

Numpy 기초

Auto Mobile Robot

Exported on 05/31/2024

Table of Contents

1	Numpy	6
1.1	import	6
1.2	ones	6
1.3	zeros	6
1.4	eye 단위행렬	6
1.5	numpy array.....	7
1.6	numpy shape.....	7
1.7	dtype	7
1.8	astype : 데이터 타입 변환	7
1.9	numpy array 곱 연산 element-wise.....	8
1.10	numpy array 덧셈 연산 element-wise	8
1.11	numpy array 나눗셈 연산 element-wise	8
1.12	numpy array 역수 연산 element-wise	8
1.13	numpy array 제곱승 연산 element-wise	9
1.14	numpy array 제곱승 연산 element-wise	9
1.15	numpy arange	9
1.16	index로 조회.....	9
1.17	numpy array slicing.....	9
1.18	numpy array 데이터 변환.....	10
1.19	다시 data1	10
1.20	2번 행 조회	10
1.21	2번 행 조회.....	10
1.22	범위 지정	11
1.23	이번에는 행과 열을 지정.....	11
1.24	다시 test 용 데이터.....	11
1.25	numpy array에 조건문	12
1.26	numpy array의 조건문 결과를 검색에 사용	12

1.27 numpy array의 조건문 결과를 슬라이싱에 이용	12
1.28 not 조건문	12
1.29 or 연산	13
1.30 2차원 numpy array에 조건문	13
1.31 2차원 numpy array에 조건문에 따라 값 변환	13
1.32 test용 데이터	14
1.33 shape 확인	14
1.34 reshape	14
1.35 다시 shape 확인	14
1.36 Transpose.....	15
1.37 행렬 곱 np.dot.....	15
1.38 놀래지 말자 수학~.....	15
1.39 matplotlib import	15
1.40 numpy arange	16
1.41 meshgrid.....	16
1.42 meshgrid 확인 xs.....	16
1.43 meshgrid 확인 ys.....	16
1.44 meshgrid 확인 shaep	17
1.45 meshgrid를 이용한 적용	17
1.46 그리자~	18
1.47 색상 변화	19
1.48 colorbar 적용	20
1.49 또 테스트용 데이터.....	20
1.50 where 함수	20
1.51 랜덤으로 테스트 데이터.....	21
1.52 where 함수 적용	21
1.53 where 함수 적용	21
2 간단한 통계.....	22
2.1 평균	22

2.2	각 축별 axis 지정해서 평균, 0이면 세로, 1이면 가로방향	22
2.3	표준편차와 분산	23
2.4	최대 최소	23
2.5	최대 최소의 index 찾기	23
3	집합	24
3.1	안녕 밥 조 월~	24
3.2	unique	24
3.3	집합 학습용 데이터	24
3.4	A데이터에 B가 포함되었는지 확인하는 조건	25
3.5	A데이터에 B가 포함되었는지 확인하는 조건을 사용해서 데이터 찾기	25
3.6	합집합	25
3.7	교집합	26
3.8	차집합	27
3.9	차집합	28
3.10	xor 집합	29
4	문자 연산	30
4.1	우리는 1/2이지만 코드는 0.5인데	30
4.2	이렇게 표현하게 할 수 있다	30
4.3	이렇게도 가능	30
4.4	이것은 무한대~	31
4.5	무한대의 성질 확인	31
4.6	이번에는 이런 문자가 포함된 식	31
4.7	이런걸 symbolic 연산이라고 함	31
4.8	이번에는 이 수식	32
4.9	약분도 해줌	32
4.10	극한 문제	32
4.11	이것도 풀어줌	32
4.12	삼각함수	33
4.13	삼각함수의 미분	33

4.14 이번에는 삼차식	33
4.15 적분도 가능.....	33

1 Numpy

1.1 import

```
1 import numpy as np
```

1.2 ones

```
1 np.ones((3,2))
```

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

1.3 zeros

```
1 np.zeros((3,2))
```

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

1.4 eye 단위행렬

```
1 np.eye(2)
```

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

1.5 numpy array

```
1 data1 = np.array([ [1, 2, 3], [4, 5, 6], [7, 8, 9] ])
2
3 data1
```

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

1.6 numpy shape

```
1 data1.shape
```

```
(3, 3)
```

1.7 dtype

```
1 data1.dtype
```

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

1.8 astype: 데이터 타입 변환

```
1 data1 = data1.astype(np.float64)
2 data1
```

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

1.9 numpy array 곱 연산 element-wise

```
1 data1 * data1
```

```
array([[ 1.,  4.,  9.],
       [16., 25., 36.],
       [49., 64., 81.]])
```

