Elixirr

파이썬 기본 3

인선미

2026.01

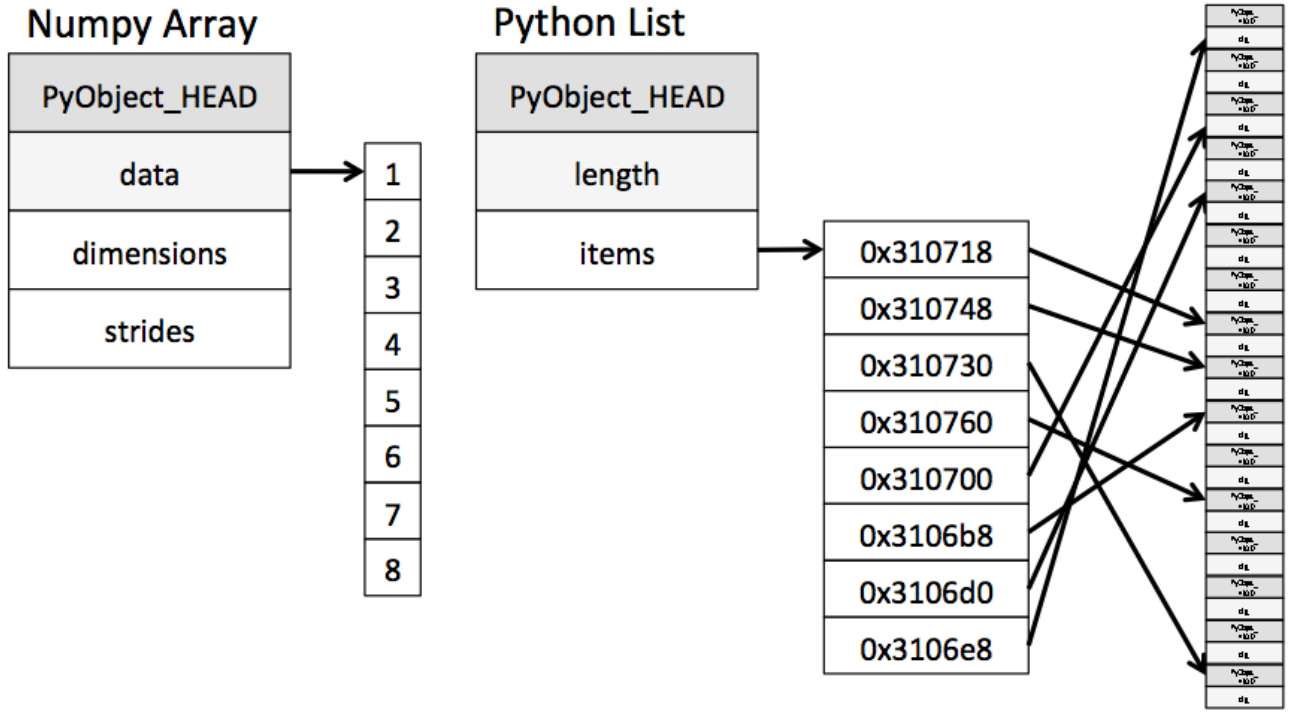
NumPy

넘파이(Numpy)

- ✓ 대수, 행렬, 통계 등 수학 및 과학 연산을 위한 라이브러리
- ✓ ndarray라는 다차원 배열을 데이터로 나타내고 처리하는데 특화
- ✓ **실행속도가 빠르고 짧고 간결한 코드 구현이** 가능하다.
- ✓ Numpy는 외부 라이브러리이므로, 설치 후 사용할 수 있다.

NumPy 배열과 파이썬 리스트의 차이

- ✓ 파이썬 리스트는 배열에 실제 데이터의 주소가 담겨져 있음
- ✓ 이것으로 인해 서로 다른 데이터타입을 리스트 요소로 넣을 수 있음
- ✓ 반면 numpy 배열은 실제 데이터가 배열에 들어 있음
- ✓ 리스트는 데이터를 읽고 쓰기에 numpy 배열보다 2배의 시간이 걸림
- ✓ 대량의 배열 연산 또는 행렬 연산에는 numpy가 유리함



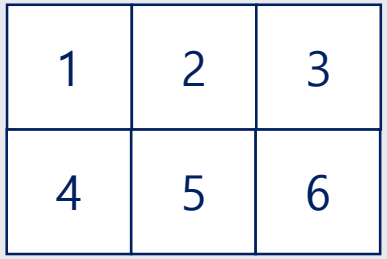
넘파이 배열(Numpy Array)

- ✓ 넘파이는 다차원 배열을 지원한다.
- ✓ 넘파이 배열은 한 가지 자료형 데이터를 요소로 갖는다.

1차원

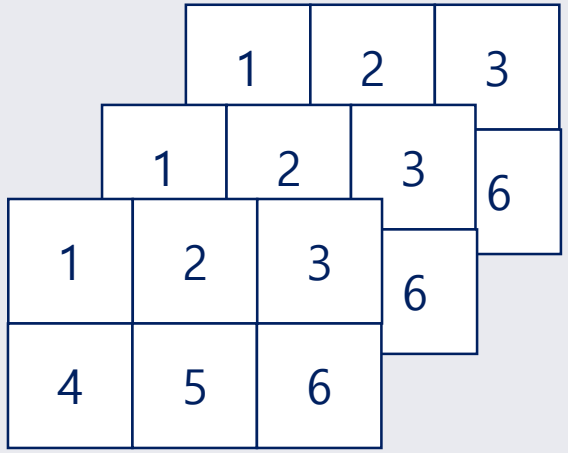
Vector

2차원



Matrix

3차원



Tensor(3차원 이상)

넘파이 배열 직접 생성

리스트 데이터를 넘파이 배열로 바꾸어 생성

np.array(리스트)

```
num_list = [1, 2, 3, 4]
int_arr2 = np.array(num_list)
print(int_arr2)
```

```
[1 2 3 4]
```

넘파이 배열은 콤마(,)없이 공백으로 데이터가 구분된다.

배열의 범위를 지정하여 생성

시작 인덱스부터 시작하여 끝 인덱스 전까지, 간격만큼 더해 넘파이 배열 생성

np.arange (시작 인덱스 : 끝 인덱스 : 간격)

```
arr1 = np.arange(0, 10)

arr1
```

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

```
arr2 = np.arange(0, 10, 2)

arr2
```

```
array([0, 2, 4, 6, 8])
```

```
arr3 = np.arange(5)

arr3
```

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

넘파이 배열에 들어가는 자료형

✓ 한 가지 데이터 타입으로 통일되어 처리된다.

```
list_data = [1, '2', 3]

arr = np.array(list_data)

print(arr)
```

```
['1' '2' '3']
```

문자열형으로 통일

```
list_data = [1, 2., 3]

arr = np.array(list_data)

print(arr)
```

```
[1. 2. 3.]
```

실수형으로 통일

2차원 배열 생성

- ✓ 2차원 배열은 행렬(matrix)이라고 한다.
- ✓ 가로줄은 행 (row), 세로줄은 열(column)

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

print(arr)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

중첩된 2차원 리스트

3차원 배열 생성

✓ 3차원 이상의 배열은 tensor라고 한다.

```
arr = np.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]]])

print(arr)
```

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

 [[ 9 10 11 12]
  [13 14 15 16]]

 [[17 18 19 20]
  [21 22 23 24]]]
```

배열의 차원과 크기 알아보기

넘파이 배열명.ndim

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

print(arr)
print(arr.ndim)
```

```
[1 2 3 4]
1
```

1차원

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

print(arr)
print(arr.ndim)
```

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

2차원

넘파이 배열명.shape

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

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

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

1차원
요소수가 4개

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

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

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

2차원
2행 4열

Elixirr

배열의 차원 변경

넘파이 배열명.reshape(크기)

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

print(arr.ndim)
print(arr.shape)

arr2 = arr.reshape(2, 5)

print(arr2.ndim)
print(arr2.shape)
```

```
1
(10,)
2
(2, 5)
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

arr2 = arr.reshape(2, -1)

print(arr2.ndim)
print(arr2.shape)
```

인수 중 하나에 -1을 주면 알아서 배열을 만들어준다

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

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

arr2 = arr.reshape(5, -1)

print(arr2.ndim)
print(arr2.shape)
```

인수 중 하나에 -1을 주면 알아서 차원을 만들어준다

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

넘파이 연산

✓ shape이 같다면 덧셈, 뺄셈, 곱셈, 나눗셈의 연산을 할 수 있다.

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

print(a + b)
print(a - b)
print(a * b)
print(a / b)
```

```
[5 7 9]
[-3 -3 -3]
[ 4 10 18]
[0.25 0.4  0.5 ]
```

동일한 인덱스 끼리 연산

넘파이 연산

✓ 백터화 연산

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

print(a + 1)
print(a - 1)
print(a * 3)
print(a / 2)
```

```
[2 3 4]
[0 1 2]
[3 6 9]
[0.5 1.  1.5]
```

모든 요소에 일괄적으로 연산 적용

※파이썬의 리스트: *연산만 가능. 리스트의 요소를 반복

배열의 범위와 개수를 지정하여 생성

시작인덱스 부터 끝 인덱스까지 n개의 요소를 갖는 넘파이 배열을 생성

np.linspace(시작 인덱스 : 끝 인덱스 : 개수n)

```
arr = np.linspace(1, 10, 10)

print(arr)
print(arr.dtype)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
float64
```

1부터 10까지 10개의 데이터 생성

```
arr = np.linspace(0, np.pi, 10)

print(arr)
print(arr.dtype)
```

```
[0.         0.34906585 0.6981317  1.04719755 1.3962634  1.74532925
 2.0943951  2.44346095 2.7925268  3.14159265]
float64
```

1부터 π까지 동일한 간격으로 10개의 데이터 생성

특별한 값을 갖는 배열 생성

np.zeros()

```
arr = np.zeros(10)

print(arr)   # 실수로 만들어준다
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
arr = np.zeros(10, dtype = int)

print(arr)
```

```
[0 0 0 0 0 0 0 0 0 0]
```

```
arr = np.zeros((2, 5))

print(arr)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

np.ones()

```
arr = np.ones(10)

print(arr)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
arr = np.ones(10, dtype = int)

print(arr)
```

```
[1 1 1 1 1 1 1 1 1 1]
```

```
arr = np.ones((2, 5))

print(arr)
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

np.eye()

```
arr = np.eye(5)

print(arr)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

```
arr = np.eye(5, dtype = int)

print(arr)
```

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

NumPy 배열 연산하기

✓ **벡터화 연산(vectorized operation)을 지원**

✓ 배열의 각 원소에 대한 반복 연산을 하나의 명령어로 처리

✓ 데이터를 모두 2배로 변경

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

✓ **반복문으로 구현해 주세요!**

NumPy 배열 연산하기

✓ **벡터화 연산(vectorized operation)을 지원**

✓ 배열의 각 원소에 대한 반복 연산을 하나의 명령어로 처리

```
1   x = np.array(data)
2   2 * x
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

도전하기

```
a = np.array([1, 2, 3])
b = np.array([10, 20, 30])
```

일 때, 다음의 결과를 예측해 주세요

1. `2 * a + b`
2. `a == 2`
3. `b > 10`
4. `(a == 2) & (b > 10)`

NumPy 배열

✓ 2차원 배열 만들고 행과 열의 개수 구하기

✓ 다음 배열의 크기는?

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

✓ 기존의 방법으로 행의 개수? 열의 개수?

NumPy 배열

✓ 3차원 배열 만들고 행과 열의 개수 구하기

✓ 다음 배열의 크기는?

```
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]]])
d
```

✓ 기존의 방법으로 배열의 깊이? 행의 개수? 열의 개수?

NumPy 배열

✓ 배열의 차원과 크기 알아내기

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

NumPy 배열

✓ 배열의 차원과 크기 알아내기

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

NumPy 배열

✓ 배열의 차원과 크기 알아내기

```
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]]])
print(d.ndim)
print(d.shape)
print(d.size)
```

NumPy 배열

✓ 배열의 인덱싱

✓ 다음의 결과는?

