, 10, 10],
        [10, 10, 10],
```

```
        [10, 10, 10]],

       [[10, 10, 10],
        [10, 10, 10],
        [10, 10, 10]]])
```

생성한 값으로 배열 생성

- `arange()`: 정수 범위로 배열 생성

```
np.arange(0, 30, 2)
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28])
```

- `linspace()`: 범위 내에서 균등 간격의 배열 생성

```
np.linspace(0, 1, 5)
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

- `logspace()`: 범위 내에서 균등간격으로 로그 스케일로 배열 생성

```
np.logspace(0.1, 1, 20)
```

```
array([ 1.25892541,  1.40400425,  1.565802  ,  1.74624535,  1.94748304,
        2.1719114 ,  2.42220294,  2.70133812,  3.0126409 ,  3.35981829,
        3.74700446,  4.17881006,  4.66037703,  5.19743987,  5.79639395,
        6.46437163,  7.2093272 ,  8.04013161,  8.9666781 , 10.        ])
```

랜덤값으로 배열 생성

함수	설명
`seed`	난수 발생을 위한 시드(seed) 지정
`permutation`	순서를 임의로 바꾸거나 임의의 순열 반환
`shuffle`	리스트나 배열의 순서를 뒤섞음
`random`	랜덤한 수의 배열 생성
`rand`	균등분포에서 표본 추출
`randint`	주어진 최소/최대 범위의 난수 추출
`randn`	표준편차가 1, 평균값 0인 정규분포의 표본 추출
`binomial`	이항분포에서 표본 추출
`normal`	정규분포(가우시안)에서 표본 추출
`beta`	베타분포에서 표본 추출
`chisquare`	카이제곱분포에서 표본 추출
`gamma`	감마분포에서 표본 추출

함수	설명
uniform	균등(0, 1)분포에서 표본 추출

- `random.random()`: 랜덤한 수의 배열 생성

```
np.random.random((3, 3))
```

```
array([[0.6575939 , 0.803762  , 0.23075293],
       [0.32144283, 0.52211783, 0.46285086],
       [0.33538115, 0.9473045 , 0.34578796]])
```

- `random.randint()`: 일정 구간의 랜덤 정수의 배열 생성

```
np.random.randint(0, 10, (3, 3))
```

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

- `random.normal()`: 정규분포(normal distribution)를 고려한 랜덤한 수의 배열 생성
- 평균=0, 표준편차=1, 3 x 3 배열

```
np.random.normal(0, 1, (3, 3))
```

```
array([[-0.50034912, -0.46117753, -2.12922878],
       [ 0.03634214,  0.95389155, -0.53742495],
       [-0.19782743,  1.43871253,  0.97251717]])
```

- `random.rand()`: 균등분포(uniform distribution)를 고려한 랜덤한 수의 배열 생성

```
np.random.rand(3, 3)
```

```
array([[0.71117799, 0.13803711, 0.48245251],
       [0.30255013, 0.03036577, 0.45992232],
       [0.67583728, 0.70792018, 0.00261856]])
```

- `random.randn()`: 표준 정규 분포(standard normal distribution)를 고려한 랜덤한 수의 배열 생성

```
np.random.randn(3, 3)
```

```
array([[-0.67104357, -0.31466349,  0.42747638],
       [ 0.00205832,  0.67304152, -0.85492265],
       [ 0.89652484, -1.27892556,  0.0610578 ]])
```

표준 데이터 타입

데이터 타입	설명
bool_	바이트로 저장된 불리언(Boolean)으로 True 또는 False 값을 가짐
int_	기본 정수(Integer) 타입
intc	C 언어에서 사용되는 int 와 동일 (일반적으로 int32 또는 int64)
intp	인덱싱에 사용되는 정수 (C 언어에서 ssize_t 와 동일; 일반적으로 int32 또는 int64)
int8	바이트(Byte) (-128 ~ 127)
int16	정수 (-32768 ~ 32767)
int32	정수 (-2147483648 ~ 2147483647)
int64	정수(-9223372036854775808 ~ 9223372036854775807)
uint8	부호 없는 정수 (0 ~ 255)
uint16	부호 없는 정수 (0 ~ 65535)
uint32	부호 없는 정수 (0 ~ 4294967295)
uint64	부호 없는 정수 (0 ~ 18446744073709551615)
float16	반정밀 부동 소수점(Half precision float): 부호 비트, 5비트 지수, 10비트 가수
float32	단정밀 부동 소수점(Single precision float): 부호 비트, 8비트 지수, 23비트 가수
float64	배정밀 부동 소수점(Double precision float): 부호 비트, 11비트 지수, 52비트 가수
float_	float64 를 줄여서 표현
complex64	복소수(Complex number), 두 개의 32비트 부동 소수점으로 표현
complex128	복소수, 두 개의 64비트 부동 소수점으로 표현
complex_	complex128 를 줄여서 표현

```
np.zeros(20, dtype=int)
```

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

```
np.ones((3, 3), dtype=bool)
```

```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

```
np.full((3, 3), 1.0, dtype=float)
```

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

▾ 날짜/시간 배열 생성

코드	의미	상대적 시간 범위	절대적 시간 범위
Y	연	± 9.2e18 년	[9.2e18 BC, 9.2e18 AD]
M	월	± 7.6e17 년	[7.6e17 BC, 7.6e17 AD]
W	주	± 1.7e17 년	[1.7e17 BC, 1.7e17 AD]
D	일	± 2.5e16 년	[2.5e16 BC, 2.5e16 AD]

코드	의미	상대적 시간 범위	절대적 시간 범위
h	시	± 1.0e15 년	[1.0e15 BC, 1.0e15 AD]
m	분	± 1.7e13 년	[1.7e13 BC, 1.7e13 AD]
s	초	± 2.9e12 년	[2.9e9 BC, 2.9e9 AD]
ms	밀리초	± 2.9e9 년	[2.9e6 BC, 2.9e6 AD]
us	마이크로초	± 2.9e6 년	[290301 BC, 294241 AD]
ns	나노초	± 292 년	[1678 AD, 2262 AD]
ps	피코초	± 106 일	[1969 AD, 1970 AD]
fs	펨토초	± 2.6 시간	[1969 AD, 1970 AD]
as	아토초	± 9.2 초	[1969 AD, 1970 AD]