1.10 numpy array 덧셈 연산 element-wise

```
1 data1 + 3
```

```
array([[ 4.,  5.,  6.],
       [ 7.,  8.,  9.],
       [10., 11., 12.]])
```

1.11 numpy array 나눗셈 연산 element-wise

```
1 data1 / 3
```

```
array([[0.33333333, 0.66666667, 1.          ],
       [1.33333333, 1.66666667, 2.          ],
       [2.33333333, 2.66666667, 3.          ]])
```

1.12 numpy array 역수 연산 element-wise

```
1 1 / data1
```

```
array([[1.          , 0.5          , 0.33333333],
       [0.25         , 0.2          , 0.16666667],
       [0.14285714, 0.125         , 0.11111111]])
```


1.13 numpy array 제곱승 연산 element-wise

```
1 data1**0.5
```

```
array([[1.          , 1.41421356, 1.73205081],
       [2.          , 2.23606798, 2.44948974],
       [2.64575131, 2.82842712, 3.          ]])
```

1.14 numpy array 제곱승 연산 element-wise

```
1 data1**2
```

```
array([[ 1.,  4.,  9.],
       [16., 25., 36.],
       [49., 64., 81.]])
```

1.15 numpy arange

```
1 data2 = np.arange(10)
2 data2
```

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

1.16 index로 조회

```
1 data2[0]
```

```
0
```

1.17 numpy array slicing

```
1 data2[1:4]
```

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

1.18 numpy array 데이터 변환

```
1 data2[1:4] = 10
2 data2
```

```
array([ 0, 10, 10, 10,  4,  5,  6,  7,  8,  9])
```

1.19 다시 data1


```
2 data1
```

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

1.20 2번 행 조회

```
1 data1[2]
```

```
array([7., 8., 9.])
```

 코드

 테스트

1.21 2번 행 조회

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

1.22 범위 지정

```
1 data1[:2]
```

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

1.23 이번에는 행과 열을 지정

```
1 data1[:2, 1:]
```

```
array([[2., 3.],
       [5., 6.]])
```

1.24 다시 test 용 데이터

```
1 names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will'])
2 names
```

```
array(['Bob', 'Joe', 'Will', 'Bob', 'Will'], dtype='<U4')
```

```
1 data_names = np.random.randn(5,4)
2 data_names
```

```
array([[ 0.03612705, -1.05848604,  0.56566209,  0.35947677],
       [-0.78369421, -0.57542503, -0.72599282, -0.6789956 ],
       [-2.62919556,  0.5663384 ,  1.78059729,  2.2488588 ],
       [ 0.97463277, -0.39598514,  2.67363052, -0.61094586],
       [ 2.46023261, -0.42160199, -1.01264017,  0.79056415]])
```

1.25 numpy array에 조건문

```
1 names == 'Bob'
```

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

1.26 numpy array의 조건문 결과를 검색에 사용

```
1 data_names[names == 'Bob']
```

```
array([[ 0.03612705, -1.05848604,  0.56566209,  0.35947677],
       [ 0.97463277, -0.39598514,  2.67363052, -0.61094586]])
```

1.27 numpy array의 조건문 결과를 슬라이싱에 이용

```
1 data_names[names == 'Bob', 2:]
```

```
array([[ 0.56566209,  0.35947677],
       [ 2.67363052, -0.61094586]])
```

1.28 not 조건문

```
1 data_names[~(names == 'Bob')]
```

```
array([[ -0.78369421, -0.57542503, -0.72599282, -0.6789956 ],
       [ -2.62919556,  0.5663384 ,  1.78059729,  2.2488588 ],
       [  2.46023261, -0.42160199, -1.01264017,  0.79056415]])
```

1.29 or 연산

```
1 mask = (names == 'Bob') | (names == 'Will')
2 mask
```

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

```
1 data_names[mask]
```

```
array([[ 0.03612705, -1.05848604,  0.56566209,  0.35947677],
       [-2.62919556,  0.5663384 ,  1.78059729,  2.2488588 ],
       [ 0.97463277, -0.39598514,  2.67363052, -0.61094586],
       [ 2.46023261, -0.42160199, -1.01264017,  0.79056415]])
```

1.30 2차원 numpy array에 조건문

```
1 data_names < 0
```

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

1.31 2차원 numpy array에 조건문에 따라 값 변환

```
1 data_names[data_names < 0] = 0
2 data_names
```

```
array([[0.03612705, 0.          , 0.56566209, 0.35947677],
       [0.          , 0.          , 0.          , 0.          ],
       [0.          , 0.5663384 , 1.78059729, 2.2488588 ],
       [0.97463277, 0.          , 2.67363052, 0.          ],
       [2.46023261, 0.          , 0.          , 0.79056415]])
```