```
a = np.array([0, 1, 2, 3, 4])
print(a[2])
print(a[-1])
```

NumPy 배열

✓ 배열의 인덱싱

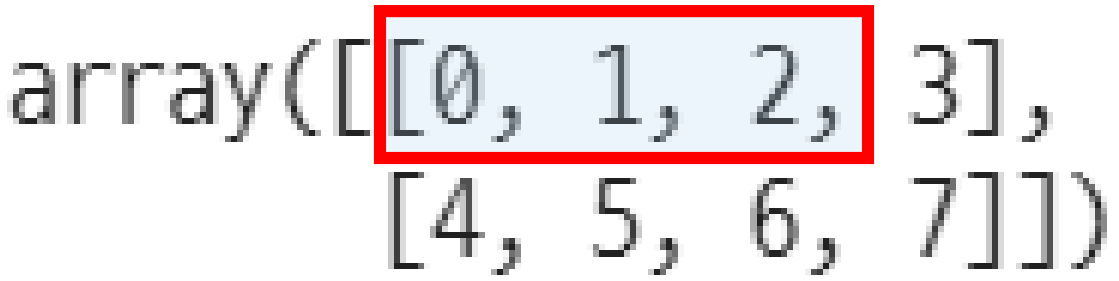
✓ 다음의 배열에서 0, 1, 5 값을 뽑아내 주세요

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

✓ 배열의 슬라이싱

✓ 다음의 배열에서 아래 부분을 뽑아내 주세요

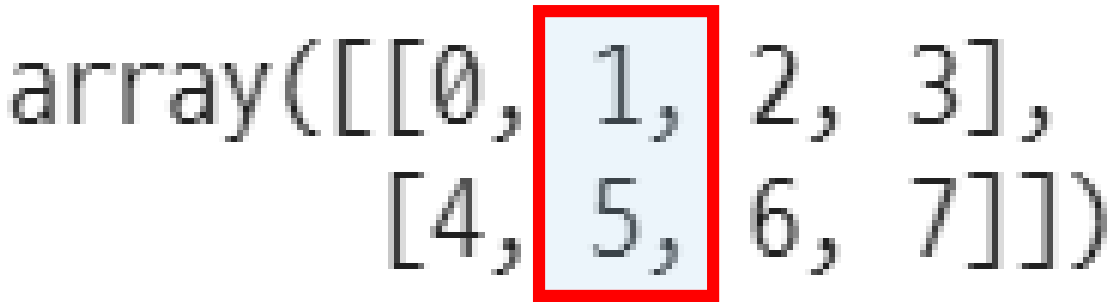
```
a = np.array([[0, 1, 2, 3], [4, 5, 6, 7]])
a
```



✓ 배열의 슬라이싱

✓ 다음의 배열에서 아래 부분을 뽑아내 주세요

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



✓ 배열의 슬라이싱

✓ 다음의 배열에서 아래 부분을 뽑아내 주세요

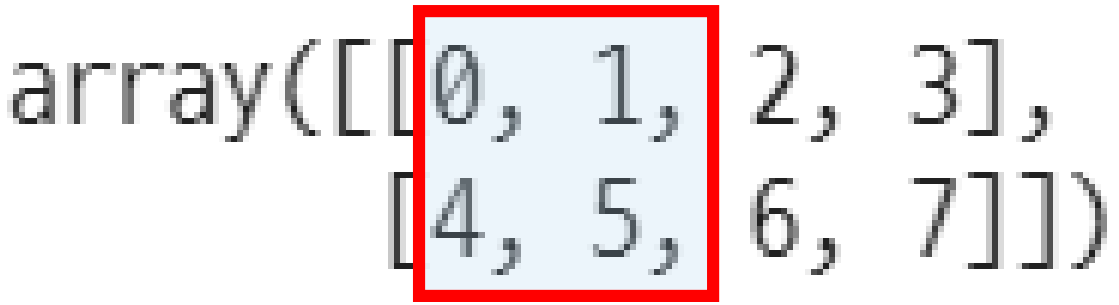
```
a = np.array([[0, 1, 2, 3], [4, 5, 6, 7]])
a
```

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

✓ 배열의 슬라이싱

✓ 다음의 배열에서 아래 부분을 뽑아내 주세요

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



NumPy 배열

- ✓ **팬시 인덱싱(fancy indexing)**
- ✓ 데이터베이스의 질의(Query) 기능을 수행
- ✓ 대괄호(Bracket, [])안의 인덱스 정보로 숫자나 슬라이스가 아니라 위치 정보를 나타내는 또 다른 ndarray 배열을 받을 수 있다.
- ✓ 불리언(Boolean) 배열 방식과 정수 배열 방식

NumPy 배열

- ✓ **팬시 인덱싱(fancy indexing)**
- ✓ 불리언 방식
- ✓ **다음의 결과는?**

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

NumPy 배열

✓ 팬시 인덱싱(fancy indexing)

✓ 조건문 연산을 사용하여 짝수인 원소만 골라내기

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

NumPy 배열

- ✓ **팬시 인덱싱(fancy indexing)**
- ✓ 정수 배열 인덱싱 방식
- ✓ **다음의 결과는?**

```
a = np.array([11, 22, 33, 44, 55, 66, 77, 88, 99])
idx = np.array([0, 2, 4, 6, 8])
a[idx]
```

NumPy 배열

- ✓ **팬시 인덱싱(fancy indexing)**
- ✓ 정수 배열 인덱싱 방식
- ✓ **다음의 결과는?**

```
a = np.array([11, 22, 33, 44, 55, 66, 77, 88, 99])
idx = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2])
a[idx]
```

NumPy 배열

✓ 팬시 인덱싱(fancy indexing)

✓ 다음의 배열에서 아래 부분을 뽑아내 주세요

```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
a
```

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

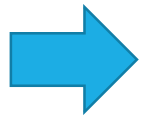
NumPy 배열

✓ 팬시 인덱싱(fancy indexing)

✓ 다음의 배열에서 아래 부분을 뽑아내 주세요

```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
a
```

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



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

도전하기

다음의 문제를 해결해 주세요

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

1. 배열에서 3의 배수를 찾아주세요.

2. 배열에서 4로 나누면 1이 남는 수를 찾아주세요.

3. 배열에서 3으로 나누면 나누어지고 4로 나누면 1이 남는 수를 찾아주세요.

NumPy 배열

✓ 넘파이 자료형

✓ 다음의 결과를 예측하면?

```
x = np.array([1, 2, 3.0])
x.dtype
```

NumPy 배열

✓ 넘파이 자료형

✓ 다음의 결과를 예측하면?

```
x = np.array([1, 2, 3.0])
x[0] + x[2]
```

NumPy 배열 생성

✓ 배열의 내부 데이터는 그대로 둔 채 형태만 바꿀 때

```
a = np.arange(12)
a
```

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

```
b = a.reshape(3, 4)
b
```

어떤 모양의 배열로 바뀌었을까요?

NumPy 배열 생성

✓ 배열의 내부 데이터는 그대로 둔 채 형태만 바꿀 때

```
a = np.arange(12)
a
```

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

```
a.reshape(3, -1)
```

어떤 모양의 배열로 바뀌었을까요?

NumPy 배열 생성

✓ 배열의 내부 데이터는 그대로 둔 채 형태만 바꿀 때

```
a = np.arange(12)
a
```

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

```
a.reshape(2, 2, -1)
```

어떤 모양의 배열로 바뀌었을까요?

NumPy 배열 생성

✓ 배열의 내부 데이터는 그대로 둔 채 형태만 바꿀 때

```
a = np.arange(12)
a
```

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

```
b = a.reshape(2, -1, 2)
b
```

어떤 모양의 배열로 바뀌었을까요?

NumPy 배열 생성

✓ 배열의 내부 데이터는 그대로 둔 채 형태만 바꿀 때

```
b
```

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

       [[ 6,  7],
        [ 8,  9],
        [10, 11]]])
```

```
b.flatten()
```

어떤 모양의 배열로 바뀌었을까요?

NumPy 벡터화연산

✓ 벡터화 연산

```
x = np.arange(1, 10001)
y = np.arange(10001, 20001)
```

```
z = np.zeros_like(x)
for i in range(10000):
    z[i] = x[i] + y[i]
```

```
z[:10]
```

결과는?

NumPy 벡터화연산

✓ 벡터화 연산

```
x = np.arange(1, 10001)
y = np.arange(10001, 20001)
```

```
z = x + y
```

```
z[:10]
```

결과는?

NumPy 벡터화연산

✓ 벡터화 연산

```
%%time
z = np.zeros_like(x)
for i in range(10000):
    z[i] = x[i] + y[i]
```

```
%%time
z = x + y
```

시간 비교

NumPy 벡터화연산

✓ 벡터화 연산

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

```
a == b
```

```
a >= b
```

결과는?

NumPy 벡터화연산

✓ 벡터화 연산

```
a = np.array([1, 2, 3, 4])
b = np.array([4, 2, 2, 4])
c = np.array([1, 2, 3, 4])
```

```
np.all(a == b)
```

```
np.all(a == c)
```

결과는?

NumPy 벡터화연산

✓ 벡터화 연산

```
x = np.arange(5)
x
```

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

```
x + 1
```

결과는?

✓ 유니버셜 함수

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

```
x.sum()
```

```
x.min()
```

```
x.argmin()
```

```
x.mean()
```

판다스(Pandas)

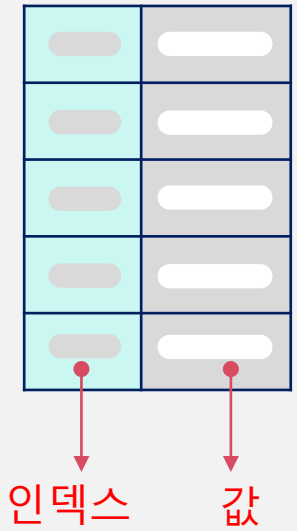
Pandas(Python Data Analysis Library)

- ✓ 데이터를 조작/분석하기 위한 라이브러리
- ✓ 다양한 데이터 분석 함수 제공
- ✓ 행과 열로 이루어진 데이터 객체를 만들고 다룰 수 있다.
- ✓ 판다스의 공식 사이트 (https://pandas.pydata.org/)

Pandas 데이터 구조

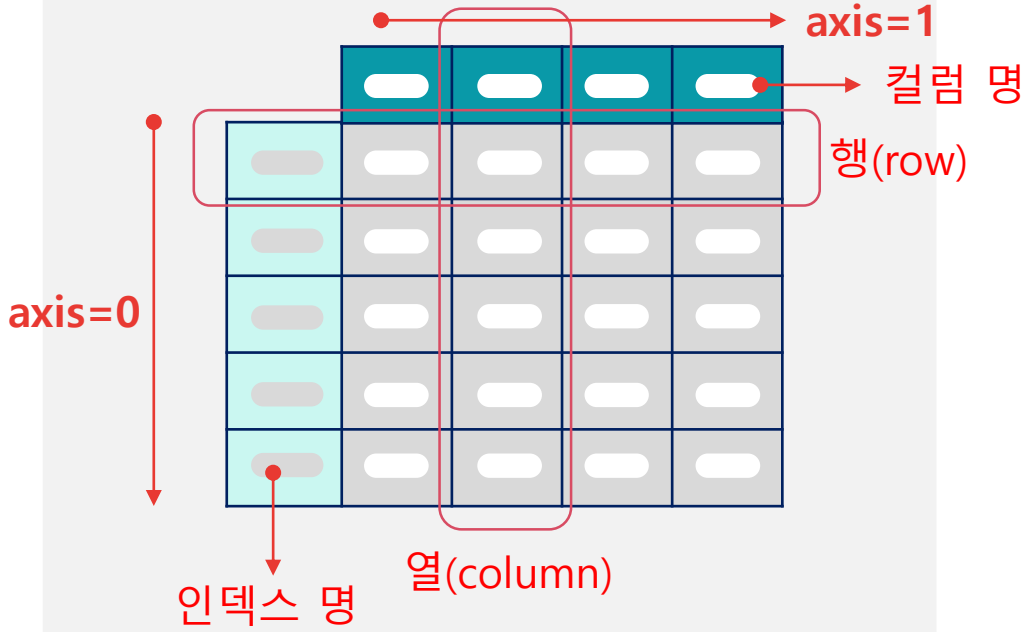
✓ 시리즈(Series)와 데이터프레임(DataFrame)이 있다.

시리즈(Series)



- 1차원 배열의 형태
- 한 가지 형의 데이터로 구성
- 인덱스는 문자로 지정 가능하고 특별히 지정하지 않으면 숫자 0부터 시작한다.

데이터 프레임(DataFrame)



- 2차원 배열의 형태(엑셀시트와 비슷)
- 인덱스와 컬럼이라는 두 가지 기준에 의하여 표 형태처럼 데이터가 저장된다.
- 시리즈가 합쳐진 형태

데이터 프레임 만들기

✓ 이중 리스트나 딕셔너리로 만들 수 있다.

✓ 딕셔너리로 데이터 프레임을 만들면 키(key)가 컬럼명으로 지정되어 만들어진다.

```
df = pd.DataFrame([['부산광역시', '051', 3364358],
                   ['전라남도','061', 1840921],
                   ['강원도', '033', 1535530]])
df
```

	0	1	2
0	부산광역시	051	3364358
1	전라남도	061	1840921
2	강원도	033	1535530

```
df = pd.DataFrame({'Division':['부산광역시', '전라남도', '강원도'],
                   'Code':['051', '061','033'],
                   'Population':[3364358, 1840921, 1535530]})
df
```

	Division	Code	Population
0	부산광역시	051	3364358
1	전라남도	061	1840921
2	강원도	033	1535530

데이터 프레임 칼럼명 지정

✓ 데이터프레임의 컬럼명 확인

```
df. columns
```

✓ 데이터프레임의 컬럼명 지정

```
df. columns = [ '컬럼1 명', '컬럼2 명'...'컬럼n 명' ]
```

데이터 프레임 인덱스 지정

✓ 데이터프레임의 인덱스 확인

```
df. index
```

✓ 인덱스를 지정하지 않으면 자동으로 0부터 시작하는 숫자 인덱스가 생성

✓ 데이터프레임의 인덱스 이름을 지정

```
df.index = ['인덱스1 명', '인덱스2 명'...'인덱스n 명']
```

데이터 프레임 인덱스 지정

✓ 데이터프레임의 특정 컬럼을 인덱스로 지정할 수 있다.

df.index = df['컬럼명']

```
df.index = df['Division']
df
```

Division	Division	Code	Population
부산광역시	부산광역시	051	3364358
전라남도	전라남도	061	1840921
강원도	강원도	033	1535530

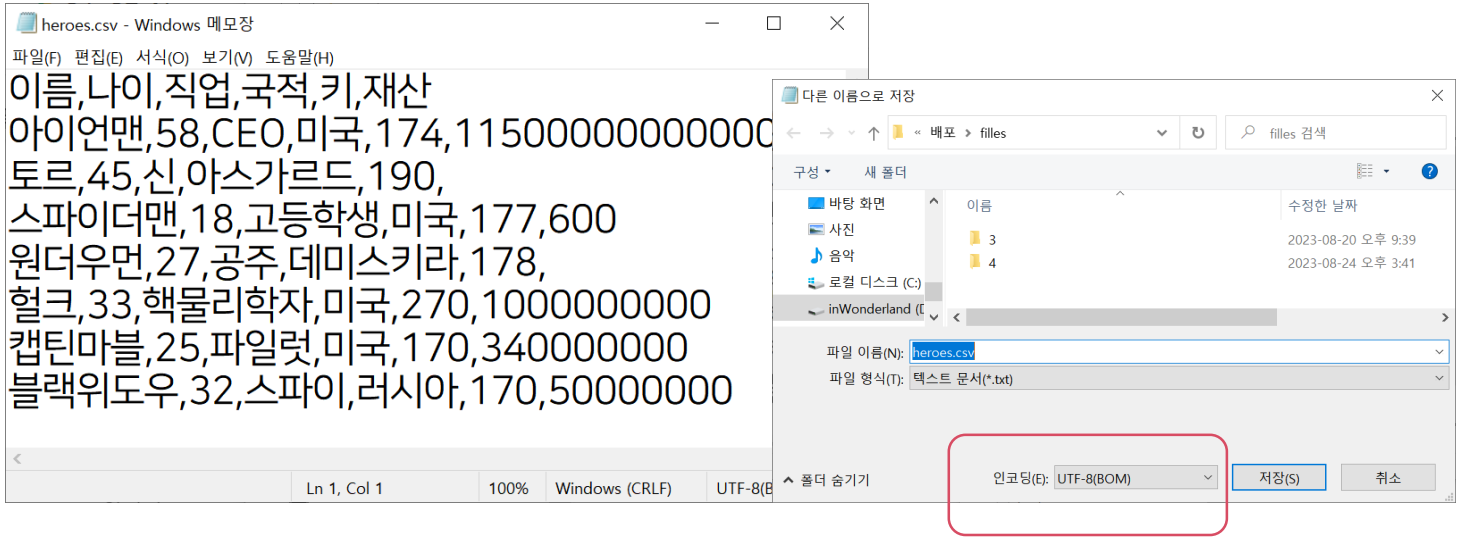
df.set_index('컬럼명')

```
new_df = df.set_index('Division')
new_df
```

Division	Code	Population
부산광역시	051	3364358
전라남도	061	1840921
강원도	033	1535530

데이터 파일 : csv

- ✓ 하나의 파일로 관리 가능한 크기의 데이터라면 일반적으로 csv포맷을 활용
- ✓ 데이터가 콤마로 분리되어 저장
- ✓ 한글이 포함된 경우 인코딩 방식을 utf8로 해야 한글이 깨지지 않게 불러온다.



