Numpy

22.07.12/ DSL 7기 전혜령

0. 목차

1. Array 생성

0. Array란?

List와의 차이 (1)

- 리스트는 여러가지 자료형 허용, 내부 배열에서 원소 개수 달라도 ok
- 배열은 한 가지 자료형만 허용, 내부 배열 내 원소 개수가 모두 같아야 함

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

```
'1.21.6'
```

```
l1 = [1, 3, 5, 'a', 'b'] # 1, 3, 5는 숫자형, 'a', 'b'는 문자열
a1 = np.array([1, 3, 5, 'a', 'b']) # '1', '3', '5', 'a', 'b'의 문자열로 전환
print(a1.dtype)
```

```
<U21
```

1. Array 생성

0. Array란?

List와의 차이 (1)

- 리스트는 여러가지 자료형 허용, 내부 배열에서 원소 개수 달라도 ok
- 배열은 한 가지 자료형만 허용, 내부 배열 내 원소 개수가 모두 같아야 함

```
l1 = [[1], [3, 5], [2, 4, 6]] # 문제 X
a1 = np.array([[1], [3, 5], [2, 4, 6]]) # 개수가 달라서 array 선언 불가능
# array([list([1]), list([3, 5]), list([2, 4, 6])], dtype=object)
print(l1)
print(a1)
```

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

1. Array 생성

0. Array란?

List와의 차이 (2)

- 리스트는 덧셈 시 항목을 이어 붙이는 concatenate를 수행
- 배열은 덧셈 시 항목 간 덧셈을 수행

```
# 리스트 연산
l1 = [1, 3, 5]
l2 = [2, 4, 6]

print(l1 + l2)
print(l1 * 3)
```

답:

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

```
# 넘파이 어레이 연산
a1 = np.array([1, 3, 5])
a2 = np.array([2, 4, 6])
```

```
print(a1 + a2 )
print(a1 * a2 )
print(a1 + 2)
print(a1 * 3 )
```

답:

```
[ 3  7 11]
[ 2 12 30]
[3 5 7]
[ 3  9 15]
```

1. Array 생성

0. Array란?

List와의 차이 (3)

- 자료형 종류가 다르기 때문에 자료형에서 지원하는 메소드의 종류에도 차이 존재

(ex. 리스트에서는 mean, argmax, round 등 어레이 메소드를 지원하지 않고,
배열에서는 append, remove, extend 등 리스트 메소드를 지원하지 않음.)

- 배열이 연산이 더 빠름

```
import time

# 리스트 연산 속도
a = list(range(10 ** 8))

start = time.time()
for i in range(10 ** 8):
    a[i] *= 2
end = time.time()
print(end - start) # 약 18.41초
```

```
# 어레이 연산 속도
a = np.array(range(10 ** 8))

start = time.time()
a = a * 2
end = time.time()
print(end - start) # 약 1.14초
```

```
18.414382934570312
1.1420695781707764
```

