

# 인공지능 활용 능력 개발 중급

## [2차시]

# 목 차

## 01. pandas 기초

- 데이터 구조 및 생성
- DATAFRAME 속성
- Indexing, Slicing
- DATAFRAME 조작
- DATAFRAME 메소드
- 연산/통계
- 병합
- Multi Index
- Groupby
- dataframe apply

## 02. 시각화

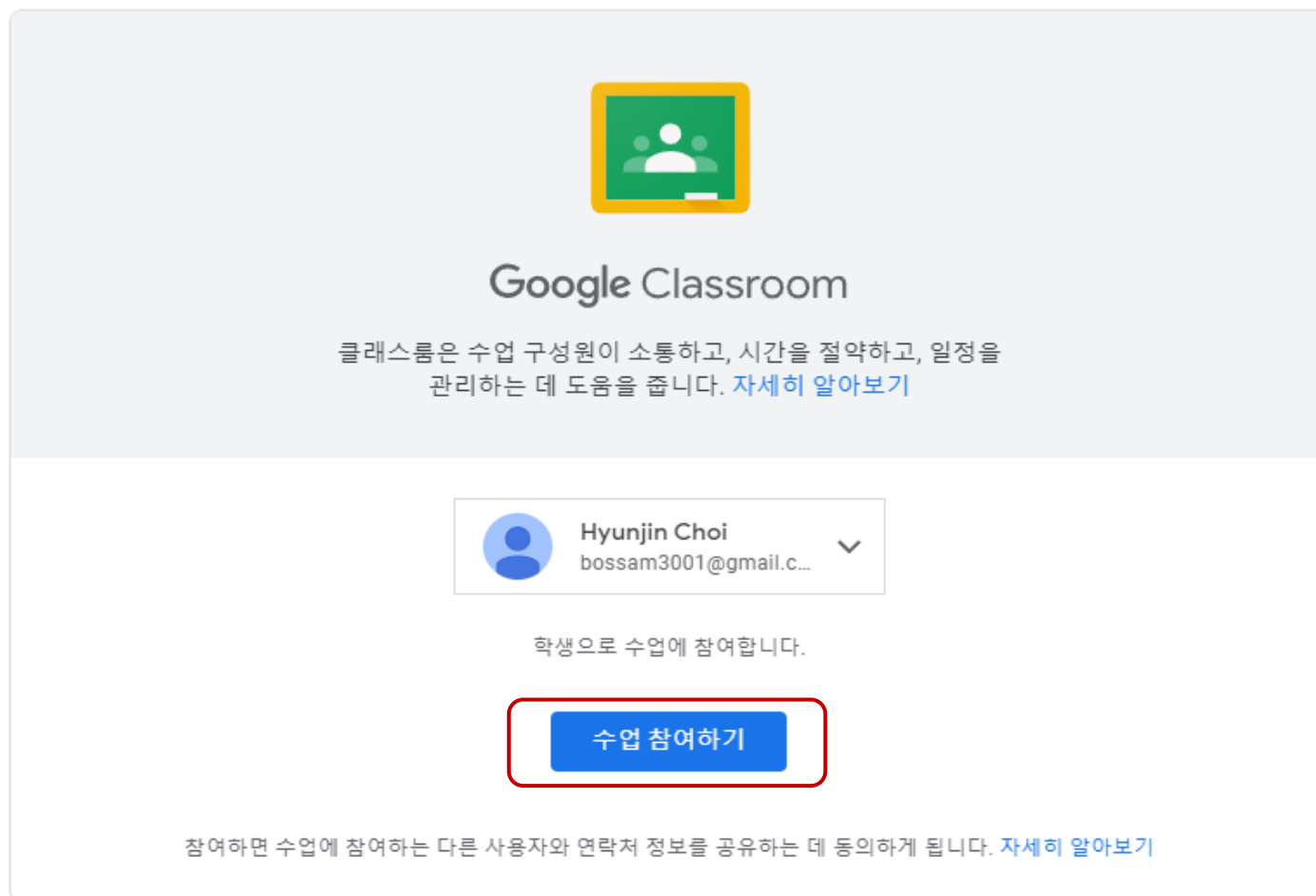
# 학습목표

---

- 인공지능 개발을 위한 기초 모듈인 Pandas 익히기
- 데이터 시각화 해보기

# 구글 클래스 룸

<https://classroom.google.com/c/NjE1NDg3NjA2ODI4?cjc=u2lk3cg>



# Github 자료

<https://github.com/aibiz00/intermediate>

The screenshot shows the GitHub repository page for 'aibiz00 / intermediate'. The repository is public and has 25 commits. The main branch is 'main'. The repository contains three files: 'README.md', '인공지능 활용 능력 개발 중급\_강의...', and '인공지능 활용 능력 개발 중급\_실습...'. The README.md file is open, showing the title 'intermediate' and the description '23년 DB 하이텍 인공지능 중급 교육 자료 입니다.' The right sidebar shows the repository's statistics: 23년 DB 하이텍 인공지능 중급 교육 자료 입니다., 0 stars, 1 watching, and 0 forks. The bottom of the page shows the GitHub footer with copyright information and links to Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Blog, and About.