Elixirr

CSV 데이터 파일 읽어오기

✓ 데이터 파일의 이름을 적어줄 때 같은 폴더 안에 위치 하지 않다면 경로명까지 정확히 적어준다.

✓ csv파일을 읽어서 데이터프레임 객체로 만들어서 반환한다.

(보통 df라는 변수명을 사용)

pd.read_csv('파일이름')

데이터 프레임, 시리즈

	이름	나이	직업	국적	키	재산
0	아이언맨	58	CEO	미국	174	11500000000000
1	토르	45	신	아스가르드	190	NaN
2	스파이더맨	18	고등학생	미국	177	600
3	원더우먼	27	공주	데미스키라	178	NaN
4	헐크	33	핵물리학자	미국	270	1000000000
5	캡틴마블	25	파일럿	미국	170	340000000
6	블랙위도우	32	스파이	러시아	170	50000000

데이터프레임
(2차원 배열)

데이터 정보 확인

행과 열의 크기, 컬럼명, 컬럼을 구성하는 값의 자료형 등을 요약하여 정보 출력

데이터프레임명.info()

칼럼별 통계량

✓ 숫자형 변수에 대한 기초 통계량을 출력한다.

데이터프레임명 .describe()

	나이	키	재산
count	7	7	5
mean	34	190	2300278000120
std	13	36	5142800956894
min	18	170	600
25%	26	172	50000000
50%	32	177	340000000
75%	39	184	1000000000
max	58	270	11500000000000

count	데이터개수	df.count()
mean	평균값	df.mean()
std	표준편차	
min	최소값	df.min()
25%	1사분위수	
50%	중위수	df.median()
75%	3사분위수	
max	최대값	df.max()

데이터 프레임의 특정 컬럼의 고유한 값 알아보기

✓ 특정 칼럼의 데이터 중 중복을 제외한 고유한 값 알아보기

데이터프레임명.[‘컬럼명’].unique()

데이터프레임명.[‘컬럼명’].value_counts()

제일 큰 값 순서대로 보여주기

✓ 기준 칼럼으로 정렬했을 때 상위 몇 개 까지 나타낼 지 지정

데이터프레임명.nlargest(n, 기준 칼럼)

데이터 일부 가져오기

✓ 상위 n개 행 :

```
데이터프레임명.head(n)
```

✓ 하위 n개 행 :

```
데이터프레임명.tail(n)
```

※ 기본값은 5개

특정 데이터 조회해서 가져오기

1 데이터프레임명[ ]

※간단히 사용하기엔 좋지만 간혹 헷갈릴 수 있음

2 데이터프레임명.iloc[행번호 , 열번호]

3 데이터프레임명.loc[인덱스명 , 컬럼명]

데이터프레임명[컬럼명] : 컬럼 데이터 가져오기

✓ 하나의 칼럼 가져오기

데이터프레임명[‘칼럼명’]

✓ 여러 개의 칼럼 가져오기 : 조회하려는 칼럼명을 리스트로 만든 후 조회

데이터프레임명[[‘ 칼럼명1 ’ , ‘ 칼럼명 2’, …,‘ 칼럼명 n’]]

데이터프레임명[행번호] : 데이터 행 가져오기

✓ 인덱스 번호를 슬라이싱으로 표현하면 행 단위로 데이터를 가져온다.

```
1 df
```

	이름	나이	직업	국적
0	아이언맨	58	CEO	미국
1	토르	45	신	아스가르드
2	스파이더맨	18	고등학생	미국
3	원더우먼	27	공주	데미스키라
4	헐크	33	핵물리학자	미국
5	캡틴마블	25	파일럿	미국
6	블랙위도우	32	스파이	러시아

인덱스로 슬라이싱 시
끝 미포함

```
5 df[0:2]
```

	이름	나이	직업	국적
0	아이언맨	58	CEO	미국
1	토르	45	신	아스가르드

데이터프레임명[인덱스명] : 데이터 행 가져오기

✓ 인덱스 명을 슬라이싱으로 표현하면 행 단위로 데이터를 가져온다.

```
3 df2 = pd.read_csv('files/heroes.csv')
4 df2.index = ['a', 'b', 'c', 'd', 'e', 'f','g']
5 df2
```

	이름	나이	직업	국적
a	아이언맨	58	CEO	미국
b	토르	45	신	아스가르드
c	스파이더맨	18	고등학생	미국
d	원더우먼	27	공주	데미스키라
e	헐크	33	핵물리학자	미국
f	캡틴마블	25	파일럿	미국
g	블랙위도우	32	스파이	러시아

라벨로 슬라이싱 시 **끝 포함**

```
1 df2['a': 'd']
```

	이름	나이	직업	국적
a	아이언맨	58	CEO	미국
b	토르	45	신	아스가르드
c	스파이더맨	18	고등학생	미국
d	원더우먼	27	공주	데미스키라

인덱스 번호로 데이터 가져오기 : 인덱싱

✓ iloc(Positional Indexing) : 위치를 통한 인덱싱

데이터프레임명.iloc[행번호 , 열번호]

```
3  df.iloc[0, 0]  #[행번호, 열번호]
```

```
'아이언맨'
```

```
1  df
```

	이름	나이	직업	국적
0	아이언맨	58	CEO	미국
1	토르	45	신	아스가르드
2	스파이더맨	18	고등학생	미국
3	원더우먼	27	공주	데미스키라
4	헐크	33	핵물리학자	미국
5	캡틴마블	25	파일럿	미국
6	블랙위도우	32	스파이	러시아

인덱스 번호로 데이터 잘라오기 : 슬라이싱

데이터프레임명.iloc[시작 행번호 : 끝 행 번호 , 시작 열 번호: 끝 열 번호]

```
3  df.iloc[0:1, 0:3 ]
```

	이름	나이	직업
0	아이언맨	58	CEO

```
1  df
```

	이름	나이	직업	국적
0	아이언맨	58	CEO	미국
1	토르	45	신	아스가르드
2	스파이더맨	18	고등학생	미국
3	원더우먼	27	공주	데미스키라
4	헐크	33	핵물리학자	미국
5	캡틴마블	25	파일럿	미국
6	블랙위도우	32	스파이	러시아

인덱스로 슬라이싱 시 **끝 미포함**

이름으로 데이터 가져오기 : 인덱싱

✓ loc(Label based Indexing) : 라벨을 통한 인덱싱

데이터프레임명.loc[인덱스명 , 칼럼명]

```
3  df.loc[0, '이름']
```

```
'아이언맨'
```

```
1  df
```

	이름	나이	직업	국적
0	아이언맨	58	CEO	미국
1	토르	45	신	아스가르드
2	스파이더맨	18	고등학생	미국
3	원더우먼	27	공주	데미스키라
4	헐크	33	핵물리학자	미국
5	캡틴마블	25	파일럿	미국
6	블랙위도우	32	스파이	러시아

이름으로 데이터 잘라오기 : 슬라이싱

데이터프레임명.loc[시작 인덱스명 : 끝 인덱스명, 시작 칼럼명: 끝 칼럼명]

```
3  df.loc[0:1, '이름' :'직업']
```

	이름	나이	직업
0	아이언맨	58	CEO
1	토르	45	신

```
1  df
```

	이름	나이	직업	국적
0	아이언맨	58	CEO	미국
1	토르	45	신	아스가르드
2	스파이더맨	18	고등학생	미국
3	원더우먼	27	공주	데미스키라
4	헐크	33	핵물리학자	미국
5	캡틴마블	25	파일럿	미국
6	블랙위도우	32	스파이	러시아

라벨로 슬라이싱 시 **끝 포함**

데이터프레임에 값이 없는 데이터가 있는가? isnull()

✓ 빈 값에 True를 출력한다.

데이터프레임명.isnull()

```
5 df2 = df.copy()
6 df2.iloc[5, 1] = None
7
8 df2
9
```

	이름	나이	직업	국적
0	아이언맨	58.0	CEO	미국
1	토르	45.0	신	아스가르드
2	스파이더맨	18.0	고등학생	미국
3	원더우먼	27.0	공주	데미스키라
4	헐크	33.0	핵물리학자	미국
5	캡틴마블	NaN	파일럿	미국
6	블랙위도우	32.0	스파이	러시아

```
3 df2.isnull()
```

	이름	나이	직업	국적
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False
5	False	True	False	False
6	False	False	False	False

```
3 df2.isnull().sum()
```

```
이름    0
나이    1
직업    0
국적    0
dtype: int64
```

컬럼별로 null값이 몇 개 있는지 알려준다.

값이 없는 데이터 가져오기

✓ 해당 컬럼에서 값이 null인 데이터들을 가져온다.

데이터프레임명[컬럼명.isnull()]

```
df[df['재산'].isnull()]
```

	이름	나이	직업	국적	키	재산
1	토르	45	신	아스가르드	190	NaN
3	원더우먼	27	공주	데미스키라	178	NaN

조건에 따라 데이터 가져오기: 불리언 인덱싱(필터링)

✓ 조건을 제시하고 조건이 True인 요소만을 조회

데이터프레임명[조건]

```
1 df
```

	이름	나이	직업	국적
0	아이언맨	58	CEO	미국
1	토르	45	신	아스가르드
2	스파이더맨	18	고등학생	미국
3	원더우먼	27	공주	데미스키라
4	헐크	33	핵물리학자	미국
5	캡틴마블	25	파일럿	미국
6	블랙위도우	32	스파이	러시아

```
3 df['나이'] > 30
```

```
0     True
1     True
2    False
3    False
4     True
5    False
6     True
Name: 나이, dtype: bool
```

조건식의 참/거짓 결과를 보여준다.

```
3 df[df['나이'] > 30]
```

	이름	나이	직업	국적
0	아이언맨	58	CEO	미국
1	토르	45	신	아스가르드
4	헐크	33	핵물리학자	미국
6	블랙위도우	32	스파이	러시아

- df[]의 안에 조건식을 넣어 준다.
- 조건식이 참인 행들을 가져온다.

조건식을 만드는 논리 연산자

논리 연산자	의미	예	설명
&	논리곱	(A) & (B)	A와 B가 모두 참일 때만 참
|	논리합	A | B	A와 B가 모두 거짓일때만 거짓 (둘 중 하나라도 참이면 결과는 참)
~	논리 부정	~ A	A가 참이면 거짓, 거짓이면 참
^	배타적 논리합	A ^ B	A와 B가 모두 참이거나 모두 거짓이면 False A와 B가 하나는 True, 하나는 False면 True

데이터 가져오기 : 불리언 인덱싱(필터링)

✓ 여러 개의 조건을 and연산(&) 하거나 or연산(|)할 수 있다.

✓ 이 때 조건식은 소괄호()안에 넣어준다.

데이터프레임명[(조건1) [&,|] (조건2)]

데이터 가져오기 : 불리언 인덱싱(필터링)

✓ 조건이 참인 행 중에 열에 해당하는 데이터를 가져온다.

데이터프레임명.loc[(조건식), 열]

칼럼별 정렬

✓ 컬럼의 값을 기준으로 데이터프레임을 정렬한다.

데이터프레임명.sort_values('컬럼명')

```
1 df.sort_values('나이') #나이 오름차순 정렬
```

	이름	나이	직업	국적
2	스파이더맨	18	고등학생	미국
5	캡틴마블	25	파일럿	미국
3	원더우먼	27	공주	데미스키라
6	블랙위도우	32	스파이	러시아
4	헐크	33	핵물리학자	미국
1	토르	45	신	아스가르드
0	아이언맨	58	CEO	미국

```
1 df.sort_values('나이', ascending = False ) #나이 내림차순 정렬
```

	이름	나이	직업	국적
0	아이언맨	58	CEO	미국
1	토르	45	신	아스가르드
4	헐크	33	핵물리학자	미국
6	블랙위도우	32	스파이	러시아
3	원더우먼	27	공주	데미스키라
5	캡틴마블	25	파일럿	미국
2	스파이더맨	18	고등학생	미국

데이터 수정 또는 추가하기

✓ 컬럼이 있다면 값이 수정된다.

✓ 컬럼이 없다면 새로운 컬럼이 추가된다.

데이터프레임명['컬럼명'] = 데이터

인덱스명, 컬럼명 수정하기

데이터프레임명.index = ['인덱스1 ', …,'인덱스n 명']

데이터프레임명.columns= ['컬럼명1', …, '컬럼명n']

데이터 삭제하기

✓ 데이터 행 삭제

```
데이터프레임명.drop(index=[삭제할 인덱스])
```

✓ 컬럼 삭제

```
데이터프레임명.drop(columns=[삭제할 컬럼명])
```

※ 원본에서 삭제하려면 inplace 값을 True로 설정하거나 데이터프레임에 다시 할당해야 한다.