1. Array 생성

0. Array란?

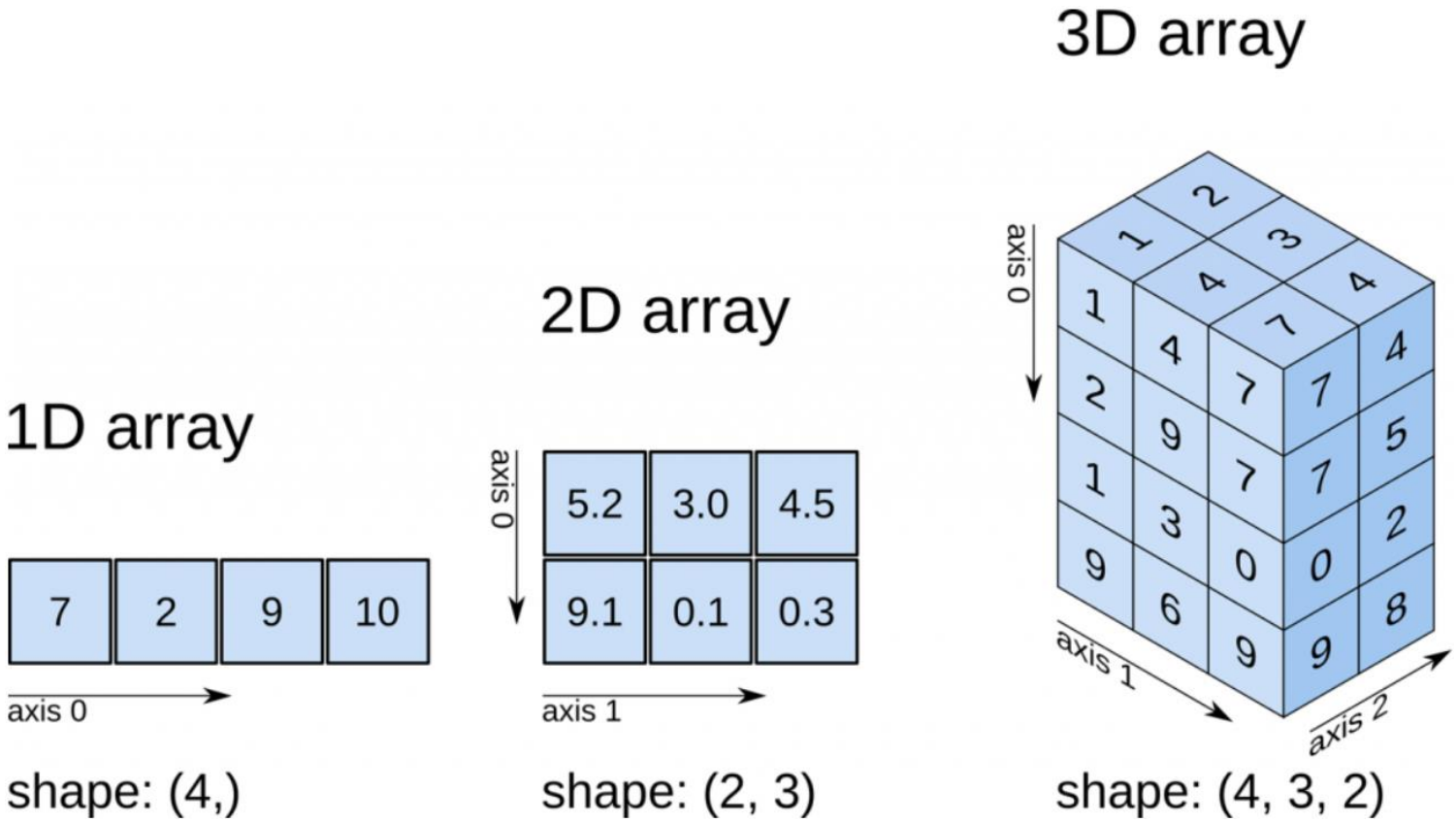
Pandas와의 차이

- Pandas는 여러 복합적인 데이터를 보고(시각화) 분석할 때 도움
- Numpy는 values들을 연산을 통해 다룰 때

1. Array 생성

- **배열 생성하기**

(0) 배열의 차원



1. Array 생성

- **배열 생성하기**

(1) 1차원 배열

```
a1 = np.array([1,2,3,4,5])
print(a1)
print(type(a1)) #nd = n dimensional n차원
print(a1.shape) # 5개의 element 존재하는 1차원 배열
print(a1[0], a1[1], a1[2]) # list와 같이 인덱스로 접근 가능
a1[0] = 4
print(a1)
```

결과:

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

1. Array 생성

- **배열 생성하기**

(2) 2, 3차원 배열

```
a2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a2)
print(a2.shape) # 3 by 3의 array
print(a2[0,0], a2[0,1]) # 인덱스로 접근 가능
```

결과:

```
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)
```

결과:

```
[[[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)
```

1. Array 생성

- **배열 생성하기**

(3) ndim( ), shape( )

- ndim: 배열의 차원 수
- shape: 배열의 차원을 튜플로 표시

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

```
print(a1.ndim)
print(a1.shape)
```