```
date = np.array('2020-01-01', dtype=np.datetime64)
date
```

```
array('2020-01-01', dtype='datetime64[D]')
```

```
date + np.arange(12)
```

```
array(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
       '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
       '2020-01-09', '2020-01-10', '2020-01-11', '2020-01-12'],
      dtype='datetime64[D]')
```

```
datetime = np.datetime64('2020-06-01 12:00')
datetime
```

```
numpy.datetime64('2020-06-01T12:00')
```

```
datetime = np.datetime64('2020-06-01 12:00:12.34', 'ns')
datetime
```

```
numpy.datetime64('2020-06-01T12:00:12.340000000')
```

배열 조회

배열 속성 정보

```
def array_info(array):
  print(array)
  print("ndim:", array.ndim)
  print("shape:", array.shape)
  print("dtype:", array.dtype)
  print("size:", array.size)
```

```
  print("itemsize:", array.itemsize)
  print("nbytes:", array.nbytes)
  print("strides:", array.strides)
```

```
array_info(a1)
```

```
[4 5 6 4 5]
ndim: 1
shape: (5,)
dtype: int64
size: 5
itemsize: 8
nbytes: 40
strides: (8,)
```

```
array_info(a2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
ndim: 2
shape: (3, 3)
dtype: int64
size: 9
itemsize: 8
nbytes: 72
strides: (24, 8)
```

```
array_info(a3)
```

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

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

 [[1 2 3]
  [4 5 6]
  [7 8 9]]]
ndim: 3
shape: (3, 3, 3)
dtype: int64
size: 27
itemsize: 8
nbytes: 216
strides: (72, 24, 8)
```

인덱싱(Indexing)

```
print(a1)
print(a1[0])
print(a1[2])
```

```
print(a1[-1])
print(a1[-2])
```

```
[4 5 6 4 5]
4
6
5
4
```

```
print(a2)
print(a2[0, 0])
print(a2[0, 2])
print(a2[1, 1])
print(a2[2, -1])
```

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

```
print(a3)
print(a3[0, 0, 0])
print(a3[1, 1, 1])
print(a3[2, 2, 2])
print(a3[2, -1, -1])
```

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

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

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

슬라이싱(Slicing)

- 슬라이싱 구문: `a[start:stop:step]`
- 기본값: start=0, stop=ndim, step=1

```
print(a1)
print(a1[0:2])
```

```
print(a1[0:])
print(a1[:1])
print(a1[::2])
print(a1[::-1])
```

```
[4 5 6 4 5]
[4 5]
[4 5 6 4 5]
[4]
[4 6 5]
[5 4 6 5 4]
```

```
print(a2)
print(a2[1])
print(a2[1, :])
print(a2[:2, :2])
print(a2[1:, ::-1])
print(a2[::-1, ::-1])
```

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

불리언 인덱싱(Boolean Indexing)

- 배열 각 요소의 선택 여부를 불리언(True or False)로 지정
- True 값인 인덱스의 값만 조회

```
print(a1)
bi = [False, True, True, False, True]
print(a1[bi])
bi = [True, False, True, True, False]
print(a1[bi])
```

```
[4 5 6 4 5]
[5 6 5]
[4 6 4]
```

```
print(a2)
bi = np.random.randint(0, 2, (3, 3), dtype=bool)
print(bi)
print(a2[bi])
```

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

팬시 인덱싱(Fancy Indexing)

```
print(a1)
print([a1[0], a1[2]])
ind = [0, 2]
print(a1[ind])
ind = np.array([[0, 1],
                [2, 0]])
print(a1[ind])
```

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

```
print(a2)
row = np.array([0, 2])
col = np.array([1, 2])
print(a2[row, col])
print(a2[row, :])
print(a2[:, col])
print(a2[row, 1])
print(a2[2, col])
print(a2[row, 1:])
print(a2[1:, col])
```

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

배열 값 삽입/수정/삭제/복사

배열 값 삽입

- `insert()`: 배열의 특정 위치에 값 삽입
- axis를 지정하지 않으면 1차원 배열로 변환
- 추가할 방향을 axis로 지정
- 원본 배열 변경없이 새로운 배열 반환

```
print(a1)
b1 = np.insert(a1, 0, 10)
print(b1)
c1 = np.insert(a1, 2, 10)
print(c1)
```

```
[4 5 6 4 5]
[10  4  5  6  4  5]
[ 4  5 10  6  4  5]
```

```
print(a2)
b2 = np.insert(a2, 1, 10, axis=0)
print(b2)
c2 = np.insert(a2, 1, 10, axis=1)
print(c2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[ 1  2  3]
 [10 10 10]
 [ 4  5  6]
 [ 7  8  9]]
[[ 1 10  2  3]
 [ 4 10  5  6]
 [ 7 10  8  9]]
```

배열 값 수정

- 배열의 인덱싱으로 접근하여 값 수정

```
print(a1)
a1[0] = 1
a1[1] = 2
a1[2] = 3
print(a1)
a1[:1] = 9
print(a1)
i = np.array([1, 3, 4])
```

```
a1[i] = 0
print(a1)
a1[i] += 4
print(a1)
```

```
[4 5 6 4 5]
[1 2 3 4 5]
[9 2 3 4 5]
[9 0 3 0 0]
[9 4 3 4 4]
```

```
print(a2)
a2[0, 0] = 1
a2[1, 1] = 2
a2[2, 2] = 3
a2[0] = 1
print(a2)
a2[1:, 2] = 9
print(a2)
row = np.array([0, 1])
col = np.array([1, 2])
a2[row, col] = 0
print(a2)
```