1.32 test용 데이터

```
1 data_mat = np.arange(15)
2 data_mat
```

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

1.33 shape 확인

```
1 data_mat.shape
```

```
(15,)
```

1.34 reshape

```
1 data_mat = data_mat.reshape((3,5))
2 data_mat
```

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

1.35 다시 shape 확인

```
[ ] 1 data_mat.shape
```

```
↳ (3, 5)
```



1.36 Transpose

```
[ ] 1 data_mat.T
```

```
↳ array([[ 0,  5, 10],
          [ 1,  6, 11],
          [ 2,  7, 12],
          [ 3,  8, 13],
          [ 4,  9, 14]])
```

1.37 행렬 곱 np.dot

```
1 np.dot(data_mat, data_mat.T)
```

```
array([[ 30,  80, 130],
       [ 80, 255, 430],
       [130, 430, 730]])
```

1.38 놀래지 말자 수학~

$$z = \sqrt{x^2 + y^2}$$

1.39 matplotlib import

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
```

1.40 numpy arange

```
1 points = np.arange(-5, 5, 0.01)
```

1.41 meshgrid

```
1 xs, ys = np.meshgrid(points, points)
```

1.42 meshgrid 확인 xs

```
1 xs
```

```
array([[ -5.    , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [ -5.    , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [ -5.    , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       ...,
       [ -5.    , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [ -5.    , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [ -5.    , -4.99, -4.98, ...,  4.97,  4.98,  4.99]])
```

1.43 meshgrid 확인 ys

```
1 ys
```

```
array([[ -5.    , -5.    , -5.    , ..., -5.    , -5.    , -5.    ],
       [-4.99, -4.99, -4.99, ..., -4.99, -4.99, -4.99],
       [-4.98, -4.98, -4.98, ..., -4.98, -4.98, -4.98],
       ...,
       [ 4.97,  4.97,  4.97, ...,  4.97,  4.97,  4.97],
       [ 4.98,  4.98,  4.98, ...,  4.98,  4.98,  4.98],
       [ 4.99,  4.99,  4.99, ...,  4.99,  4.99,  4.99]])
```


1.44 meshgrid 확인 shaep

```
1 xs.shape, ys.shape
```

```
((1000, 1000), (1000, 1000))
```

1.45 meshgrid를 이용한 적용

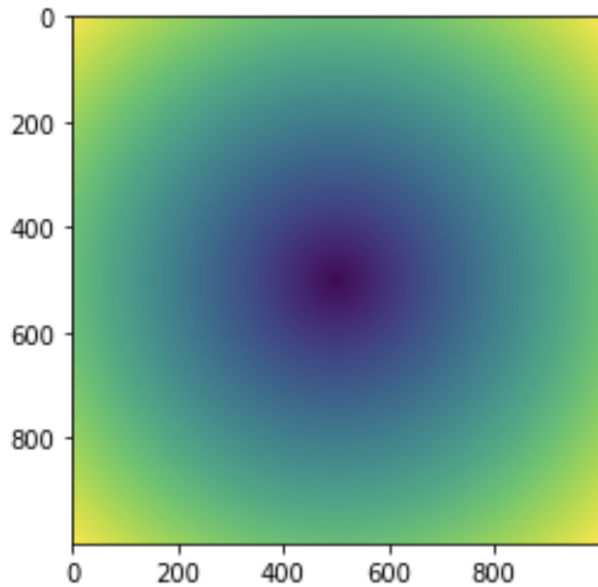
```
1 z = np.sqrt(xs**2 + ys**2)
2 z
```

```
array([[7.07106781, 7.06400028, 7.05693985, ..., 7.04988652, 7.05693985,
        7.06400028],
       [7.06400028, 7.05692568, 7.04985815, ..., 7.04279774, 7.04985815,
        7.05692568],
       [7.05693985, 7.04985815, 7.04278354, ..., 7.03571603, 7.04278354,
        7.04985815],
       ...,
       [7.04988652, 7.04279774, 7.03571603, ..., 7.0286414 , 7.03571603,
        7.04279774],
       [7.05693985, 7.04985815, 7.04278354, ..., 7.03571603, 7.04278354,
        7.04985815],
       [7.06400028, 7.05692568, 7.04985815, ..., 7.04279774, 7.04985815,
        7.05692568]])
```

1.46 그리자~