aibiz00 / intermediate

intermediate Public

main 1 branch 0 tags

Go to file Add file Code

aibiz00 Delete 중급과정테스트\_답안지포함.zip 5655967 now 25 commits

README.md	Initial commit	2 weeks ago
인공지능 활용 능력 개발 중급_강의...	Add files via upload	2 weeks ago
인공지능 활용 능력 개발 중급_실습...	Add files via upload	2 weeks ago

README.md

## intermediate

23년 DB 하이텍 인공지능 중급 교육 자료 입니다.

About

23년 DB 하이텍 인공지능 중급 교육 자료 입니다.

Readme Activity 0 stars 1 watching 0 forks

Releases

No releases published  
[Create a new release](#)

Packages

No packages published  
[Publish your first package](#)

© 2023 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

## ndarray vs. list 구조

# Pandas 소개

---

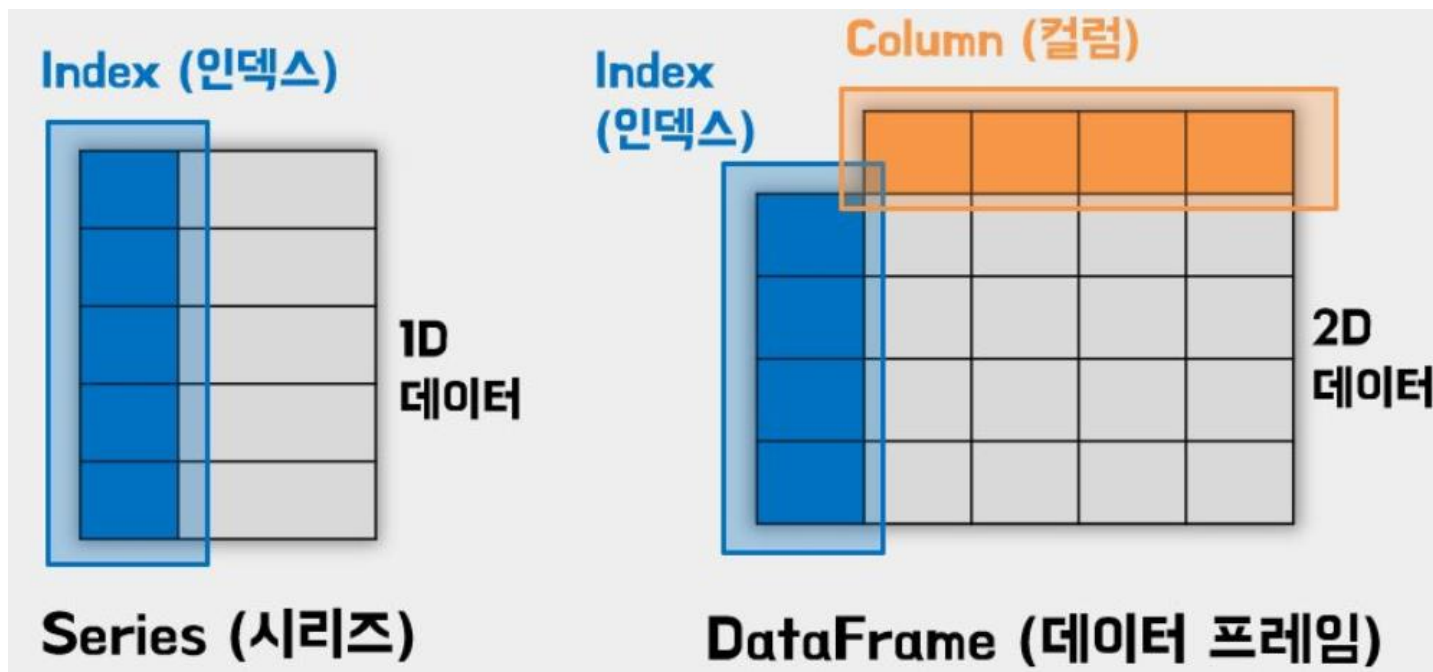
- 판다스(pandas)는 파이썬 언어로 작성된 데이터를 분석 및 조작하기 위한 소프트웨어 라이브러리
- 판데이터를 조작함에 있어 대표적인 라이브러리
- 데이터 프레임 조작에 특화