결과:

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

```
print(a2.ndim)
print(a2.shape)
```

결과:

1. Array 생성

- **배열 생성 함수들**

(1) 초기화 함수

- zeros( ): 모든 요소를 0으로 초기화
- ones( ): 모든 요소를 1로 초기화
- full( ): 모든 요소를 지정한 값으로 초기화

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

결과:

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

```
np.ones(10)
```

결과:

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

```
np.full((3,3), 1.5)
```

결과:

```
array([[1.5, 1.5, 1.5],
       [1.5, 1.5, 1.5],
       [1.5, 1.5, 1.5]])
```

1. Array 생성

- 배열 생성 함수들

(2) 단위 행렬 생성

- eye( ): 단위행렬(identity matrix) 생성

  + 주대각선의 원소가 모두 1이고 나머지 원소는 모두 0인 정사각행렬

```
np.eye(3)
# 정사각행렬이므로 size만 지정
```

결과:

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

(3) 같은 shape 행렬 생성

`_like( )`: 지정된 배열과 shape가 같은 행렬 생성

+ `np.zeros_like( )`
+ `np.ones_like( )`
+ `np.full_like( )`

```
print(a1)
np.zeros_like(a1) #a1과 같은 shape의 배열 생성
```

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

1. Array 생성

- **배열 생성 함수들**

(4) 범위 지정 행렬 생성

- arange( ): 정수 범위로 배열 생성
- linspace( ): 범위 내에서 균등 간격의 배열 생성
- logspace( ): 범위 내에서 균등간격으로 로그 스케일로 배열 생성

```
np.arange(0,30,2)
#np.arange(0,30,2).reshape(3,5)
```

결과:

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

```
np.linspace(0,1,5) # 0부터 1까지 균등하게 5개로 나눠줘
```

결과:

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

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

결과:

```
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.        ])
```

1. Array 생성

- **배열 생성 함수들**

(5) 랜덤값 이용하여 행렬 생성

- random.random( ): 랜덤한 수의 배열 생성
- random.randint( ): 일정 구간의 랜덤 정수의 배열 생성
- random.normal( ): 정규분포(normal distribution)를 고려한 랜덤한 수의 배열 생성

```
np.random.random((3,3)) # 실수값 반환
```

```
array([[0.04732586, 0.83772451, 0.22877097],
       [0.15577746, 0.95388896, 0.6658491 ],
       [0.3898443 , 0.98612763, 0.53938792]])
```

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

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

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

```
array([[-0.21487574, -0.4839892 ,  1.38124562],
       [ 1.14280215, -1.30456212,  0.31445083],
       [-0.6503469 , -1.07499843, -0.02582802]])
```

1. Array 생성

- 날짜/ 시간 배열 생성하기

```
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')
```

2. 연산

- **브로드 캐스팅**

모양이 다른 배열들 간의 연산이 어떤 조건을 만족했을 때 가능해지도록 배열 자동 변환

(조건)

1. 값이 하나인 배열은 어떤 배열에나 브로드캐스팅(Broadcasting)이 가능

(단, 값이 하나도 없는 빈 배열을 제외)

ex) 4x4 + 1

1	2	3	4
2	5	6	7
8	9	10	11
12	13	14	15

+ [1] =

2	3	4	5
3	6	7	8
9	10	11	12
13	14	15	16

2. 연산

- **브로드 캐스팅**

모양이 다른 배열들 간의 연산이 어떤 조건을 만족했을 때 가능해지도록 배열 자동 변환

(조건)