결측치 삭제하기

✓ 빈 값이 있는 데이터가 있는 행 삭제하기 (axis=0이 기본값)

```
데이터프레임명.dropna()
```

✓ 빈 값이 있는 데이터가 있는 열 삭제하기

```
데이터프레임명.dropna(axis = 1)
```

결측치에 값 채우기

✓ 결측치에 어떤 값으로 채우기

	이름	나이	직업	국적
0	아이언맨	58.0	CEO	미국
1	토르	45.0	신	아스가르드
2	스파이더맨	18.0	고등학생	미국
3	원더우먼	27.0	공주	데미스키라
4	헐크	33.0	핵물리학자	미국
5	캡틴마블	NaN	파일럿	미국
6	블랙위도우	32.0	스파이	러시아

데이터프레임명.fillna()

특정값으로 채운다

```
7 df3.fillna(0)
```

	이름	나이	직업	국적
0	아이언맨	58.0	CEO	미국
1	토르	45.0	신	아스가르드
2	스파이더맨	18.0	고등학생	미국
3	원더우먼	27.0	공주	데미스키라
4	헐크	33.0	핵물리학자	미국
5	캡틴마블	0.0	파일럿	미국
6	블랙위도우	32.0	스파이	러시아

다음 값으로 채운다

```
7 df3.fillna(method = 'bfill')
```

	이름	나이	직업	국적
0	아이언맨	58.0	CEO	미국
1	토르	45.0	신	아스가르드
2	스파이더맨	18.0	고등학생	미국
3	원더우먼	27.0	공주	데미스키라
4	헐크	33.0	핵물리학자	미국
5	캡틴마블	32.0	파일럿	미국
6	블랙위도우	32.0	스파이	러시아

이전 값으로 채운다

```
7 df3.fillna(method = 'ffill')
```

	이름	나이	직업	국적
0	아이언맨	58.0	CEO	미국
1	토르	45.0	신	아스가르드
2	스파이더맨	18.0	고등학생	미국
3	원더우먼	27.0	공주	데미스키라
4	헐크	33.0	핵물리학자	미국
5	캡틴마블	33.0	파일럿	미국
6	블랙위도우	32.0	스파이	러시아

그룹별 데이터 집계하기

✓ 칼럼 데이터를 기준으로 데이터를 집계한다.

데이터프레임명.groupby('칼럼명').통계함수()

```
국적
데미스키라    27
러시아       32
미국         34
아스가르드    45
Name: 나이, dtype: float64
```

나라별 히어로들의 평균 나이

함수 적용하기

✓ 컬럼에 함수를 일괄적으로 적용한다.

데이터프레임명['컬럼명'].apply(함수명)

```
def add_cm(height):
  return str(height) + 'cm'

df3['키'] = df3['키'].apply(add_cm)
df3
```

	이름	나이	직업	키	재산
0	아이언맨	58	CEO	174cm	11500000000000.00000
1	토르	45	신	190cm	NaN
2	스파이더맨	18	고등학생	177cm	600.00000
3	원더우먼	27	공주	178cm	NaN
4	헐크	33	핵물리학자	270cm	1000000000.00000
5	NaN	25	파일럿	170cm	340000000.00000
6	블랙위도우	32	스파이	170cm	50000000.00000

Matplotlib

데이터 시각화

- ✓ 전체적인 데이터의 구조를 시각적으로 분석하거나 분석방향을 가늠하기 위해 탐색적 데이터 분석(EDA)에서 사용.
- ✓ 분석 결과를 도식화하여 의사 결정에 반영하기 위해 사용 (데이터 표현)
- ✓ 시각화 전에 데이터 전처리를 수행

데이터 시각화의 목적에 맞는 그래프 선택

- ✓ 데이터의 비교와 순위 매기기(Ranking) : 막대 그래프
- ✓ 전체 중 얼만큼의 비중인지 알아보기 위해: 파이 그래프
- ✓ 트랜드(추세)를 알아보기 위해 : 선 그래프, 막대 그래프
- ✓ 어떤 요인간의 관계를 알아보기 위해 :히트맵, 버블, 산점도
- ✓ 데이터의 분포를 보기 위해 :히스토그램, 지도 , 박스플롯

matplotlib을 이용한 시각화

✓ 파이썬에서 가장 많이 사용하는 시각화 라이브러리 중 하나

✓ 다양한 차트와 플롯(plot)을 그릴 수 있다.

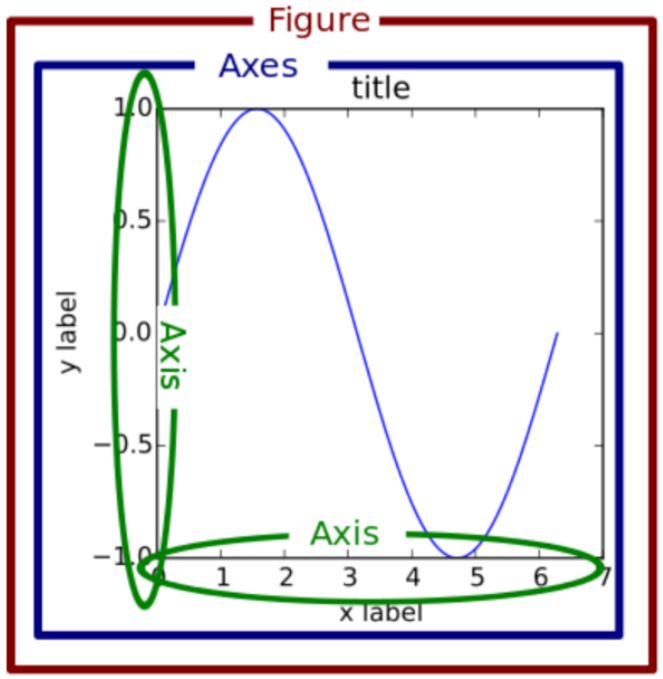
- line plot, bar chart, pie chart, histogram, box plot, scatter plot 등

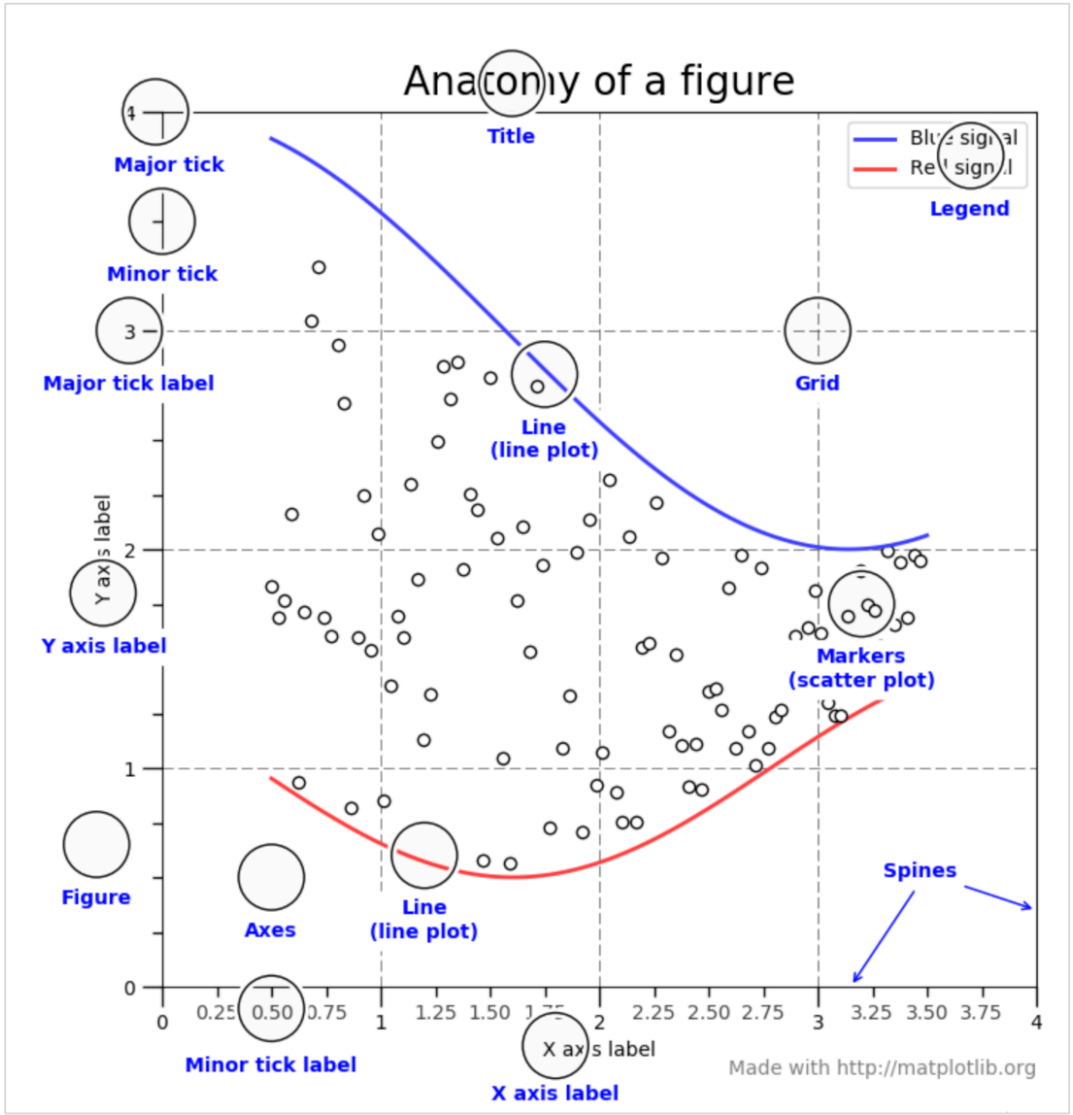
✓ 넘파이나 판다스가 제공하는 자료들과 잘 연동된다.

※ 튜토리얼 사이트 https://matplotlib.org/stable/tutorials/index.html

pyplot 의 구성 요소

- ✓ figure : 그림이 그려지는 도화지
  - 도화지 분할 가능
  - figure의 크기 조절 가능
- ✓ Axes : plot이 그려지는 공간
- ✓ Axis : plot의 축





선 그래프(line plot)

✓ 시간에 따른 데이터의 연속적인 변화량(트랜드, 추세)을 관찰하고자 할 때

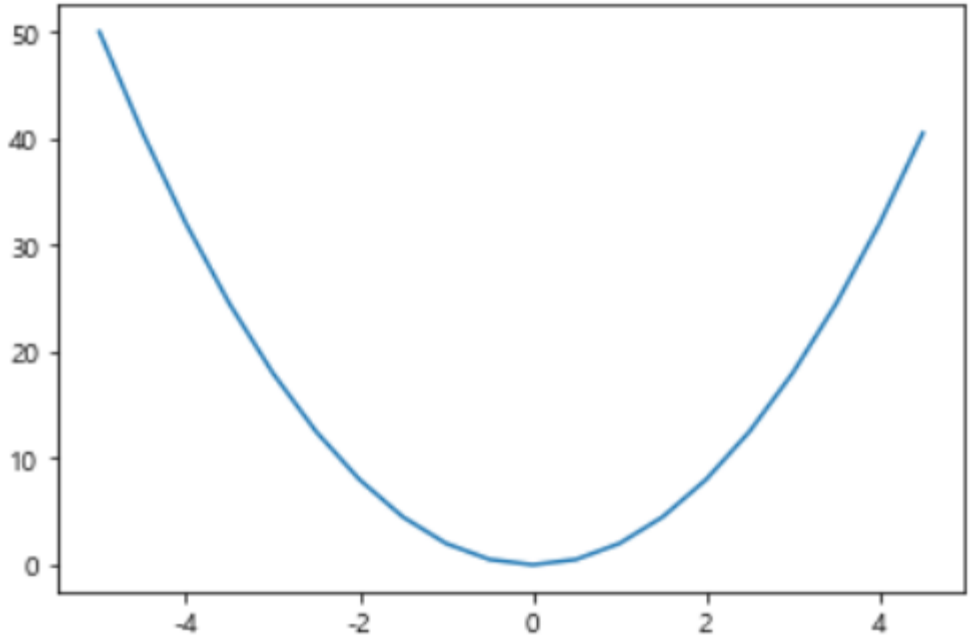
```
plt.plot([x], y, [fmt])
```

선 그래프(line plot)

✓ 기본 그래프 그리기 : x축, y축 데이터

```
x = np.arange(-5, 5, 0.5)
y = 2*x**2

plt.plot(x, y)
plt.show()
```



선 그래프(line plot)

✓ 그래프의 제목 넣기

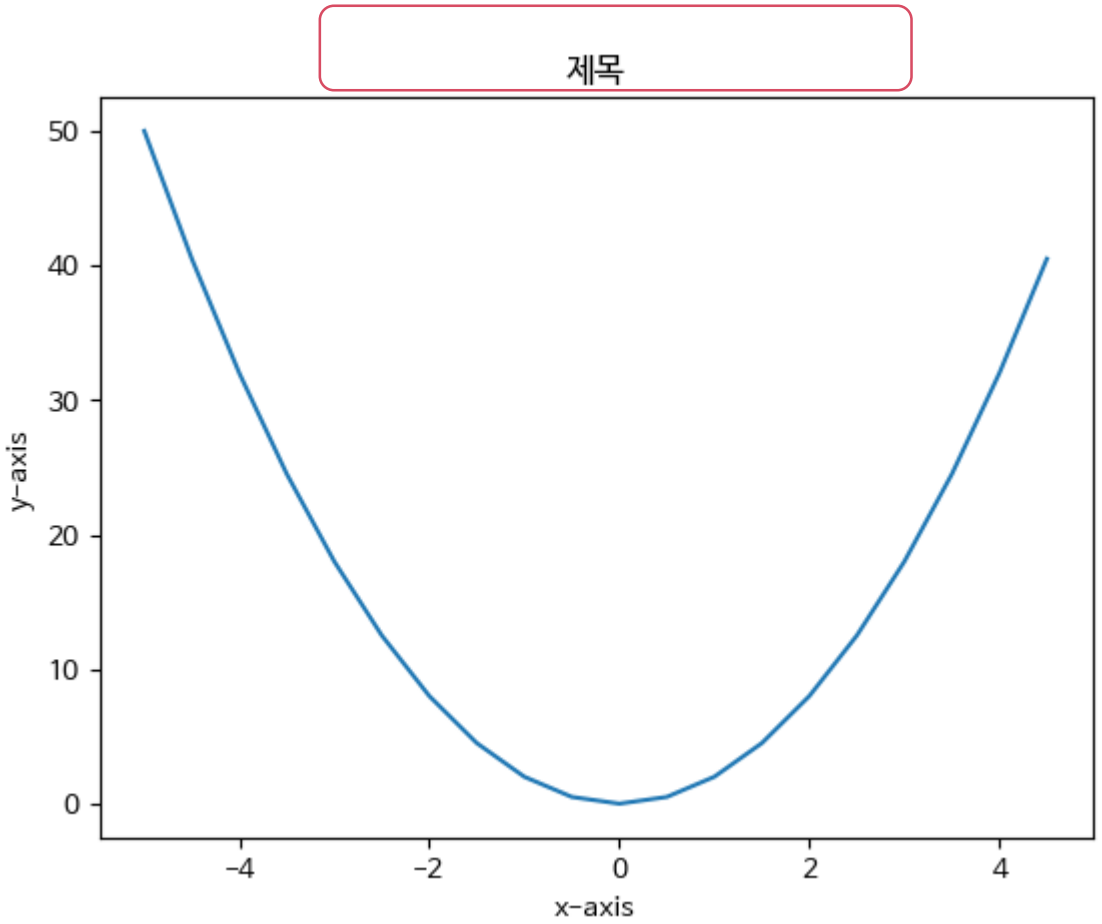
plt.title ('제목')

```
x = np.arange(-5, 5, 0.5)
y = 2*x**2

plt.plot(x, y)

plt.title('제목')
plt.xlabel('x-axis')
plt.ylabel('y-axis')

plt.show()
```