```
1 plt.imshow(z)
```

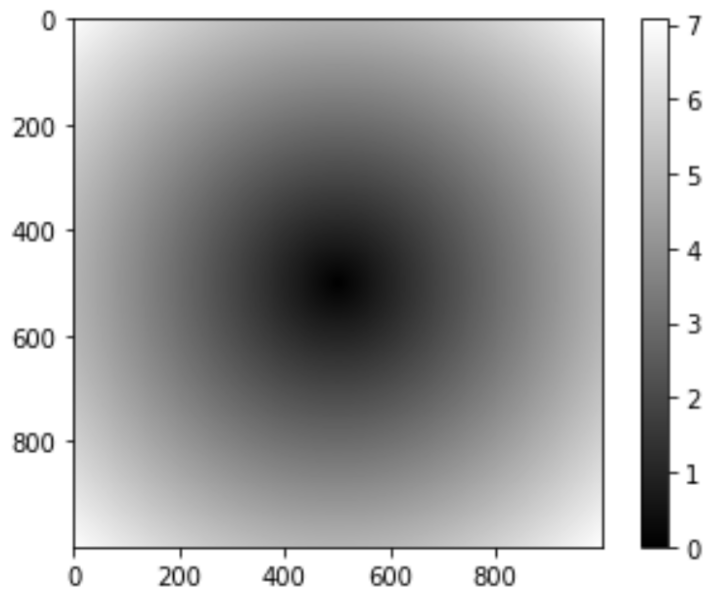
<matplotlib.image.AxesImage at 0x7f115f63b8d0>



1.47 색상 변화

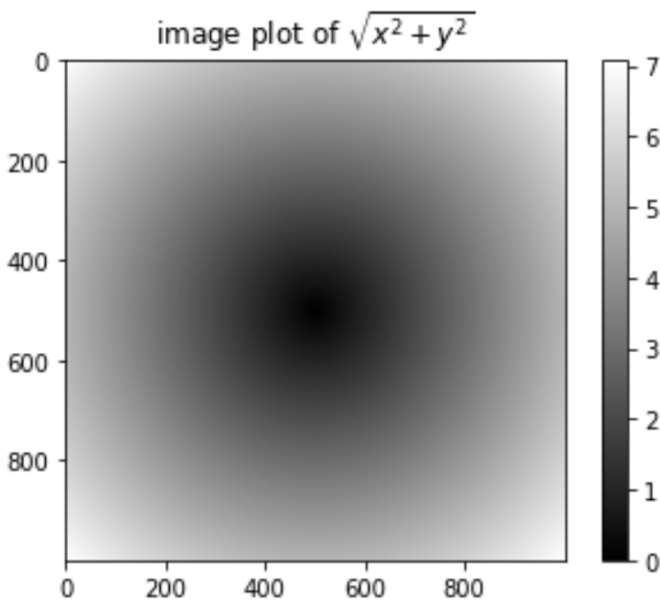
```
1 plt.imshow(z, cmap=plt.cm.gray)
2 plt.colorbar()
```

<matplotlib.colorbar.Colorbar at 0x7f115f1420f0>



1.48 colorbar 적용

```
1 plt.imshow(z, cmap=plt.cm.gray)
2 plt.colorbar()
3 plt.title('image plot of  $\sqrt{x^2 + y^2}$ ')
4 plt.show()
```



1.49 또 테스트용 데이터

```
1 x = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
2 y = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
3
4 condi = np.array([True, False, True, True, False])
```

1.50 where 함수

```
1 np.where(condi, x, y)
```

```
array([1.1, 2.2, 1.3, 1.4, 2.5])
```

1.51 랜덤으로 테스트 데이터

```
1 test = np.random.randn(4,4)
2 test
```

```
array([[ 1.32038108, -0.34381875, -0.39094012, -0.9902911 ],
       [ 1.63267262,  0.04760321,  0.78570584, -2.06811086],
       [-0.14984864,  0.87803223, -1.09720975, -1.12187609],
       [-0.5580423 , -0.15057052,  1.36309171, -2.35563164]])
```

1.52 where 함수 적용

```
[ ] 1 np.where(test > 0, 2, -1)
```

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

1.53 where 함수 적용

```
1 np.where(test > 0, 2, test)
```

```
array([[ 2.          , -0.34381875, -0.39094012, -0.9902911 ],
       [ 2.          ,  2.          ,  2.          , -2.06811086],
       [-0.14984864,  2.          , -1.09720975, -1.12187609],
       [-0.5580423 , -0.15057052,  2.          , -2.35563164]])
```

2 간단한 통계

2.1 평균