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

배열 값 삭제

- `delete()`: 배열의 특정 위치에 값 삭제
- axis를 지정하지 않으면 1차원 배열로 변환
- 삭제할 방향을 axis로 지정
- 원본 배열 변경없이 새로운 배열 반환

```
print(a1)
b1 = np.delete(a1, 1)
print(b1)
```

```
[9 4 3 4 4]
[9 3 4 4]
```

```
print(a2)
b2 = np.delete(a2, 1, axis=0)
print(b2)
c2 = np.delete(a2, 1, axis=1)
print(c2)
```

```
[[1 0 1]
 [4 2 0]
 [7 8 9]]
[[1 0 1]
 [7 8 9]]
[[1 1]
 [4 0]
 [7 9]]
```

배열 복사

- 리스트 자료형과 달리 배열의 슬라이스는 복사본이 아님

```
print(a2)
print(a2[:2, :2])
a2_sub = a2[:2, :2]
print(a2_sub)
a2_sub[:, 1] = 0
print(a2_sub)
print(a2)
```

```
[[1 0 1]
 [4 2 0]
 [7 8 9]]
[[1 0]
 [4 2]]
[[1 0]
 [4 2]]
[[1 0]
 [4 0]]
[[1 0 1]
 [4 0 0]
 [7 8 9]]
```

- `copy()`: 배열이나 하위 배열 내의 값을 명시적으로 복사

```
print(a2)
a2_sub_copy = a2[:2, :2].copy()
print(a2_sub_copy)
a2_sub_copy[:, 1] = 1
print(a2_sub_copy)
print(a2)
```

```
[[1 0 1]
 [4 0 0]
 [7 8 9]]
[[1 0]
```

```
 [4 0]]
[[1 1]
 [4 1]]
[[1 0 1]
 [4 0 0]
 [7 8 9]]
```

배열 변환

배열 전치 및 축 변경

```
print(a2)
print(a2.T)
```

```
[[1 0 1]
 [4 0 0]
 [7 8 9]]
[[1 4 7]
 [0 0 8]
 [1 0 9]]
```

```
print(a3)
print(a3.T)
```

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

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

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

 [[2 2 2]
  [5 5 5]
  [8 8 8]]

 [[3 3 3]
  [6 6 6]
  [9 9 9]]]
```

```
print(a2)
print(a2.swapaxes(1, 0))
```

```
[[1 0 1]
 [4 0 0]
 [7 8 9]]
[[1 4 7]
 [0 0 8]
 [1 0 9]]
```

```
print(a3)
print(a3.swapaxes(0, 1))
print(a3.swapaxes(1, 2))
```

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

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

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

 [[4 5 6]
  [4 5 6]
  [4 5 6]]

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

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

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

배열 재구조화

- `reshape()`: 배열의 형상을 변경

```
n1 = np.arange(1, 10)
print(n1)
print(n1.reshape(3, 3))
```

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

- `newaxis()`: 새로운 축 추가

```
print(n1)
print(n1[np.newaxis, :5])
print(n1[:5, np.newaxis])
```

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

배열 크기 변경

- 배열 모양만 변경

```
n2 = np.random.randint(0, 10, (2, 5))
print(n2)
n2.resize((5, 2))
print(n2)
```

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

- 배열 크기 증가
- 남은 공간은 0으로 채워짐

```
n2.resize((5, 5))
print(n2)
```

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

- 배열 크기 감소

- 포함되지 않은 값은 삭제됨

```
n2.resize((3, 3))
print(n2)
```

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

배열 추가

- `append()`: 배열의 끝에 값 추가

```
a2 = np.arange(1, 10).reshape(3, 3)
print(a2)
b2 = np.arange(10, 19).reshape(3, 3)
print(b2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[10 11 12]
 [13 14 15]
 [16 17 18]]
```

- axis 지정이 없으면 1차원 배열 형태로 변형되어 결합

```
c2 = np.append(a2, b2)
print(c2)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18]
```

- axis를 0으로 지정
- shape[0]을 제외한 나머지 shape은 같아야 함

```
c2 = np.append(a2, b2, axis=0)
print(c2)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]
 [16 17 18]]
```

- axis를 1로 지정
- shape[1]을 제외한 나머지 shape은 같아야 함

```
c2 = np.append(a2, b2, axis=1)
print(c2)
```

```
[[ 1  2  3 10 11 12]
 [ 4  5  6 13 14 15]
 [ 7  8  9 16 17 18]]
```

배열 연결

- `concatenate()`: 튜플이나 배열의 리스트를 인수로 사용해 배열 연결

```
a1 = np.array([1, 3, 5])
b1 = np.array([2, 4, 6])
np.concatenate([a1, b1])
```

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

```
c1 = np.array([7, 8, 9])
np.concatenate([a1, b1, c1])
```

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

```
a2 = np.array([[1, 2, 3],
               [4, 5, 6]])
np.concatenate([a2, a2])
```

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

```
a2 = np.array([[1, 2, 3],
               [4, 5, 6]])
np.concatenate([a2, a2], axis=1)
```

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

- `vstack()`: 수직 스택(vertical stack), 1차원으로 연결

```
np.vstack([a2, a2])
```

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

- `hstack()`: 수평 스택(horizontal stack), 2차원으로 연결

```
np.hstack([a2, a2])
```

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

- `dstack()`: 깊이 스택(depth stack), 3차원으로 연결

```
np.dstack([a2, a2])
```

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

       [[4, 4],
        [5, 5],
        [6, 6]]])
```

- `stack()`: 새로운 차원으로 연결

```
np.stack([a2, a2])
```

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

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

배열 분할

- `split()`: 배열 분할

```
a1 = np.arange(0, 10)
print(a1)
b1, c1 = np.split(a1, [5])
print(b1, c1)
b1, c1, d1, e1, f1 = np.split(a1, [2, 4, 6, 8])
print(b1, c1, d1, e1, f1)
```