Elixirr

선 그래프(line plot)

✓ x축 라벨 표시

plt.xlabel ('라벨')

✓ y축 라벨 표시

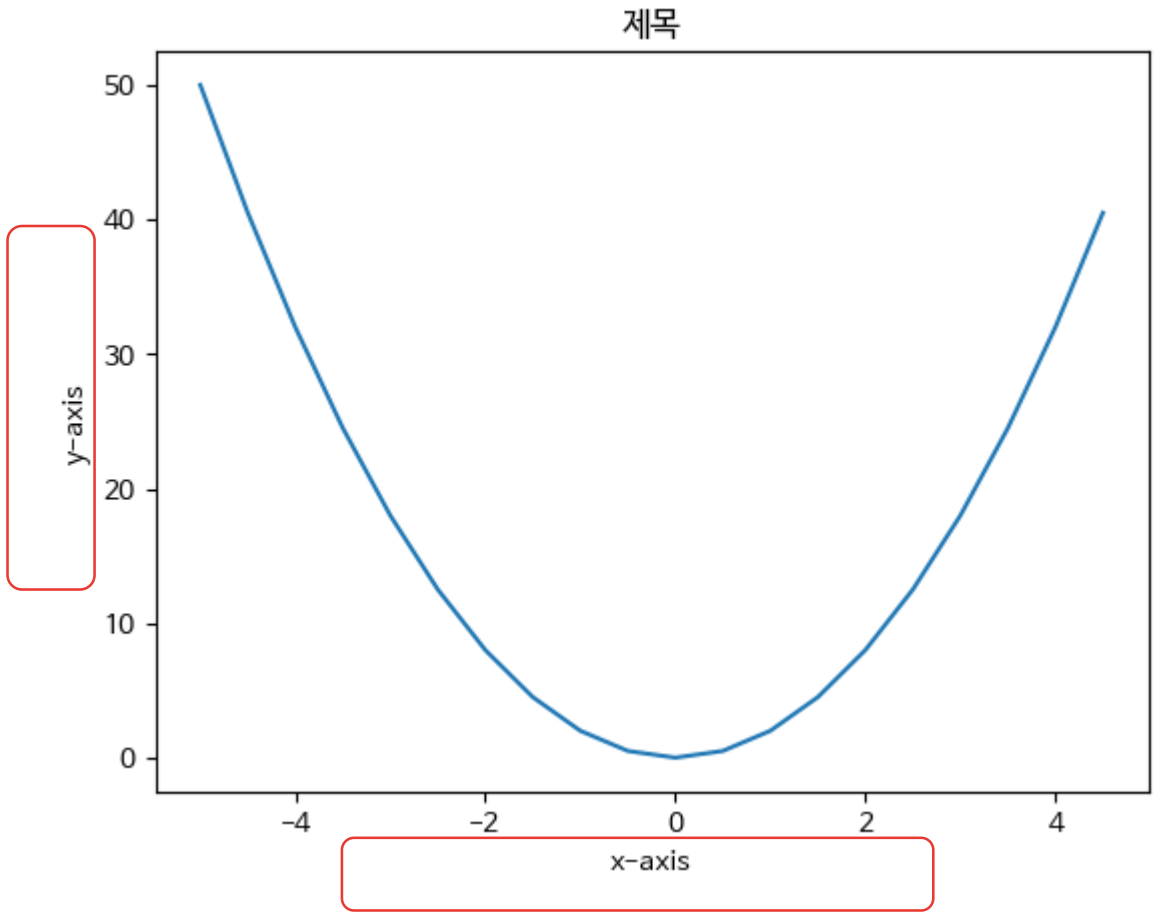
plt.ylabel ('라벨')

```
x = np.arange(-5, 5, 0.5)
y = 2*x**2

plt.plot(x, y)

plt.title('제목')
plt.xlabel('x-axis')
plt.ylabel('y-axis')

plt.show()
```



선 그래프(line plot)

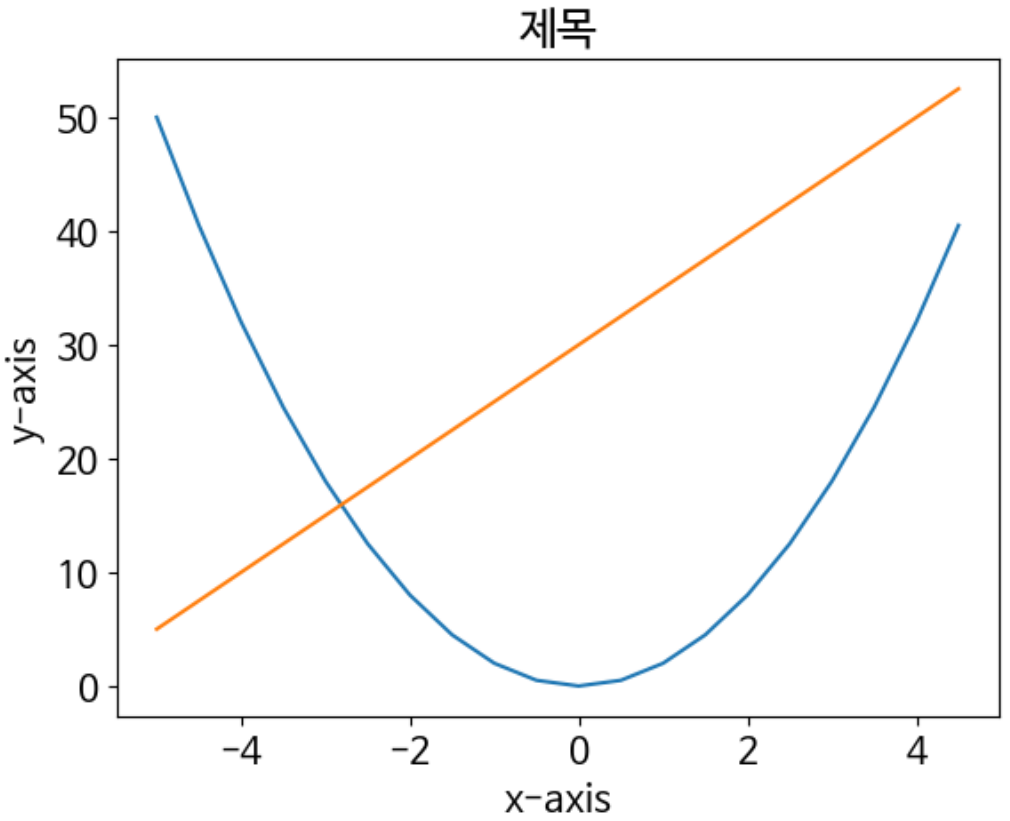
✓ 그래프 추가

```
x = np.arange(-5, 5, 0.5)
y1 = 2*x**2
y2 = 5*x + 30

plt.plot(x, y1, x, y2)

plt.title('제목')
plt.xlabel('x-axis')
plt.ylabel('y-axis')

plt.show()
```



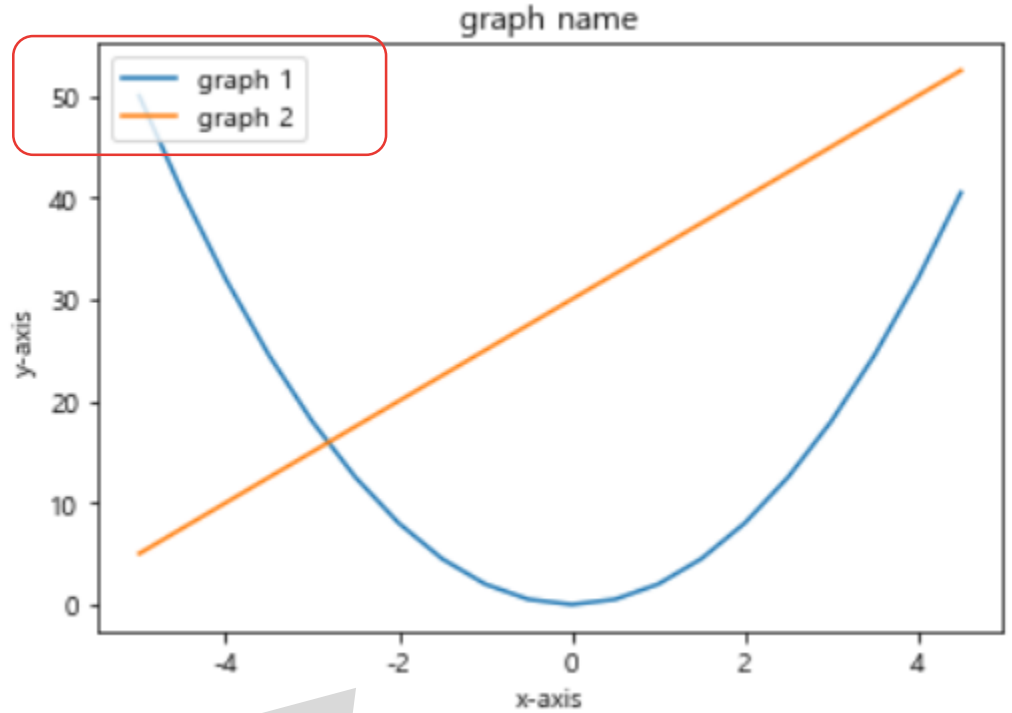
선 그래프(line plot)

✓ 범례 위치 지정하기

```
x = np.arange(-5, 5, 0.5)
y1 = 2*x**2
y2 = 5*x + 30

plt.plot(x, y1,label = 'graph 1')
plt.plot(x, y2,label = 'graph 2')
plt.legend(loc = 2)

plt.title('graph name')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```



default: 최적의 위치

[범례 위치]

2	9	1
6	10	5,7
3	8	4

선 그래프(line plot)

✓ 격자 표시하기

plt.grid (True/False)

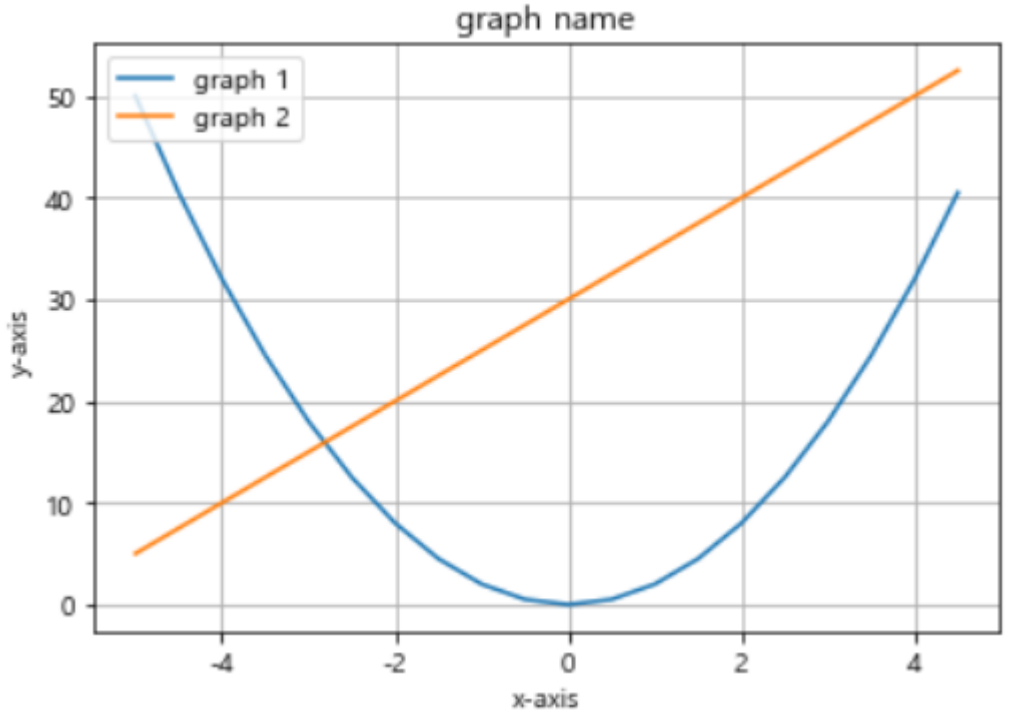
```
x = np.arange(-5, 5, 0.5)
y1 = 2*x**2
y2 = 5*x + 30

plt.plot(x, y1,label = 'graph 1')
plt.plot(x, y2,label = 'graph 2')
plt.legend(loc = 2)

plt.title('graph name')
plt.xlabel('x-axis')
plt.ylabel('y-axis')

plt.grid(True)

plt.show()
```



Elixirr

선 그래프(line plot)

✓ 그래프 꾸미기 : 출력의 형식 '[색깔] [선 모양] [마커 모양]'

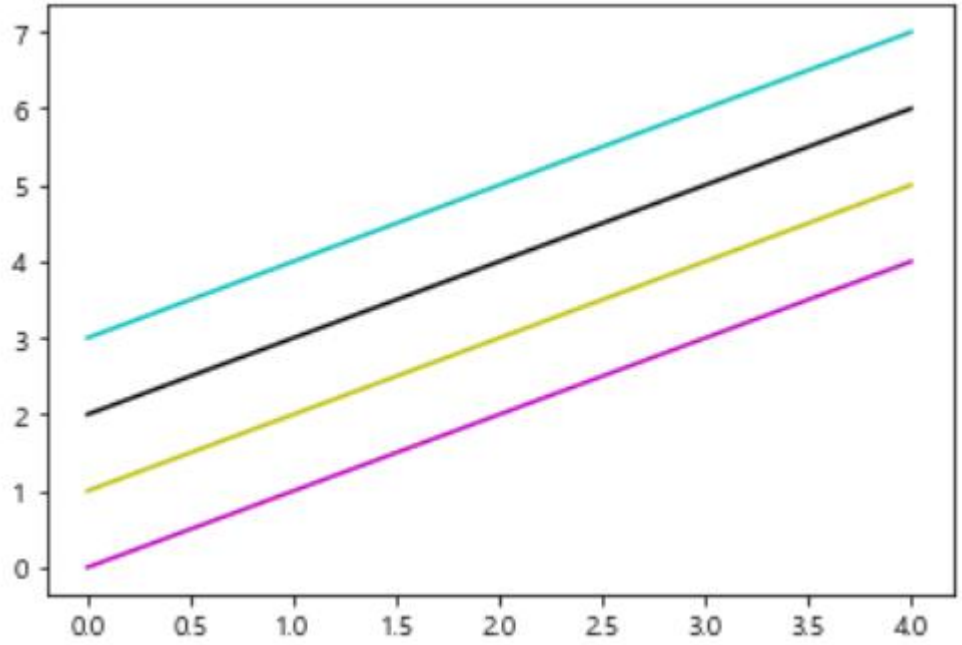
✓ 그래프 색상 지정하기