```
1 np.mean(test)
```

```
-0.1999283181097743
```

```
1 test.mean()
```

```
-0.1999283181097743
```

```
1 test.sum()
```

```
-3.1988530897563887
```

2.2 각 축별 axis 지정해서 평균, 0이면 세로, 1이면 가로방향

```
1 test.mean(axis=0)
```

```
array([ 0.56129069,  0.10781154,  0.16516192, -1.63397742])
```

```
1 test.mean(axis=1)
```

```
array([-0.10116722,  0.0994677 , -0.37272556, -0.42528819])
```

2.3 표준편차와 분산

```
1 test.std()
```

```
1.1433192232301272
```

```
1 test.var()
```

```
1.3071788462075415
```

2.4 최대 최소

```
1 [test.min(), test.max()]
```

```
[-2.355631638905563, 1.6326726174369945]
```

2.5 최대 최소의 index 찾기

```
1 [test.argmin(), test.argmax()]
```

```
[15, 4]
```

3 집합

3.1 안녕 밥 조 윌~

```
1 names
```

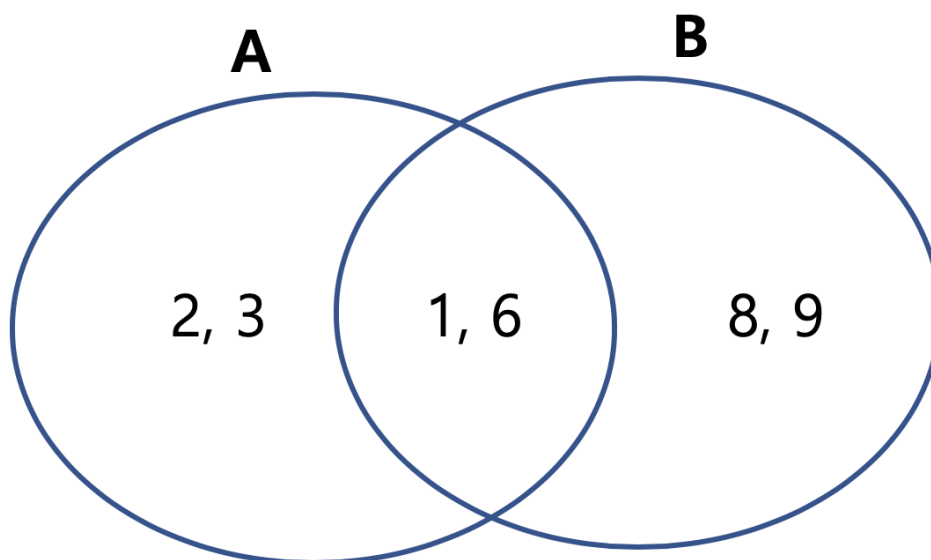
```
array(['Bob', 'Joe', 'Will', 'Bob', 'Will'], dtype='<U4')
```

3.2 unique

```
1 np.unique(names)
```

```
array(['Bob', 'Joe', 'Will'], dtype='<U4')
```

3.3 집합 학습용 데이터




```
1 A = np.array([1, 2, 3, 6])
2 B = np.array([1, 6, 8, 9])
```

3.4 A데이터에 B가 포함되었는지 확인하는 조건

```
1 np.in1d(A, B)
```

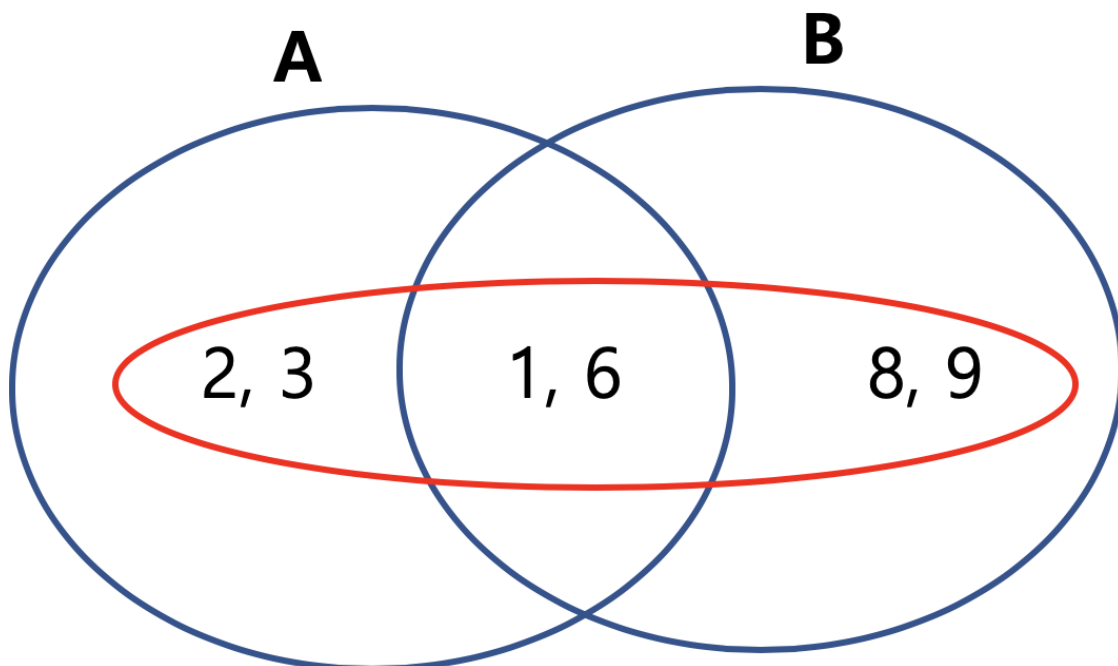
```
array([ True, False, False,  True])
```

3.5 A데이터에 B가 포함되었는지 확인하는 조건을 사용해서 데이터 찾기