2. 하나의 배열의 차원이 1인 경우 브로드캐스팅(Broadcasting)이 가능

ex) 4x4 + 1x4

1	2	3	4
2	5	6	7
8	9	10	11
12	13	14	15

+

3	3	3	3

=

4	5	6	7
5	8	9	10
11	12	13	14
15	16	17	18

2. 연산

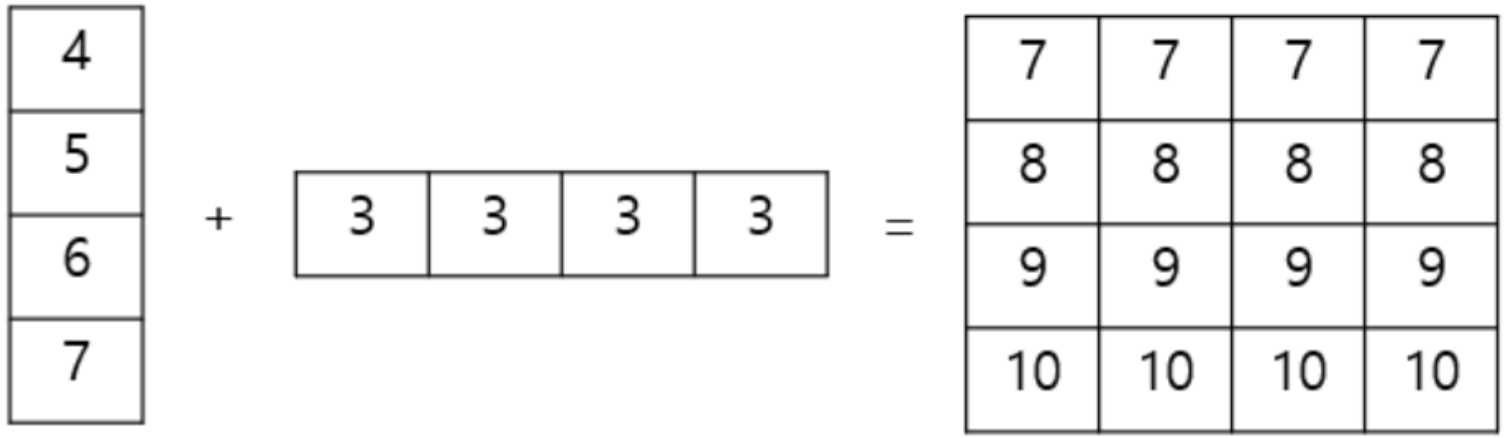
- **브로드 캐스팅**

모양이 다른 배열들 간의 연산이 어떤 조건을 만족했을 때 가능해지도록 배열 자동 변환

(조건)

3. 차원의 짝이 맞을 때 브로드캐스팅(Broadcasting)가능

ex) 4x1 + 1x4



2. 연산

- **브로드 캐스팅**

모양이 다른 배열들 간의 연산이 어떤 조건을 만족했을 때 가능해지도록 배열 자동 변환

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

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

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

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

결과:

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

2. 연산

- **산술 연산**

종 류	의 미
+	숫자를 덧셈하거나 문자열을 결합
-	좌항을 우항으로 뺌(또는 부호 변경)
*	숫자를 곱하거나, 문자열을 곱한 수 만큼 반복하여 결합
**	좌항을 우항으로 거듭 제곱
/	좌항을 우항으로 나눔(실수형)
//	좌항을 우항으로 나눔(정수형)
%	좌항을 우항으로 나눈 나머지

2. 연산

- **산술 연산**

```
a1 = np.arange(1,10)
print(a1)
print(a1+1)
print(np.add(a1,1)) # 연산자와 동일한 역할을 하는 범용 함수 존재
print(a1-2)
print(np.subtract(a1,2))
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))
```

결과:

```
[1 2 3 4 5 6 7 8 9]
[ 2  3  4  5  6  7  8  9 10]
[ 2  3  4  5  6  7  8  9 10]
[-1  0  1  2  3  4  5  6  7]
[-1  0  1  2  3  4  5  6  7]
[-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]
```

2. 연산

- **산술 연산**

```
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)
```

결과:

```
[1 2 3 4 5 6 7 8 9]
[7 9 4 9 3 5 5 7 4]
[ 8 11  7 13  8 11 12 15 13]
[-6 -7 -1 -5  2  1  2  1  5]
[ 7 18 12 36 15 30 35 56 36]
[0.14285714 0.22222222 0.75       0.44444444 1.66666667 1.2
 1.4        1.14285714 2.25      ]
[0 0 0 0 1 1 1 1 2]
[1 2 3 4 2 1 2 1 1]
```