```
x = np.arange(0, 5, 1)
y1 = x
y2 = x + 1
y3 = x + 2
y4 = x + 3

plt.plot(x, y1, 'm')
plt.plot(x, y2, 'y')
plt.plot(x, y3, 'k')
plt.plot(x, y4, 'c')

plt.show()
```

색상 약어	컬러
b	파랑
g	녹색
r	빨강
c	청녹색(cyan)
m	자홍색(magenta)
y	노랑
k	검은색(balck)
w	흰색(white)



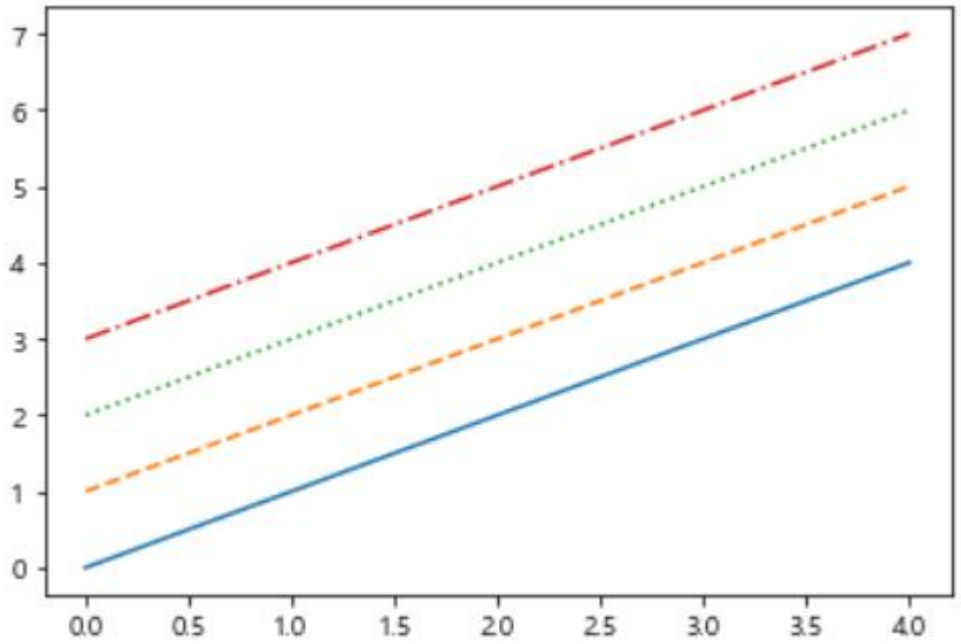
선 그래프(line plot)

✓ 선 모양 지정하기

```
x = np.arange(0, 5, 1)
y1 = x
y2 = x + 1
y3 = x + 2
y4 = x + 3

plt.plot(x, y1, '-')
plt.plot(x, y2, '--')
plt.plot(x, y3, ':')
plt.plot(x, y4, '-.')

plt.show()
```



선 스타일 약어	선 스타일
-	실선 (solid)
--	파선(dashed)
:	점선(dotted)
-.	파선-점선 혼합(dash-dot)

선 그래프(line plot)

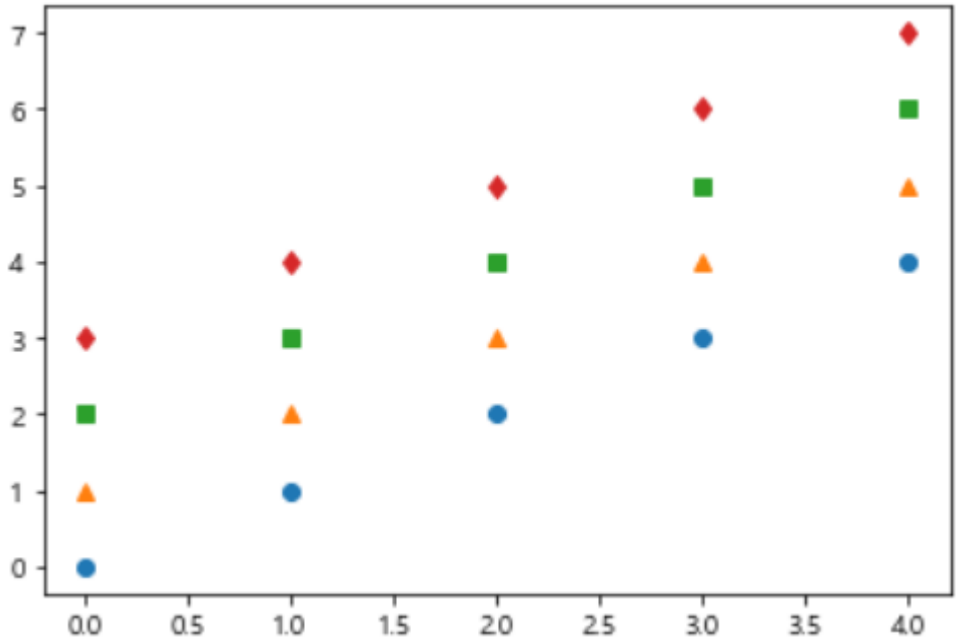
✓ 마커 추가하기

```
x = np.arange(0, 5, 1)
y1 = x
y2 = x + 1
y3 = x + 2
y4 = x + 3

plt.plot(x, y1, 'o')
plt.plot(x, y2, '^')
plt.plot(x, y3, 's')
plt.plot(x, y4, 'd')

plt.show()
```

마커 약어	마커
.	●
o	●
^	▲
v	▼
>	◀
x	X
s	■
d	◆



https://matplotlib.org/stable/api/markers_api.html

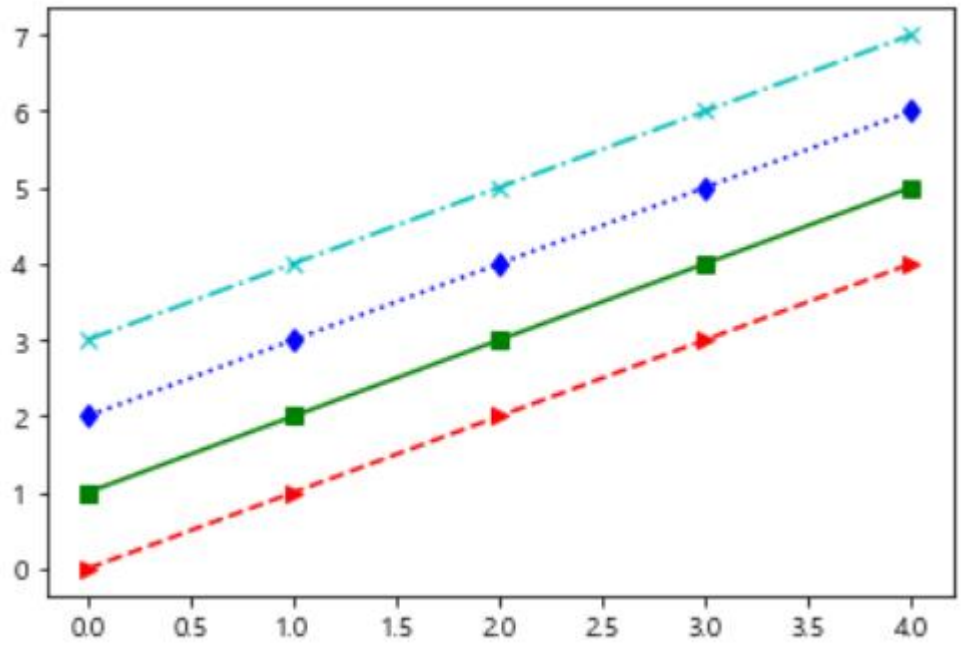
선 그래프(line plot)

✓ 색상과 선스타일, 마커를 한꺼번에 지정하기

```
x = np.arange(0, 5, 1)
y1 = x
y2 = x + 1
y3 = x + 2
y4 = x + 3

plt.plot(x, y1, '>--r')
plt.plot(x, y2, 's-g')
plt.plot(x, y3, 'd:b')
plt.plot(x, y4, '-.xc')

plt.show()
```



선 그래프(line plot)

✓ 그래프 안 x, y 위치에 text(텍스트) 넣기

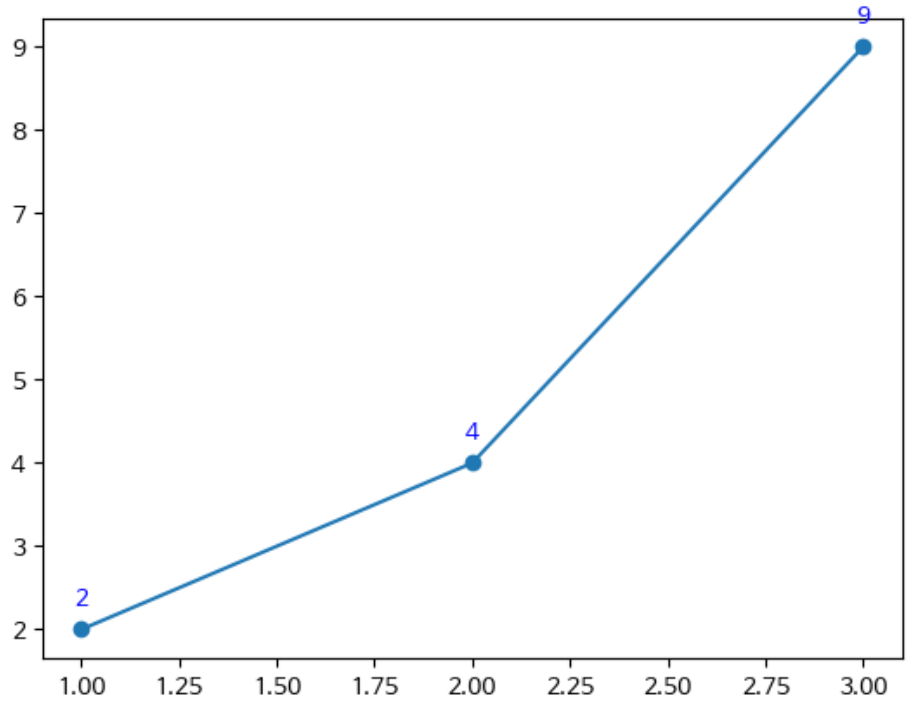
plt.text (x , y, text)

```
x = [1, 2, 3]
y = [2, 4, 9]

plt.plot(x, y, marker='o')

for idx, data in enumerate(y):
  plt.text(x[idx], y[idx]+0.3, data , ha='center', color = 'blue')

plt.show()
```



막대 그래프

✓ 여러 항목의 데이터를 비교할 때, 랭킹을 표시할 때 주로 사용

plt.bar (x, height [, width, color, tick_label, align, label])

- x: height와 길이가 일치하는 데이터로 x축에 표시될 위치를 지정.
- height : 막대로 표시할 데이터
- width : 막대의 폭을 지정(0~1사이의 실수를 지정. 기본값은 0.8)
- color : 막대그래프의 색깔을 지정
- tick_label : 막대그래프 각각의 이름을 지정(기본값:숫자)
- align : x 위치를 기준으로 막대그래프의 정렬(기본값: center)
- label : 범례에 표시할 이름을 지정

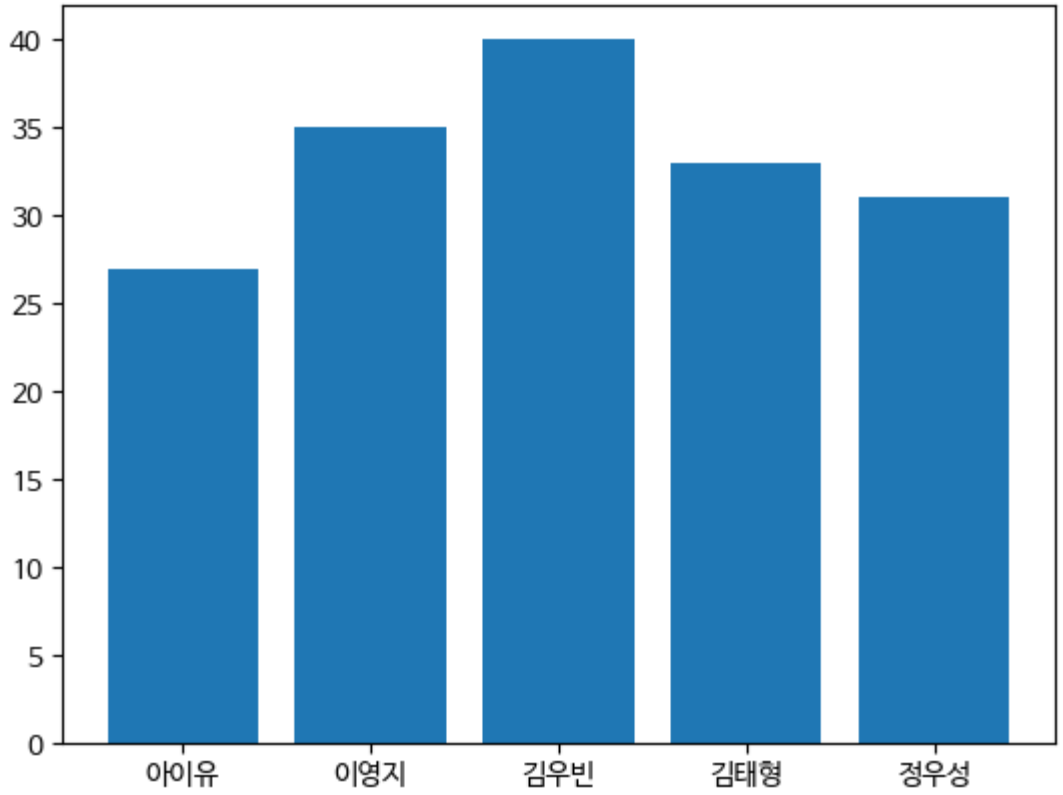
막대 그래프

✓ 기본 막대 그래프 그리기