```
1 A[np.in1d(A, B)]
```

```
array([1, 6])
```

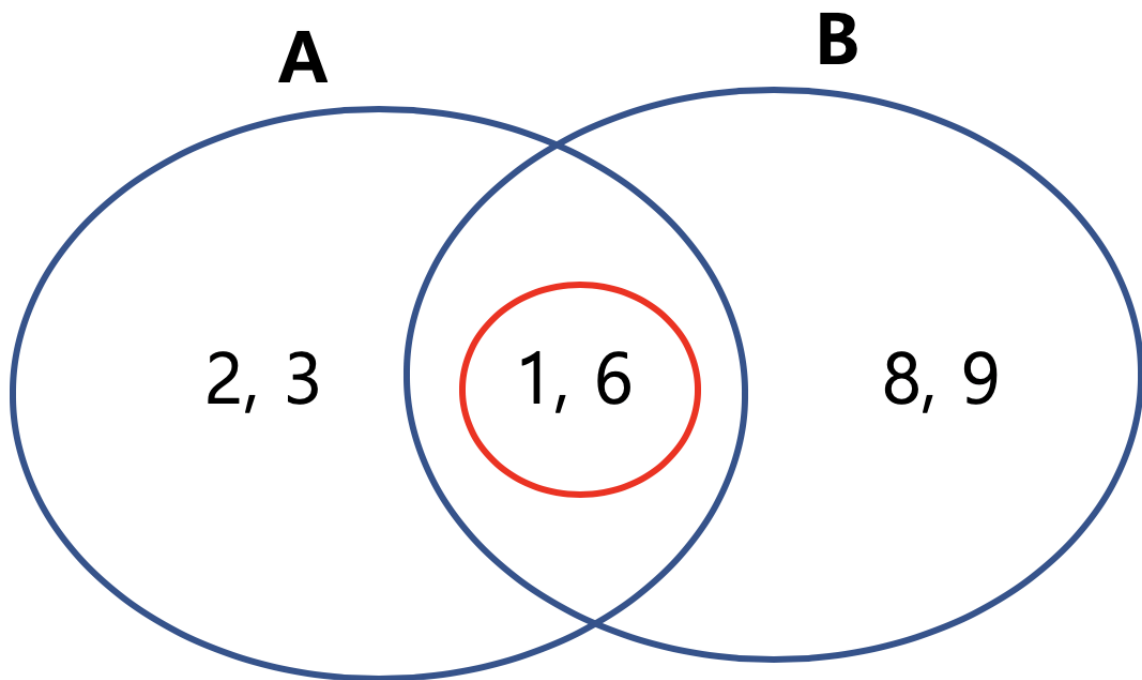
3.6 합집합



```
1 np.union1d(A, B)
```

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

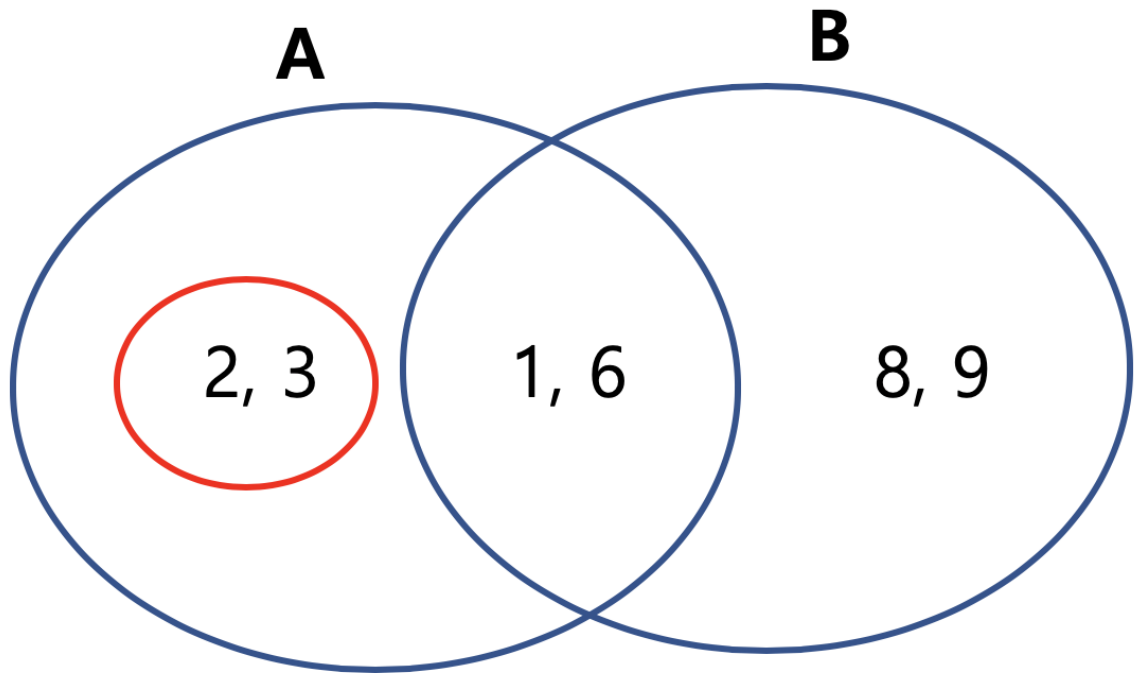
3.7 교집합