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

- `vsplit()`: 수직 분할, 1차원으로 분할

```
a2 = np.arange(1, 10).reshape((3, 3))
print(a2)
b2, c2 = np.vsplit(a2, [2])
```

```
print(b2)
print(c2)
```

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

- `hsplit()`: 수평 분할, 2차원으로 분할

```
a2 = np.arange(1, 10).reshape((3, 3))
print(a2)
b2, c2 = np.hsplit(a2, [2])
print(b2)
print(c2)
```

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

- `dsplit()`: 깊이 분할, 3차원으로 분할

```
a3 = np.arange(1, 28).reshape((3, 3, 3))
print(a3)
b3, c3 = np.dsplit(a3, [2])
print(b3)
print(c3)
```

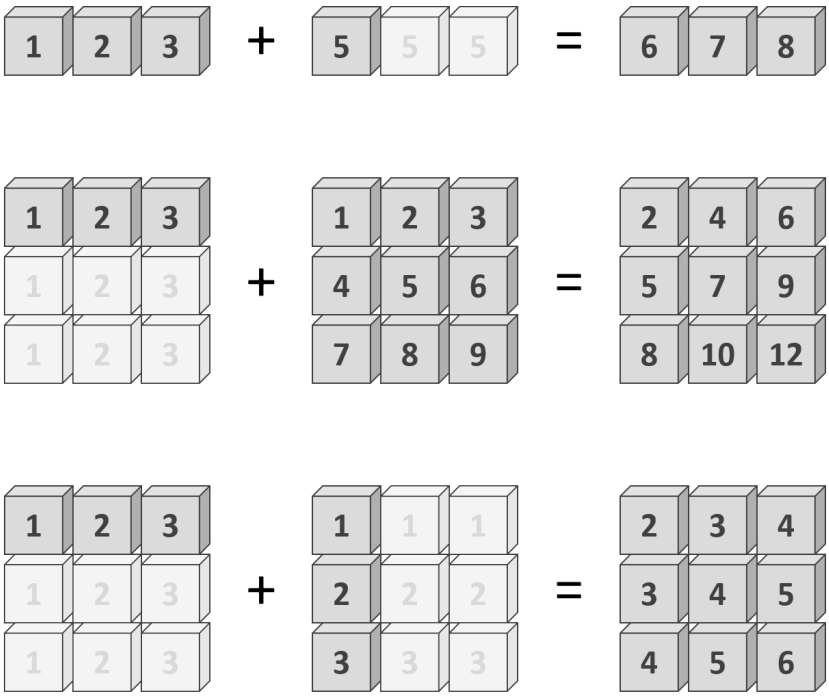
```
[[[ 1  2  3]
  [ 4  5  6]
  [ 7  8  9]]

 [[10 11 12]
  [13 14 15]
  [16 17 18]]

 [[19 20 21]
  [22 23 24]
  [25 26 27]]]
[[[ 1  2]
  [ 4  5]
  [ 7  8]]

 [[10 11]
  [13 14]
  [16 17]]
```

```
 [[19 20]
  [22 23]
  [25 26]]]
[[[ 3]
  [ 6]
  [ 9]]

 [[12]
  [15]
  [18]]

 [[21]
  [24]
  [27]]]
```

배열 연산

- NumPy의 배열 연산은 벡터화(vectorized) 연산을 사용
- 일반적으로 NumPy의 범용 함수(universal functions)를 통해 구현
- 배열 요소에 대한 반복적인 계산을 효율적으로 수행

브로드캐스팅(Broadcasting)




```
a1 = np.array([1, 2, 3])
print(a1)
print(a1 + 5)
```

```
a2 = np.arange(1, 10).reshape(3, 3)
print(a2)
print(a1 + a2)

b2 = np.array([1, 2, 3]).reshape(3, 1)
print(b2)
print(a1 + b2)
```

```
[1 2 3]
[6 7 8]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[ 2  4  6]
 [ 5  7  9]
 [ 8 10 12]]
[[1]
 [2]
 [3]]
[[2 3 4]
 [3 4 5]
 [4 5 6]]
```

산술 연산(Arithmetic Operators)