```
x = df['회원ID']
y = df['운동시작전']

plt.bar(x, y)

plt.show()
```



[데이터: : 피트니스 회원별 윗몸일으키기 횟수]

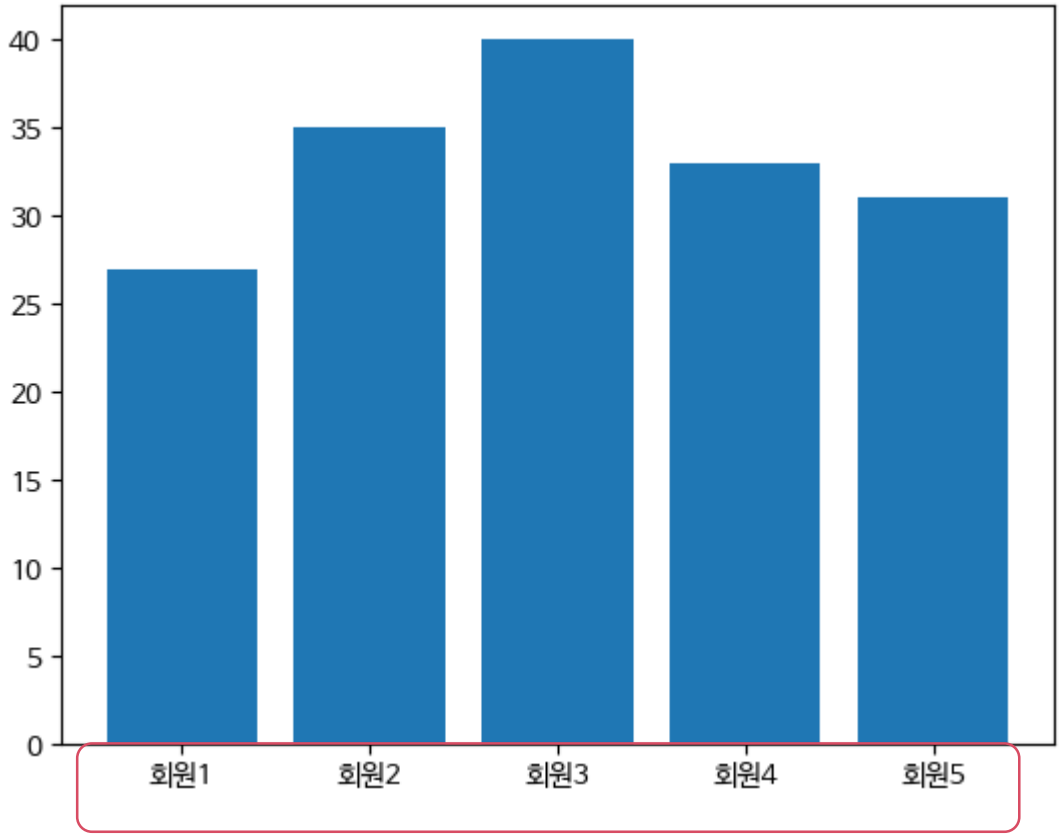
	회원ID	운동시작전	운동시작후
0	아이유	27	35
1	이영지	35	38
2	김우빈	40	42
3	김태형	33	37
4	정우성	31	34

막대 그래프

✓ x축 tick의 라벨을 지정

```
x = df['회원ID']
y = df['운동시작전']
ticks = ['회원1', '회원2', '회원3', '회원4', '회원5']

plt.bar(x, y)
plt.xticks(x, ticks)
plt.show()
```



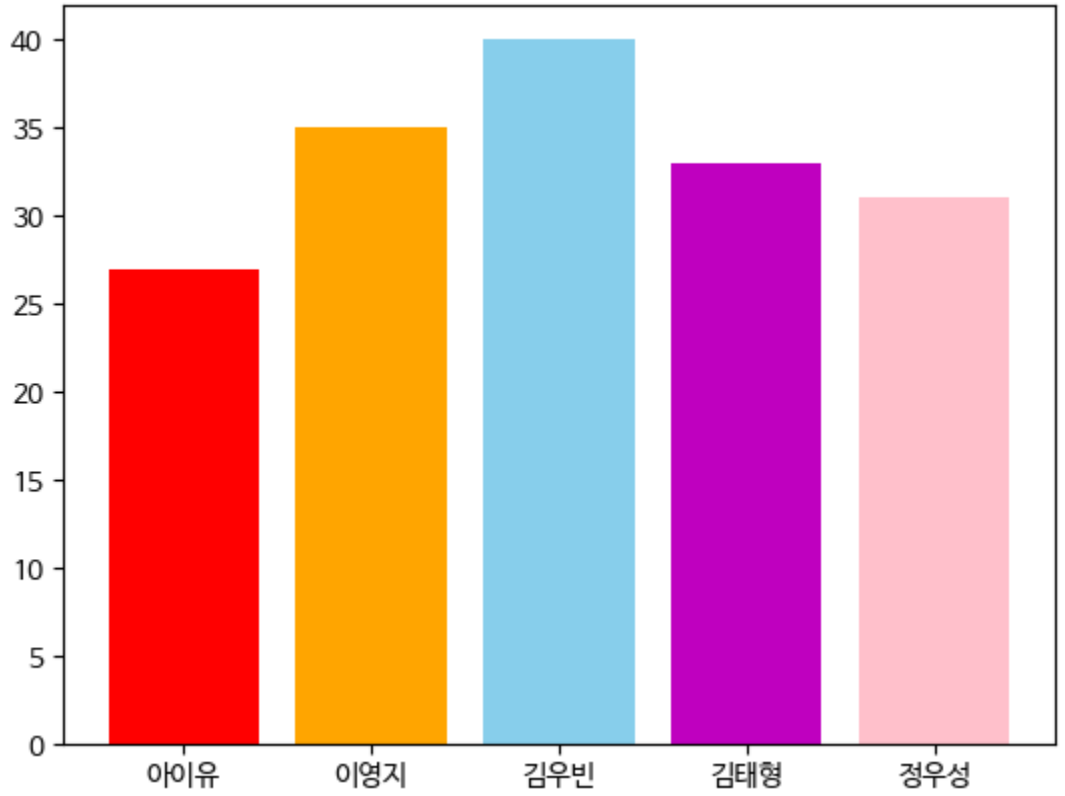
막대 그래프

✓ 막대 그래프의 색깔 지정

```
x = df['회원ID']
y = df['운동시작전']

colors = ['r', 'orange', 'skyblue', 'm', 'pink']

plt.bar(x, y, color = colors)
plt.show()
```



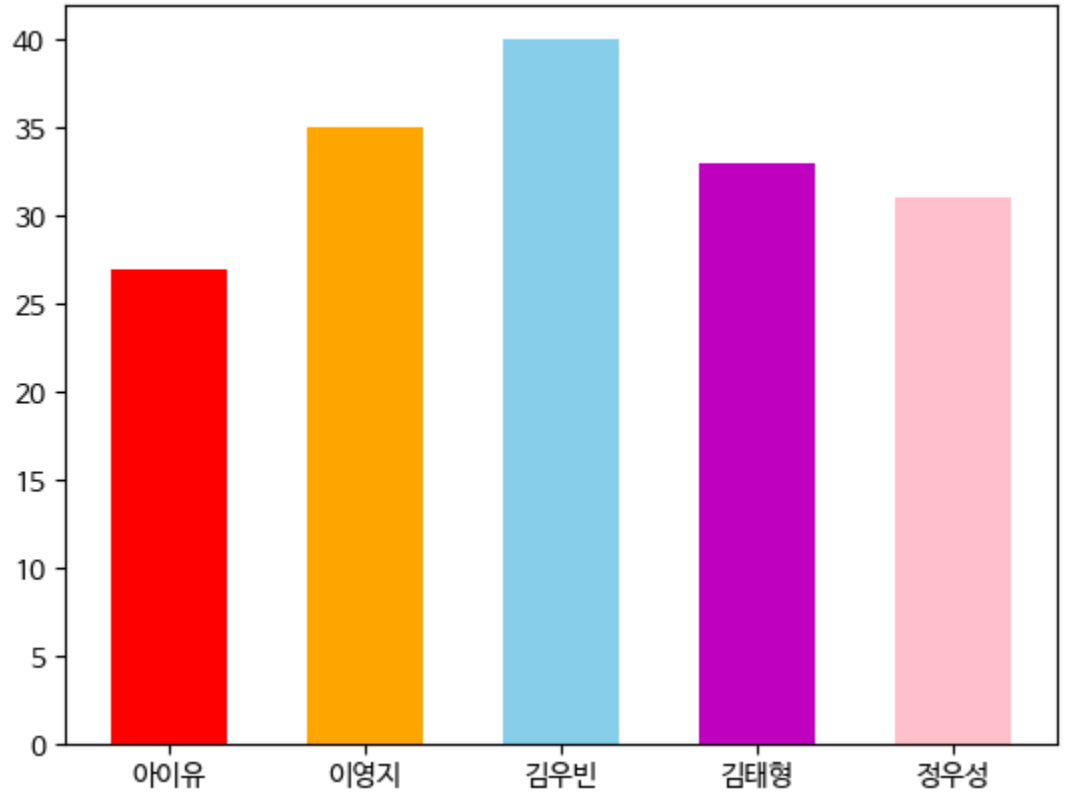
막대 그래프

✓ 막대의 폭 지정

```
x = df['회원ID']
y = df['운동시작전']

bar_width = 0.6
colors = ['r', 'orange', 'skyblue', 'm', 'pink']

plt.bar(x, y, color = colors, width = bar_width)
plt.show()
```



Elixirr

막대 그래프

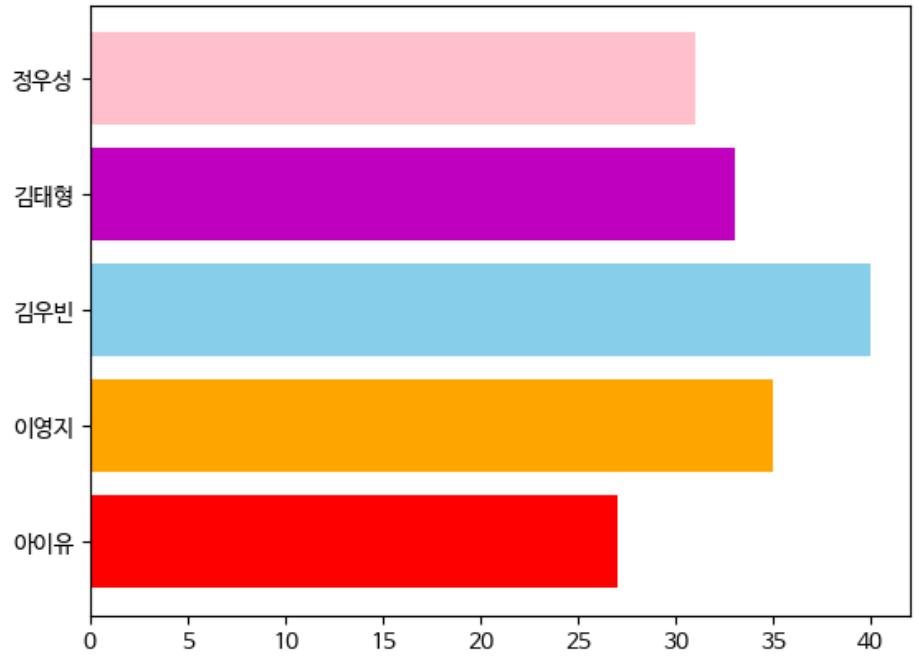
✓ 가로막대 그래프

- plt.barh() 사용
- 데이터 양의 비교와 함께 순위를 보는 데 사용하는 편

```
x = df['회원ID']
y = df['운동시작전']

colors = ['r', 'orange', 'skyblue', 'm', 'pink']

plt.barh(x, y, color = colors)
plt.show()
```



막대 그래프

✓ 가로막대 그래프

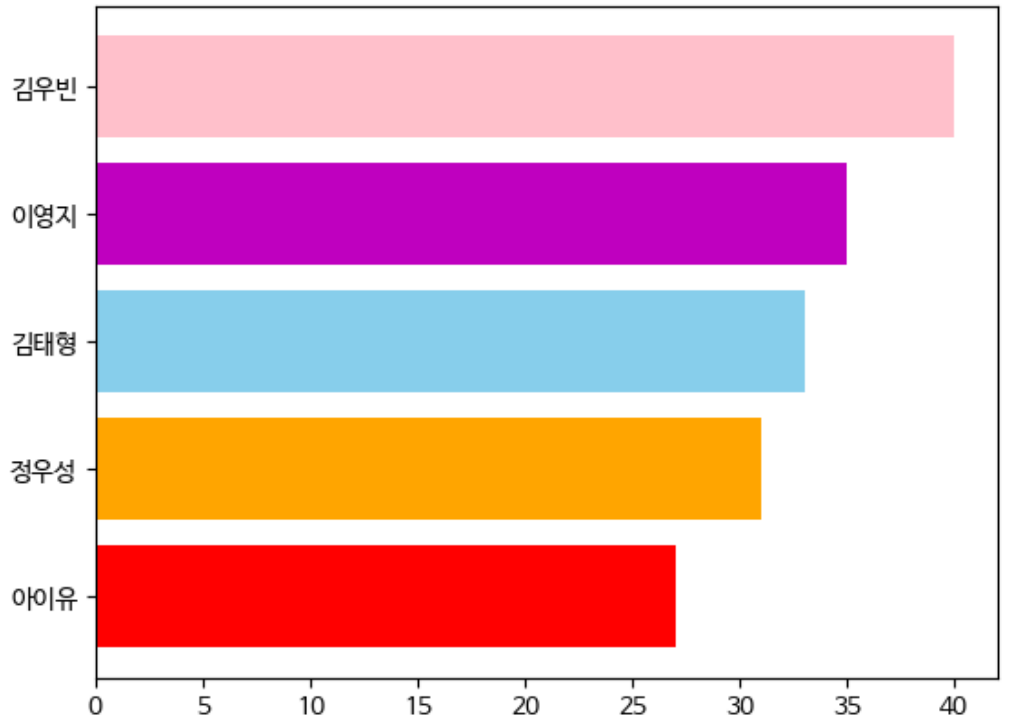
- 운동 시작 전 데이터 순서대로 정렬하여 표시