```
1 np.intersect1d(A, B)
```

```
array([1, 6])
```

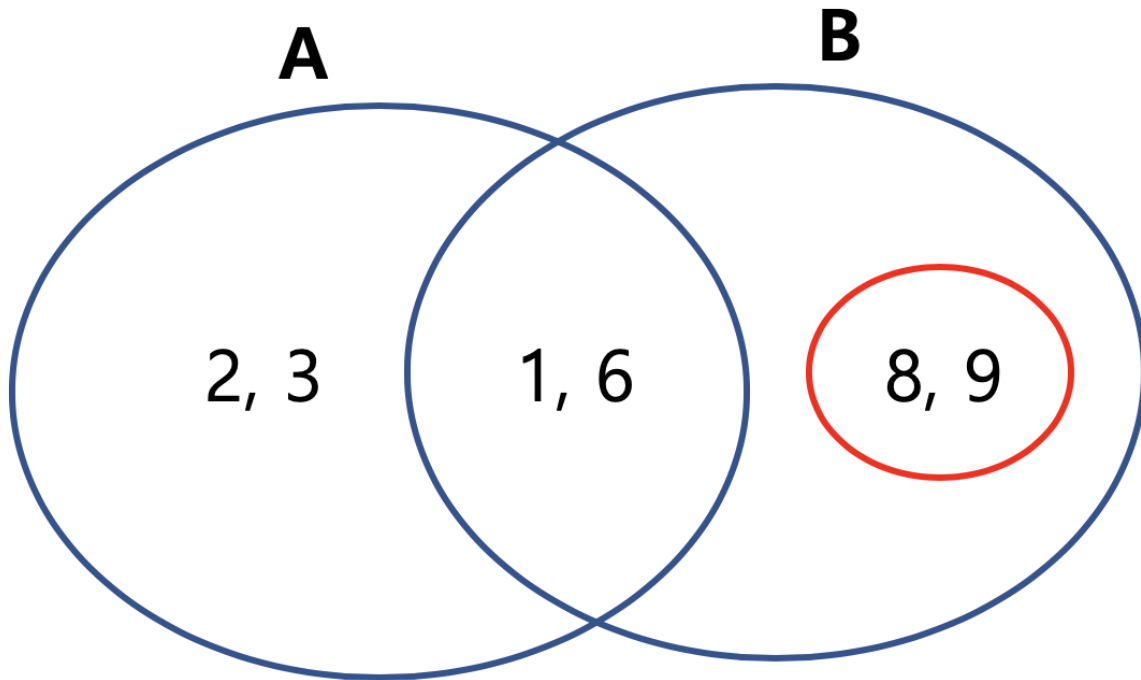
3.8 차집합



```
1 np.setdiff1d(A, B)
```

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

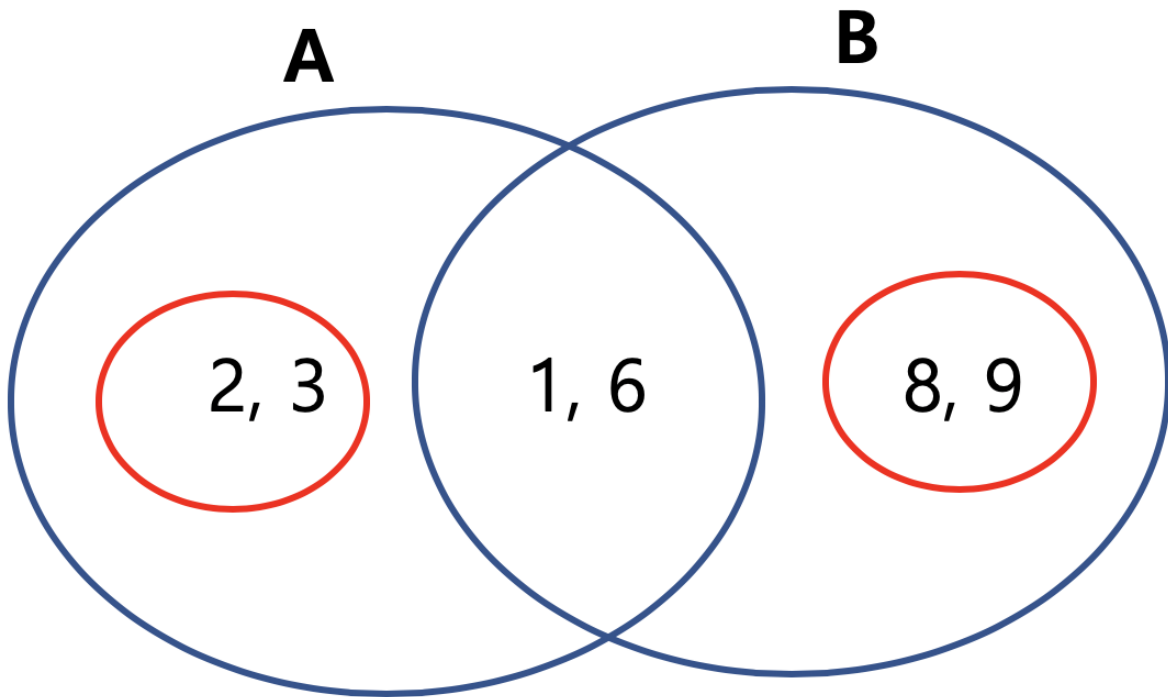
3.9 차집합



```
1 np.setdiff1d(B, A)
```

```
array([8, 9])
```

3.10 xor 집합



```
1 np.setxor1d(A, B)
```

```
array([2, 3, 8, 9])
```

4 문자 연산

4.1 우리는 1/2이지만 코드는 0.5인데

$$\frac{1}{2}$$

4.2 이렇게 표현하게 할 수 있다

```
import sympy as sym

a = sym.Rational(1, 2)
a
```

[53] ✓ 0.0s Python

... $\frac{1}{2}$

4.3 이렇게도 가능

```
a * 2
```

[54] ✓ 0.0s Python

... 1

```
a / 2
```

[55] ✓ 0.0s Python

... $\frac{1}{4}$

4.4 이것은 무한대~

∞

4.5 무한대의 성질 확인

```
print(sym.oo > 9999)

print(sym.oo + 1)
```