연산자	범용 함수	설명
+	np.add	덧셈
-	np.subtract	뺄셈
-	np.negative	단항 음수
*	np.multiply	곱셈
/	np.divide	나눗셈
//	np.floor_divide	나눗셈 내림
**	np.power	지수 연산
%	np.mod	나머지 연산

```
a1 = np.arange(1, 10)
print(a1)
print(a1 + 1)
print(np.add(a1, 10))
print(a1 - 2)
print(np.subtract(a1, 10))
print(-a1)
print(np.negative(a1))
print(a1 * 2)
print(np.multiply(a1, 2))
print(a1 / 2)
print(np.divide(a1, 2))
print(a1 // 2)
print(np.floor_divide(a1, 2))
print(a1 ** 2)
print(np.power(a1, 2))
```

```
print(a1 % 2)
print(np.mod(a1, 2))
```

```
[1 2 3 4 5 6 7 8 9]
[ 2  3  4  5  6  7  8  9 10]
[11 12 13 14 15 16 17 18 19]
[-1  0  1  2  3  4  5  6  7]
[-9 -8 -7 -6 -5 -4 -3 -2 -1]
[-1 -2 -3 -4 -5 -6 -7 -8 -9]
[-1 -2 -3 -4 -5 -6 -7 -8 -9]
[ 2  4  6  8 10 12 14 16 18]
[ 2  4  6  8 10 12 14 16 18]
[0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]
[0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]
[0 1 1 2 2 3 3 4 4]
[0 1 1 2 2 3 3 4 4]
[ 1  4  9 16 25 36 49 64 81]
[ 1  4  9 16 25 36 49 64 81]
[1 0 1 0 1 0 1 0 1]
[1 0 1 0 1 0 1 0 1]
```

```
a1 = np.arange(1, 10)
print(a1)
b1 = np.random.randint(1, 10, size=9)
print(b1)
print(a1 + b1)
print(a1 - b1)
print(a1 * b1)
print(a1 / b1)
print(a1 // b1)
print(a1 ** b1)
print(a1 % b1)
```

```
[1 2 3 4 5 6 7 8 9]
[1 8 9 4 1 4 6 9 5]
[ 2 10 12  8  6 10 13 17 14]
[ 0 -6 -6  0  4  2  1 -1  4]
[ 1 16 27 16  5 24 42 72 45]
[1.         0.25       0.33333333 1.         5.         1.5
 1.16666667 0.88888889 1.8       ]
[1 0 0 1 5 1 1 0 1]
[        1       256     19683       256         5      1296    117649
 134217728     59049]
[0 2 3 0 0 2 1 8 4]
```

```
a2 = np.arange(1, 10).reshape(3, 3)
print(a2)
b2 = np.random.randint(1, 10, size=(3, 3))
print(b2)
print(a2 + b2)
print(a2 - b2)
print(a2 * b2)
print(a2 / b2)
print(a2 // b2)
print(a2 ** b2)
print(a2 % b2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[6 5 6]
 [4 8 9]
 [9 7 7]]
[[ 7  7  9]
 [ 8 13 15]
 [16 15 16]]
[[-5 -3 -3]
 [ 0 -3 -3]
 [-2  1  2]]
[[ 6 10 18]
 [16 40 54]
 [63 56 63]]
[[0.16666667 0.4        0.5       ]
 [1.         0.625      0.66666667]
 [0.77777778 1.14285714 1.28571429]]
[[0 0 0]
 [1 0 0]
 [0 1 1]]
[[        1        32       729]
 [      256    390625  10077696]
 [40353607   2097152   4782969]]
[[1 2 3]
 [0 5 6]
 [7 1 2]]
```

절대값 함수(Absolute Function)

- `absolute`, `abs`, `fabs`: 내장된 절대값 함수

```
a1 = np.random.randint(-10, 10, size=5)
print(a1)
print(np.absolute(a1))
print(np.abs(a1))
```

```
[  3  -2   2 -10  -3]
[ 3  2  2 10  3]
[ 3  2  2 10  3]
```

제곱/제곱근 함수

- `square`, `sqrt` : 제곱, 제곱근 함수

```
print(a1)
print(np.square(a1))
print(np.sqrt(a1))
```

```
[  3  -2   2 -10  -3]
[  9   4   4 100   9]
[1.73205081        nan 1.41421356        nan        nan]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: invalid value
  This is separate from the ipykernel package so we can avoid doing imports until
```

지수와 로그 함수 (Exponential and Log Function)

```
a1 = np.random.randint(1, 10, size=5)
print(a1)
print(np.exp(a1))
print(np.exp2(a1))
print(np.power(a1, 2))
```

```
[6 3 8 1 4]
[4.03428793e+02 2.00855369e+01 2.98095799e+03 2.71828183e+00
 5.45981500e+01]
[ 64.   8. 256.   2.  16.]
[36  9 64  1 16]
```

```
print(a1)
print(np.log(a1))
print(np.log2(a1))
print(np.log10(a1))
```

```
[6 3 8 1 4]
[1.79175947 1.09861229 2.07944154 0.         1.38629436]
[2.5849625 1.5849625 3.        0.        2.       ]
[0.77815125 0.47712125 0.90308999 0.         0.60205999]
```

삼각 함수(Trigonometrical Function)

함수	설명
`np.sin`	요소 별 사인
`np.cos`	요소 별 코사인
`np.tan`	요소 별 탄젠트
`np.arcsin`	요소 별 아크 사인
`np.arccos`	요소 별 아크 코사인
`np.arctan`	요소 별 아크 탄젠트
`np.arctan2`	요소 별 아크 탄젠트
`np.sinh`	요소 별 하이퍼볼릭 사인
`np.cosh`	요소 별 하이퍼볼릭 코사인

함수	설명
np.tanh	요소 별 하이퍼볼릭 탄젠트
np.arcsinh	요소 별 하이퍼볼릭 아크 사인
np.arccosh	요소 별 하이퍼볼릭 아크 코사인
np.arctanh	요소 별 하이퍼볼릭 아크 탄젠트
np.deg2rad	요소 별 각도에서 라디안 변환
np.rad2deg	요소 별 라디안에서 각도 변환
np.hypot	요소 별 유클리드 거리 계산

```
t = np.linspace(0, np.pi, 3)
print(t)
print(np.sin(t))
print(np.cos(t))
print(np.tan(t))
```

```
[0.         1.57079633 3.14159265]
[0.0000000e+00 1.0000000e+00 1.2246468e-16]
[ 1.000000e+00  6.123234e-17 -1.000000e+00]
[ 0.00000000e+00  1.63312394e+16 -1.22464680e-16]
```

```
x = [-1, 0, 1]
print(x)
print(np.arcsin(x))
print(np.arccos(x))
print(np.arctan(x))
```

```
[-1, 0, 1]
[-1.57079633  0.          1.57079633]
[3.14159265 1.57079633 0.        ]
[-0.78539816  0.          0.78539816]
```

집계 함수(Aggregate Functions)

함수	NaN 안전 모드	설명
np.sum	np.nansum	요소의 합 계산
np.cumsum	np.nancumsum	요소의 누적 합
np.diff	N/A	요소의 차분
np.prod	np.nanprod	요소의 곱 계산
np.cumprod	np.nancumprod	요소의 누적 곱
np.dot	N/A	점 곱(dot product)
np.matmul	N/A	행렬 곱
np.tensordot	N/A	텐서곱(tensor product)
np.cross	N/A	벡터곱(cross product)
np.inner	N/A	내적(inner product)
np.outer	N/A	외적(outer product)
np.mean	np.nanmean	요소의 평균 계산