```
df_sorted = df.sort_values('운동시작전')

x = df_sorted['회원ID']
y = df_sorted['운동시작전']

colors = ['r', 'orange', 'skyblue', 'm', 'pink']

plt.barh(x, y, color = colors)
plt.show()
```



Elixirr

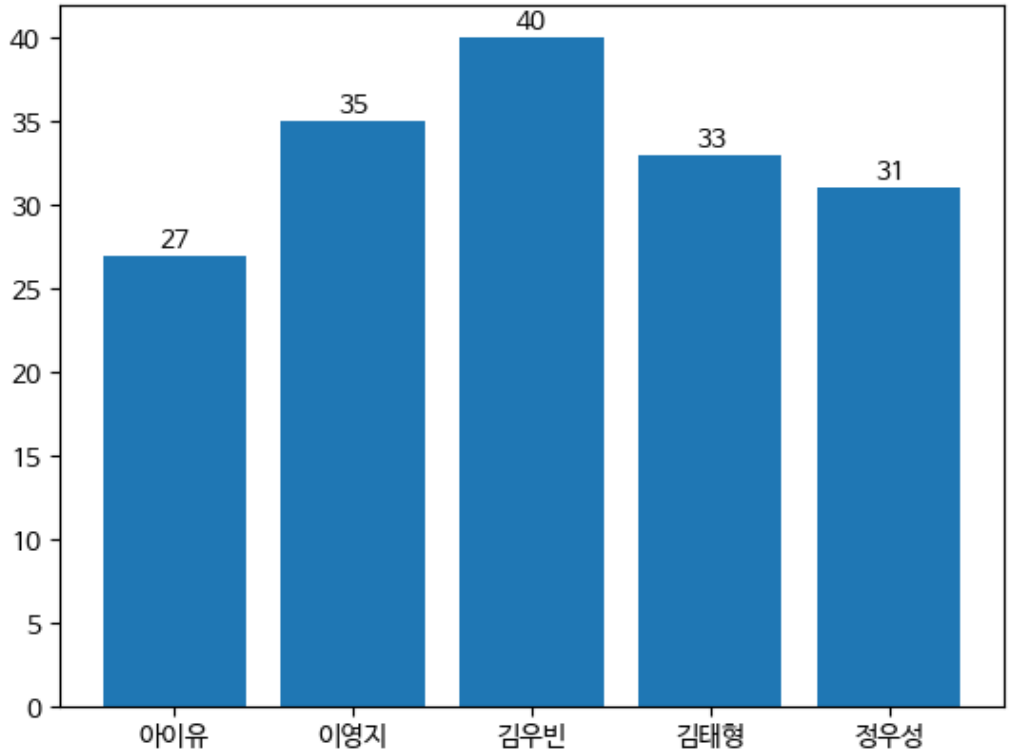
막대 그래프

✓ 그래프 안에 데이터 표시하기

```
x = df['회원ID']
y = df['운동시작전']

bar = plt.bar(x, y)

for idx, rect in enumerate(bar):
  plt.text(idx, rect.get_height()+0.5, df.loc[idx, '운동시작전'], ha = 'center')
```



Elixirr

막대 그래프

✓ 누적 막대 그래프

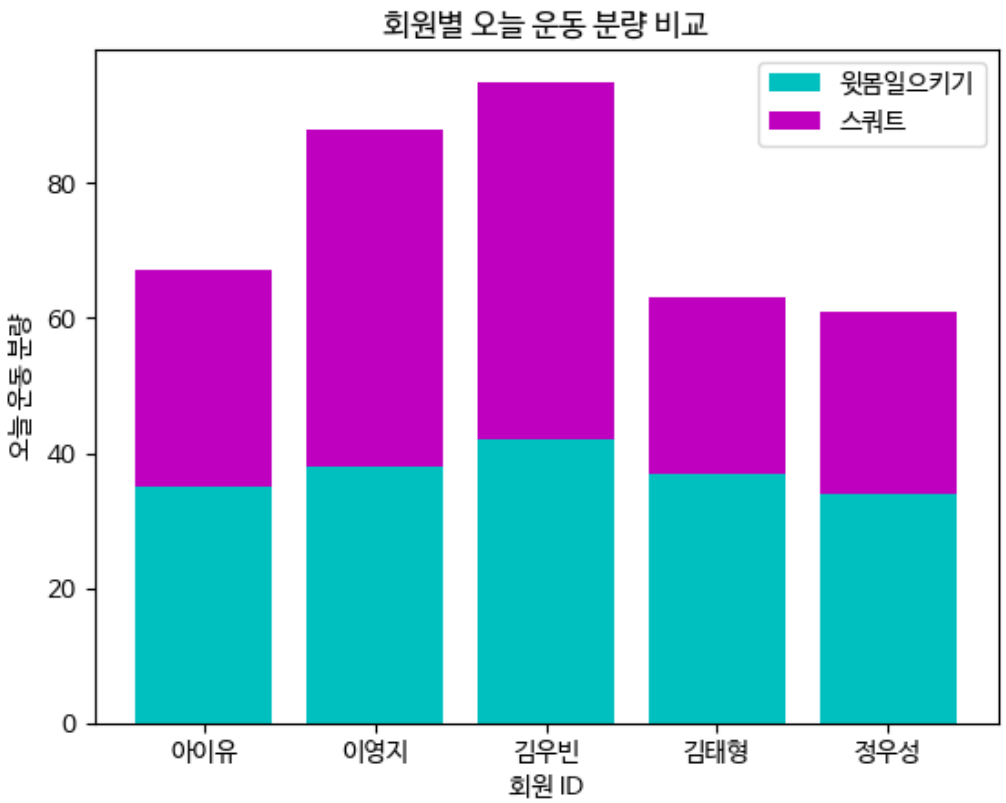
- 누적할 막대의 bottom을 지정한다.

```
x = df['회원ID']

plt.bar(x, df['운동시작후'], color = 'c', label = '윗몸일으키기')
plt.bar(x, df['스쿼트'], color = 'm', label = '스쿼트', bottom = df['운동시작후'])

plt.legend()
plt.xlabel('회원ID')
plt.ylabel('오늘 운동 분량')
plt.title('회원별 오늘 운동 분량 비교')

plt.show()
```



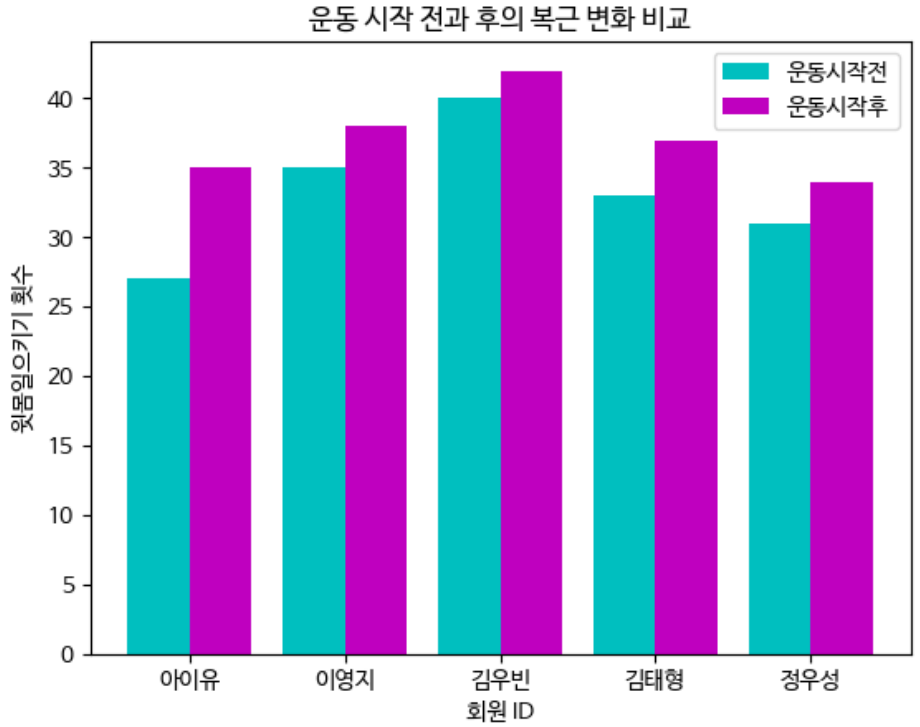
Elixirr

막대 그래프

✓ 다중 막대 그래프

- 몇 개의 막대를 다중으로 그릴지에 따라 width를 결정하고 x를 지정한다.

- 다중으로 그릴 막대의 개수만큼 plt.bar()로 그린다.




```
n_data = len(df)
x = np.arange(n_data)
bar_width = 0.4

plt.bar(x, df['운동시작전'], color = 'c', align = 'edge', width = bar_width, label = '운동시작전')
plt.bar(x+bar_width, df['운동시작후'], color = 'm', align = 'edge', width = bar_width, label = '운동시작후')

plt.xticks(x+bar_width, df['회원ID'])
plt.legend()
plt.xlabel('회원ID')
plt.ylabel('윗몸일으키기 횟수')
plt.title('운동 시작 전과 후의 복근 변화 비교')

plt.show()
```

산점도(scatter plot)

✓ 2개의 연속형 변수간의 관계(영향)를 보기 위해 x축과 y축에 관측점을 찍어서 만든 그래프

plt.scatter (x, y, [s, c, marker, alpha])

- s: 마커의 크기(기본값 36)
- c: 색깔(기본값 'b')
- marker : 마커의 모양(기본값 'o')
- alpha : 투명도 지정(기본값 1) 0 ~1 사이의 실수 사용.

산점도(scatter plot)

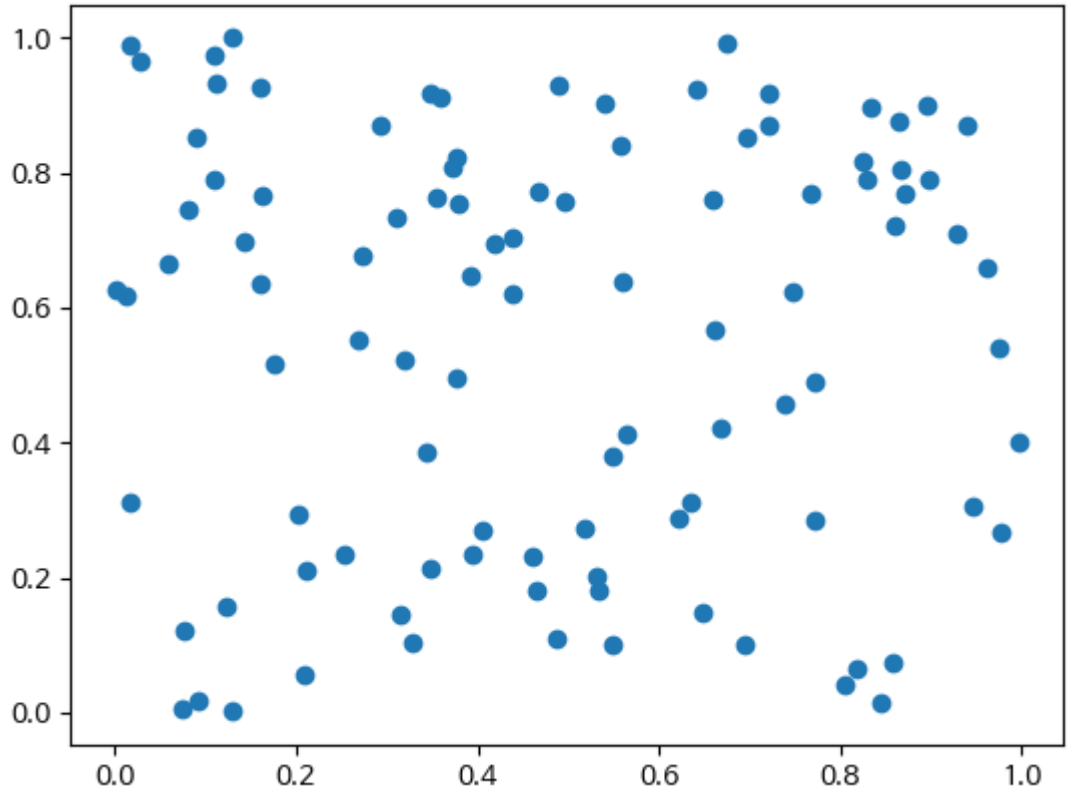
✓ 기본 산점도 그리기

```
np.random.seed(133)

x = np.random.rand(100)
y = np.random.rand(100)

plt.scatter(x, y)

plt.show()
```



히스토그램

- ✓ 연속 데이터의 빈도 분포를 표시
- ✓ 하나의 시리즈(x축)만을 가지고 그린다.
- ✓ x축의 범위는 기본적으로 최대값/최소값을 기준으로 총 10개 계급 구간을 동일하게 나누도록 정해진다.

plt.hist (x [,bins, alpha])

히스토그램

```
weight = [68, 81, 64, 56, 78, 74, 61, 77, 66, 68, 59, 71,
          80, 59, 67, 81, 69, 73, 69, 74, 70, 65]
plt.hist(weight)
plt.show()
```

구간	빈도수
56 ~ 58.5	1
58.5 ~ 61	2
61 ~ 63.5	1
63.5 ~ 66	2
66 ~ 68.5	4
68.5 ~ 71	3
71 ~ 73.5	2
73.5 ~ 76	2
76 ~ 78.5	2
78.5 ~81	3