2. 연산

- **산술 연산**

여러가지 함수들

- square( ), sqrt( ): 제곱, 제곱근 함수
- Absolute( ), abs( ): 내장된 절대값 함수

```
a1 = np.array([1,4,9,16,25])
print(a1)
print(np.square(a1))
print(np.sqrt(a1))
```

```
[ 1  4  9 16 25]
[  1  16  81 256 625]
[1. 2. 3. 4. 5.]
```

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

```
[ 9 -8 -3  0  7]
[9 8 3 0 7]
```

2. 연산

- **산술 연산**

여러가지 함수들

- prod( ): 곱 계산
- cumprod( ): 누적곱 계산
- dot()/matmul( ): 점곱/행렬곱 계산
- tensordot( ): 텐서곱 계산
- cross( ): 벡터곱

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

```
[[9 9 4]
 [6 7 2]
 [6 7 3]]
3429216
[324 441  24]
[324  84 126]
```

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

```
[[9 9 4]
 [6 7 2]
 [6 7 3]]
[[1 1 1]
 [1 1 1]
 [1 1 1]]
[[22 22 22]
 [15 15 15]
 [16 16 16]]
[[22 22 22]
 [15 15 15]
 [16 16 16]]
```

2. 연산

- **산술 연산**

여러가지 함수들

- 지수와 로그 함수 (Exponential and Log Function)

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

```
[3 7 3 8 3]
[  20.08553692 1096.63315843   20.08553692 2980.95798704   20.08553692]
[ 9 49  9 64  9]
```

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

```
[ 1  4  9 16 25]
[0.         1.38629436 2.19722458 2.77258872 3.21887582]
[0.         2.         3.169925   4.         4.64385619]
```

2. 연산

- **산술 연산**

여러가지 함수들

- 삼각 함수 (Trigonometrical Function)

```
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]
```

2. 연산

- **비교 연산**

종 류	의 미
==	좌항과 우항의 값이 같다
!=	좌항과 우항의 값이 다르다
>, <	좌항 또는 우항이 크거나 작다
>=, <=	좌항 또는 우항이 크거나 같고 작거나 같다

2. 연산

- **비교 연산**

```
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)
```

결과:

```
[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]
```

2. 연산

- **비교 연산 (+)**

```
# isclose()
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]
```

```
# inf, nan, NINF
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]
```

2. 연산

- 비교 연산 (+)
  - any(): 하나라도 True이면 True
  - all(): 하나라도 False이면 False

```
# 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],
               [False, 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]
 [False  True  True]
 [False  True  True]]
False
[False False  True]
[False False False]
```

2. 연산

- **불리언 연산자(Boolean Operators)**

```
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)])
```

결과:

```
[[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]
```

2. 연산

- **집계 함수**
  - sum( ): 합 계산
  - cumsum( ): 누적합 계산

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

```
[[9 9 4]
 [6 7 2]
 [6 7 3]]
53
[21 23  9] [21 23  9]
[22 15 16] [22 15 16]
```

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

```
[[9 9 4]
 [6 7 2]
 [6 7 3]]
[ 9 18 22 28 35 37 43 50 53]
[[ 9  9  4]
 [15 16  6]
 [21 23  9]]
[[ 9 18 22]
 [ 6 13 15]
 [ 6 13 16]]
```

2. 연산

- **집계 함수**
  - mean( ): 평균 계산
  - var( ): 분산 계산
  - std( ): 표준 편차 계산

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

```
[[9 9 4]
 [6 7 2]
 [6 7 3]]
5.888888888888889
[7.         7.66666667 3.        ]
[7.33333333 5.         5.33333333]
```

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

```
[[9 9 4]
 [6 7 2]
 [6 7 3]]
5.432098765432098
[2.         0.88888889 0.66666667]
[5.55555556 4.66666667 2.88888889]
```