함수	NaN 안전 모드	설명
np.std	np.nanstd	표준 편차 계산
np.var	np.nanvar	분산 계산
np.min	np.nanmin	최소값
np.max	np.nanmax	최대값
np.argmin	np.nanargmin	최소값 인덱스
np.argmax	np.nanargmax	최대값 인덱스
np.median	np.nanmedian	중앙값
np.percentile	np.nanpercentile	요소의 순위 기반 백분위 수 계산
np.any	N/A	요소 중 참이 있는지 평가
np.all	N/A	모든 요소가 참인지 평가

sum(): 합 계산

```
a2 = np.random.randint(1, 10, size=(3, 3))
print(a2)
print(a2.sum(), np.sum(a2))
print(a2.sum(axis=0), np.sum(a2, axis=0))
print(a2.sum(axis=1), np.sum(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
39 39
[13 16 10] [13 16 10]
[17 15  7] [17 15  7]
```

cumsum(): 누적합 계산

```
print(a2)
print(np.cumsum(a2))
print(np.cumsum(a2, axis=0))
print(np.cumsum(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
[ 7 14 17 22 26 32 33 38 39]
[[ 7  7  3]
 [12 11  9]
 [13 16 10]]
[[ 7 14 17]
 [ 5  9 15]
 [ 1  6  7]]
```

diff(): 차분 계산

```

```

```
print(a2)
print(np.diff(a2))
print(np.diff(a2, axis=0))
print(np.diff(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
[[ 0 -4]
 [-1  2]
 [ 4 -4]]
[[-2 -3  3]
 [-4  1 -5]]
[[ 0 -4]
 [-1  2]
 [ 4 -4]]
```

- prod(): 곱 계산

```
print(a2)
print(np.prod(a2))
print(np.prod(a2, axis=0))
print(np.prod(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
88200
[ 35 140  18]
[147 120   5]
```

- cumprod(): 누적곱 계산

```
print(a2)
print(np.cumprod(a2))
print(np.cumprod(a2, axis=0))
print(np.cumprod(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
[    7    49   147   735  2940 17640 17640 88200 88200]
[[  7   7   3]
 [ 35  28  18]
 [ 35 140  18]]
[[  7  49 147]
 [  5  20 120]
 [  1   5   5]]
```

- dot()/matmul(): 점곱/행렬곱 계산

```

```

```
print(a2)
b2 = np.ones_like(a2)
print(b2)
print(np.dot(a2, b2))
print(np.matmul(a2, b2))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
[[1 1 1]
 [1 1 1]
 [1 1 1]]
[[17 17 17]
 [15 15 15]
 [ 7  7  7]]
[[17 17 17]
 [15 15 15]
 [ 7  7  7]]
```

tensordot(): 텐서곱 계산

```
print(a2)
print(b2)
print(np.tensordot(a2, b2))
print(np.tensordot(a2, b2, axes=0))
print(np.tensordot(a2, b2, axes=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
[[1 1 1]
 [1 1 1]
 [1 1 1]]
39
[[[[7 7 7]
   [7 7 7]
   [7 7 7]]

  [[7 7 7]
   [7 7 7]
   [7 7 7]]

  [[3 3 3]
   [3 3 3]
   [3 3 3]]]


 [[[5 5 5]
   [5 5 5]
   [5 5 5]]

  [[4 4 4]
   [4 4 4]
   [4 4 4]]

  [[6 6 6]
```

```
  [6 6 6]
  [6 6 6]]]


 [[[1 1 1]
   [1 1 1]
   [1 1 1]]

  [[5 5 5]
   [5 5 5]
   [5 5 5]]

  [[1 1 1]
   [1 1 1]
   [1 1 1]]]]
[[17 17 17]
 [15 15 15]
 [ 7  7  7]]
```

cross(): 벡터곱

```
x = [1, 2, 3]
y = [4, 5, 6]
print(np.cross(x, y))
```

```
[-3  6 -3]
```

inner()/outer(): 내적/외적

```
print(a2)
print(b2)
print(np.inner(a2, b2))
print(np.outer(a2, b2))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
[[1 1 1]
 [1 1 1]
 [1 1 1]]
[[17 17 17]
 [15 15 15]
 [ 7  7  7]]
[[7 7 7 7 7 7 7 7 7]
 [7 7 7 7 7 7 7 7 7]
 [3 3 3 3 3 3 3 3 3]
 [5 5 5 5 5 5 5 5 5]
 [4 4 4 4 4 4 4 4 4]
 [6 6 6 6 6 6 6 6 6]
 [1 1 1 1 1 1 1 1 1]
 [5 5 5 5 5 5 5 5 5]
 [1 1 1 1 1 1 1 1 1]]
```

- mean(): 평균 계산

```
print(a2)
print(np.mean(a2))
print(np.mean(a2, axis=0))
print(np.mean(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
4.333333333333333
[4.33333333 5.33333333 3.33333333]
[5.66666667 5.         2.33333333]
```

- std(): 표준 편차 계산

```
print(a2)
print(np.std(a2))
print(np.std(a2, axis=0))
print(np.std(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
2.160246899469287
[2.49443826 1.24721913 2.05480467]
[1.88561808 0.81649658 1.88561808]
```

- var(): 분산 계산

```
print(a2)
print(np.var(a2))
print(np.var(a2, axis=0))
print(np.var(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
4.666666666666667
[6.22222222 1.55555556 4.22222222]
[3.55555556 0.66666667 3.55555556]
```

- min(): 최소값

```
print(a2)
print(np.min(a2))
print(np.min(a2, axis=0))
print(np.min(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
1
[1 4 1]
[3 4 1]
```