[56] ✓ 0.0s Python

... True
oo

4.6 이번에는 이런 문자가 포함된 식

$$(x+y)^3$$

4.7 이런걸 symbolic 연산이라고 함

```
x = sym.Symbol('x')
y = sym.Symbol('y')

sym.expand((x + y) ** 3)
```

[52] ✓ 0.0s Python

... $x^3 + 3x^2y + 3xy^2 + y^3$

4.8 이번에는 이 수식

$$\frac{x^2 - y^2}{x + y}$$

4.9 약분도 해줌

```
[57] sym.simplify((x**2 - y**2) / (x + y))
```

✓ 0.1s Python

... $x - y$

4.10 극한 문제

$$\lim_{x \rightarrow \infty} \frac{2x + 10}{x + 1}$$

4.11 이것도 풀어줌

```
[58] sym.limit((2 * x + 10) / (x + 1), x, sym.oo)
```

✓ 0.0s Python

... 2

4.12 삼각함수

$$\sin 2x$$

4.13 삼각함수의 미분

```
[59] sym.diff(sym.sin(2 * x), x)
✓ 0.0s Python
... 2 cos(2x)
```

4.14 이번에는 삼차식

$$3x^2$$

4.15 적분도 가능

```
[60] sym.integrate(3 * x ** 2, x)
✓ 0.0s Python
... x^3
```