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

```
[[9 9 4]
 [6 7 2]
 [6 7 3]]
2.3306863292670035
[1.41421356 0.94280904 0.81649658]
[2.3570226  2.1602469  1.69967317]
```

2. 연산

- **집계 함수**
  - min( ): 최소값
  - max( ): 최대값
  - argmin( ): 최소값 인덱스
  - argmax( ): 최대값 인덱스

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

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

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

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

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

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

2. 연산

- **집계 함수**
  - median( ): 중앙값
  - percentile( ): 백분위 수

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

```
[[9 9 4]
 [6 7 2]
 [6 7 3]]
6.0
[6. 7. 3.]
[9. 6. 6.]
```

```
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. ]
```

3. Indexing & Slicing

- **인덱싱(Indexing)**

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

결과:

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

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

결과:

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

```
print(a3)
print(a3[0, 0, 0])
print(a3[1, 0, 2])
print(a3[-1, -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
3
9
```

3. Indexing & Slicing

- **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]) # 0부터 1 전까지
print(a1[::2])
print(a1[::-1]) # 역으로 접근
```

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

결과:

```
print(a2)
print(a2[1]) # 1행 출력
print(a2[1, :]) # 1행 출력
print(a2[:2, :2]) # 0,1 출력
print(a2[1:, ::-1])
print(a2[::-1, ::-1])
```

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

결과:

3. Indexing & Slicing

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

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

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

```
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 False  True]
 [False  True  True]
 [False False False]]
[1 3 5 6]
```

3. Indexing & Slicing

- **팬시 인덱싱(Fancy Indexing)**
  - 배열에 인덱스 값을 넣어 반환

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

```
[ nan   2.  inf   4. -inf]
[nan, inf]
[nan inf]
[[nan  2.]
 [inf nan]]
```

- fancy indexing + slicing

```
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[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 3]
 [8 9]]
[[5 6]
 [8 9]]
```

3. Indexing & Slicing

- **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)
```

```
[ nan   2.  inf   4. -inf]
[  1.   2.   3.   4. -inf]
[  9.   2.   3.   4. -inf]
[9. 0. 3. 0. 0.]
[9. 4. 3. 4. 4.]
```

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

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

4. 함수

- ## 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)
print(a1)
```

결과:

```
[9. 4. 3. 4. 4.]
[10.  9.  4.  3.  4.  4.]
[9. 4. 3. 4. 4.]
```

```
print(a2)
b2 = np.insert(a2, 1, 10, axis = 0)
print(b2)
c2 = np.insert(a2, 1, [1,2,10], axis = 1)
print(c2)
```

```
[[0 1 1]
 [4 5 9]
 [7 0 9]]
[[ 0  1  1]
 [10 10 10]
 [ 4  5  9]
 [ 7  0  9]]
[[ 0  1  1  1]
 [ 4  2  5  9]
 [ 7 10  0  9]]
```

4. 함수

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

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

결과:

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

```
print(a2)
b2 = np.delete(a2, 1, axis = 0)
print(b2)
c2 = np.delete(a2, 1, axis = 1)
print(c2)
# 원본 배열 변경 없음 -> 속도 면에서 장점
```

```
[[0 1 1]
 [4 5 9]
 [7 0 9]]
[[0 1 1]
 [7 0 9]]
[[0 1]
 [4 9]
 [7 9]]
```

- **배열 복사**
  - **copy( ): 배열이나 하위 배열 내의 값을 명시적으로 복사**

```
print(a2)
print(a2[:2, :2])
a2_sub = a2[:2, :2]
print(a2_sub)
a2_sub[:, 1] = 0
# 받아온 배열도 원본 배열에 영향을 줌 -> 동일한 메모리 위치 사용
# 영향 안주고 싶으면 copy해야 함
print(a2_sub)
print(a2)
```

결과:

```
[[0 1 1]
 [4 5 9]
 [7 0 9]]
[[0 1]
 [4 5]]
[[0 1]
 [4 5]]
[[0 0]
 [4 0]]
[[0 0 1]
 [4 0 9]
 [7 0 9]]
```

- **배열 복사**
  - **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)
```