- max(): 최대값

```
print(a2)
print(np.max(a2))
print(np.max(a2, axis=0))
print(np.max(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
7
[7 7 6]
[7 6 5]
```

- argmin(): 최소값 인덱스

```
print(a2)
print(np.argmin(a2))
print(np.argmin(a2, axis=0))
print(np.argmin(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
6
[2 1 2]
[2 1 0]
```

- argmax(): 최대값 인덱스

```
print(a2)
print(np.argmax(a2))
print(np.argmax(a2, axis=0))
print(np.argmax(a2, axis=1))
```

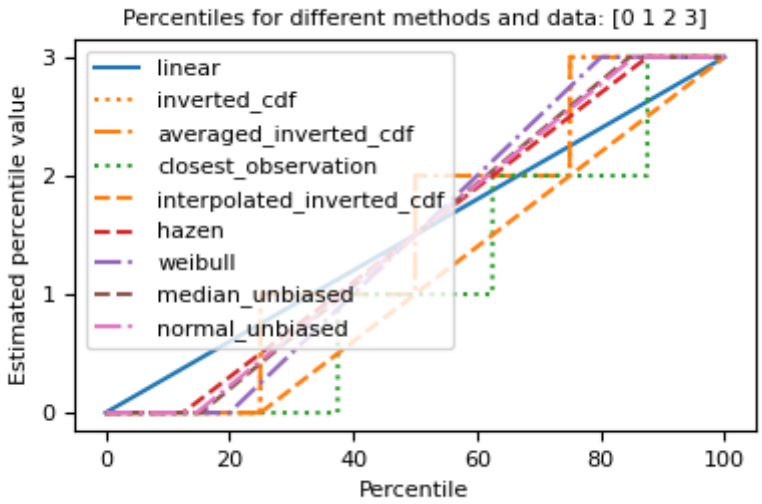
```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
0
[0 0 1]
[0 2 1]
```

median(): 중앙값

```
print(a2)
print(np.median(a2))
print(np.median(a2, axis=0))
print(np.median(a2, axis=1))
```

```
[[7 7 3]
 [5 4 6]
 [1 5 1]]
5.0
[5. 5. 3.]
[7. 5. 1.]
```

percentile(): 백분위 수




```
a1 = np.array([0, 1, 2, 3])
print(a1)
print(np.percentile(a1, [0, 20, 40, 60, 80, 100], interpolation='linear'))
print(np.percentile(a1, [0, 20, 40, 60, 80, 100], interpolation='higher'))
print(np.percentile(a1, [0, 20, 40, 60, 80, 100], interpolation='lower'))
print(np.percentile(a1, [0, 20, 40, 60, 80, 100], interpolation='nearest'))
print(np.percentile(a1, [0, 20, 40, 60, 80, 100], interpolation='midpoint'))
```

```
[0 1 2 3]
[0.  0.6 1.2 1.8 2.4 3. ]
[0 1 2 2 3 3]
[0 0 1 1 2 3]
[0 1 1 2 2 3]
[0.  0.5 1.5 1.5 2.5 3. ]
```

any()

```
a2 = np.array([[False, False, False],
               [False, True, True],
               [False, True, True]])
print(a2)
print(np.any(a2))
print(np.any(a2, axis=0))
print(np.any(a2, axis=1))
```

```
[[False False False]
 [False  True  True]
 [False  True  True]]
True
[False  True  True]
[False  True  True]
```

all()

```
a2 = np.array([[False, False, True],
               [True, True, True],
               [False, True, True]])
print(a2)
print(np.all(a2))
print(np.all(a2, axis=0))
print(np.all(a2, axis=1))
```

```
[[False False  True]
 [ True  True  True]
 [False  True  True]]
False
[False False  True]
[False  True False]
```

비교 연산(Comparison Operators)

연산자	비교 범용 함수
==	np.equal
!=	np.not_equal
<	np.less
<=	np.less_equal
>	np.greater
>=	np.greater_equal

```
a1 = np.arange(1, 10)
print(a1)
print(a1 == 5)
print(a1 != 5)
print(a1 < 5)
print(a1 <= 5)
print(a1 > 5)
print(a1 >= 5)
```

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

```
a2 = np.arange(1, 10).reshape(3, 3)
print(a2)
print(np.sum(a2))
print(np.count_nonzero(a2 > 5))
print(np.sum(a2 > 5))
print(np.sum(a2 > 5, axis=0))
print(np.sum(a2 > 5, axis=1))
print(np.any(a2 > 5))
print(np.any(a2 > 5, axis=0))
print(np.any(a2 > 5, axis=1))
print(np.all(a2 > 5))
print(np.all(a2 > 5, axis=0))
print(np.all(a2 > 5, axis=1))
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
45
4
4
[1 1 2]
[0 1 3]
True
[ True  True  True]
[False  True  True]
False
[False False False]
[False False  True]
```

비교 범용 함수	설명
`np.isclose`	배열 두개가 (z*1e+02)% 내외로 가까우면 True, 아니면 False
`np.isinf`	배열이 inf이면 True, 아니면 False
`np.isfinite`	배열이 inf, nan이면 False, 아니면 True
`np.isnan`	배열이 nan이면 True, 아니면 False

```
a1 = np.array([1, 2, 3, 4, 5])
print(a1)
b1 = np.array([1, 2, 3, 3, 4])
print(b1)
print(np.isclose(a1, b1))
```

```
[1 2 3 4 5]
[1 2 3 3 4]
[ True  True  True False False]
```

```
a1 = np.array([np.NaN, 2, np.Inf, 4, np.NINF])
print(a1)
print(np.isnan(a1))
print(np.isinf(a1))
print(np.isfinite(a1))
```

```
[ nan   2.  inf   4. -inf]
[ True False False False False]
[False False  True False  True]
[False  True False  True False]
```

불리언 연산자(Boolean Operators)

연산자	비교 범용 함수
&	np.bitwise_and
|	np.bitwise_or
^	np.bitwise_xor
~	np.bitwise_not

```
a2 = np.arange(1, 10).reshape(3, 3)
print(a2)

print((a2 > 5) & (a2 < 8))
print(a2[((a2 > 5) & (a2 < 8))])

print((a2 > 5) | (a2 < 8))
print(a2[((a2 > 5) | (a2 < 8))])

print((a2 > 5) ^ (a2 < 8))
print(a2[((a2 > 5) ^ (a2 < 8))])

print(~(a2 > 5))
print(a2[(~(a2 > 5))])
```

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

배열 정렬

```
a1 = np.random.randint(1, 10, size=10)
print(a1)
print(np.sort(a1))
print(a1)
print(np.argsort(a1))
print(a1)
print(a1.sort())
print(a1)
```

```
[8 3 3 8 6 9 3 5 6 6]
[3 3 3 5 6 6 6 8 8 9]
[8 3 3 8 6 9 3 5 6 6]
[1 2 6 7 4 8 9 0 3 5]
[8 3 3 8 6 9 3 5 6 6]
None
[3 3 3 5 6 6 6 8 8 9]
```

```
a2 = np.random.randint(1, 10, size=(3, 3))
print(a2)
print(np.sort(a2, axis=0))
print(np.sort(a2, axis=1))
```

```
[[9 8 8]
 [9 9 2]
 [7 3 1]]
[[7 3 1]
 [9 8 2]
 [9 9 8]]
[[8 8 9]
 [2 9 9]
 [1 3 7]]
```

부분 정렬

- `partition()`: 배열에서 k개의 작은 값을 반환

```
a1 = np.random.randint(1, 10, size=10)
print(a1)
print(np.partition(a1, 3))
```

```
[3 5 2 9 4 3 2 9 5 8]
[2 2 3 3 4 5 9 9 5 8]
```

```
a2 = np.random.randint(1, 10, size=(5, 5))
print(a2)
print(np.partition(a2, 3))
print(np.partition(a2, 3, axis=0))
print(np.partition(a2, 3, axis=1))
```

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

배열 입출력

함수	설명	파일 종류
np.save()	NumPy 배열 객체 1개를 파일에 저장	바이너리
np.savez()	NumPy 배열 객체 여러개를 파일에 저장	바이너리
np.load()	NumPy 배열 저장 파일로부터 객체 로딩	바이너리
np.loadtxt()	텍스트 파일로부터 배열 로딩	텍스트
np.savetxt()	텍스트 파일에 NumPy 배열 객체 저장	텍스트

```
a2 = np.random.randint(1, 10, size=(5, 5))
print(a2)
np.save("a", a2)
```

```
[[1 5 4 2 2]
 [5 1 8 9 7]
 [9 7 7 5 8]
 [6 5 1 6 7]
 [5 5 2 2 5]]
```

```
!ls
```

```
a.npy  sample_data
```

```
b2 = np.random.randint(1, 10, size=(5, 5))
print(b2)
np.savez("ab", a2, b2)
```

```
[[3 2 5 5 2]
 [2 4 5 7 1]
 [5 2 2 7 1]
```

```
 [4 1 6 6 4]
 [4 9 5 7 1]]
```

```
!ls
```

```
ab.npz  a.npy  sample_data
```

```
npy = np.load("a.npy")
print(npy)
```

```
[[1 5 4 2 2]
 [5 1 8 9 7]
 [9 7 7 5 8]
 [6 5 1 6 7]
 [5 5 2 2 5]]
```

```
npz = np.load("ab.npz")
print(npz.files)
print(npz['arr_0'])
print(npz['arr_1'])
```

```
['arr_0', 'arr_1']
[[1 5 4 2 2]
 [5 1 8 9 7]
 [9 7 7 5 8]
 [6 5 1 6 7]
 [5 5 2 2 5]]
[[3 2 5 5 2]
 [2 4 5 7 1]
 [5 2 2 7 1]
 [4 1 6 6 4]
 [4 9 5 7 1]]
```

```
print(a2)
np.savetxt("a.csv", a2, delimiter=',')
```

```
[[1 5 4 2 2]
 [5 1 8 9 7]
 [9 7 7 5 8]
 [6 5 1 6 7]
 [5 5 2 2 5]]
```

```
!ls
```

```
ab.npz  a.csv  a.npy  sample_data
```

```
!cat a.csv
```

```
1.0000000000000000e+00,5.0000000000000000e+00,4.0000000000000000e+00,2.0000000000000000
5.0000000000000000e+00,1.0000000000000000e+00,8.0000000000000000e+00,9.0000000000000000
9.0000000000000000e+00,7.0000000000000000e+00,7.0000000000000000e+00,5.0000000000000000
6.0000000000000000e+00,5.0000000000000000e+00,1.0000000000000000e+00,6.0000000000000000
5.0000000000000000e+00,5.0000000000000000e+00,2.0000000000000000e+00,2.0000000000000000
```

```
csv = np.loadtxt("a.csv", delimiter=',')
print(csv)
```

```
[[1. 5. 4. 2. 2.]
 [5. 1. 8. 9. 7.]
 [9. 7. 7. 5. 8.]
 [6. 5. 1. 6. 7.]
 [5. 5. 2. 2. 5.]]
```

```
print(b2)
np.savetxt("b.csv", b2, delimiter=',', fmt='%.2e', header='c1, c2, c3, c4, c5')
```

```
[[3 2 5 5 2]
 [2 4 5 7 1]
 [5 2 2 7 1]
 [4 1 6 6 4]
 [4 9 5 7 1]]
```

```
!cat b.csv
```

```
# c1, c2, c3, c4, c5
3.00e+00,2.00e+00,5.00e+00,5.00e+00,2.00e+00
2.00e+00,4.00e+00,5.00e+00,7.00e+00,1.00e+00
```