결과:

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

4. 함수

- **배열 재구조화**
  - reshape( ): 배열의 형상을 변경
  - newaxis( ): 새로운 축 추가

```
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]]
```

```
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]]
```

4. 함수

- **배열 재구조화**
  - resize( ): 배열의 크기를 변경

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

```
[[0 6 7 8 0]
 [1 0 0 2 2]]
[[0 6]
 [7 8]
 [0 1]
 [0 0]
 [2 2]]
```

```
n2.resize((5,5),refcheck=False)
print(n2)
# 남은 공간 0으로 채워짐
```

```
[[0 6 7 8 0]
 [1 0 0 2 2]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

```
n2.resize((3,3),refcheck=False)
print(n2)
# 포함되지 않은 값은 삭제됨
```

```
[[0 6 7]
 [8 0 1]
 [0 0 2]]
```

- 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]]
```

```
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]
```

```
# axis = 0으로 지정
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]]
```

```
# axis = 1로 지정
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]]
```

4. 함수

- **배열 값 확인**
  - np.where( ): 조건 만족 인덱스 반환
  - np.unique( ): 주어진 배열에서 모든 고유 값을 검색하고 이러한 고유 값을 정렬

```
a1 = np.arange(5, 15)
print(a1)
print(np.where(a > 10))
```

```
[ 5  6  7  8  9 10 11 12 13 14]
(array([6, 7, 8, 9]),)
```

```
a2 = np.array([[2,3,4],
               [5,4,7],
               [4,2,3]])
unique_array=np.unique(a2, return_counts=True)
print(unique_array)
```

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

5. 병합 & 분할 (stack, split)

- **배열 병합**
  - 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])
```

```
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]])
```

5. 병합 & 분할 (stack, split)

- **배열 병합**
  - vstack( ): 수직 스택(vertical stack), row로 연결
  - hstack( ): 수평 스택(horizontal stack), col로 연결
  - dstack( ): 깊이 스택(depth stack), depth로 연결
  - stack( ): 새로운 차원으로 연결

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

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

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

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

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

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

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

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

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

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

5. 병합 & 분할 (stack, split)

- **배열 분할**
  - **split( ): 배열 분할**

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

결과:

```
[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]
```

5. 병합 & 분할 (stack, split)

- 배열 분할
  - vsplit( ): 수직 분할
  - hsplit( ): 수평 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]]
```

```
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

```
a3 = np.arange(1,28).reshape(3,3,3)
print(a3)
b3, c3 = np.hsplit(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  3]
  [ 4  5  6]]

 [[10 11 12]
  [13 14 15]]

 [[19 20 21]
  [22 23 24]]]
[[[ 7  8  9]]

 [[16 17 18]]

 [[25 26 27]]]
```

6. Summary

Summary

- **Quiz 1 : 값이 10~30인 배열 만들기**

```
q1 = np.arange(10,31)
print(q1)
```

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30]
```

- **Quiz 2 : 크기 5의 랜덤 배열을 만들고 최대값을 0으로 바꾸기**

```
q2 = np.random.random(5)
print(q2)
q2[q2.argmax()] = 0
print(q2)
```

```
[0.74477615 0.29900594 0.39137998 0.91495816 0.35923376]
[0.74477615 0.29900594 0.39137998 0.         0.35923376]
```

6. Summary

Summary

- **Quiz 3 : 3x3 랜덤 배열을 만들고 값이 동일하지 않은 행 추출하기 (ex. [2 2 3])**

```
q3 = np.random.randint(0,5,(3,3))
print(q3)
# solution 1
E = np.all(q3[:,1:] == q3[:,:-1], axis =1)
print(E)
U = q3[~E]
print(U)
# solution 2
U = q3[q3.max(axis=1) != q3.min(axis=1), :]
print(U)
```

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

Reference

- https://www.youtube.com/watch?v=mirZPrWwvao
- https://jimmy-ai.tistory.com/90
- https://appia.tistory.com/184