# 데이터 구조 및 생성

# Series 구조 : 1차원

## 데이터 종류

Index, Column(header)로 구성된 1차원 혹은 2차원 데이터 컨테이너

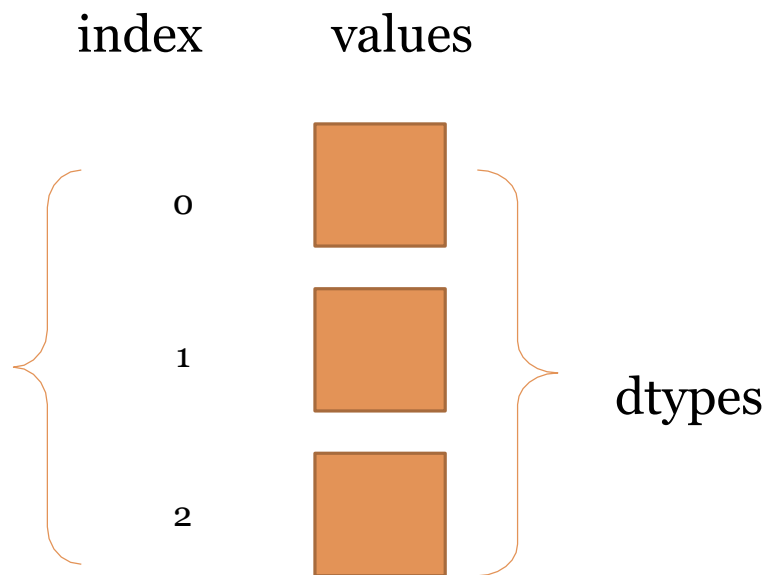




## Series 구조 : 1차원

## Series 구조 : 1차원

1차원의 데이터를 관리하는 컨테이너로 dict 타입처럼 index와 value가 항상 연계되어 처리



data: 실제 데이터 값

index : 데이터를 접근할 정보  
index.name으로 index도  
name을 지정할 수 있음

dtypes : 데이터들의 타입

name : Series 인스턴스의 이름

# Series 구조 생성

## Series 구조 생성

1차원의 데이터를 관리하는 컨테이너이며 index 등을 별도로 정의할 수 있음

```
import pandas as pd

s3 = pd.Series([4, 5, 6], index = ['a', 'b', 'c'], name = "Series Data")

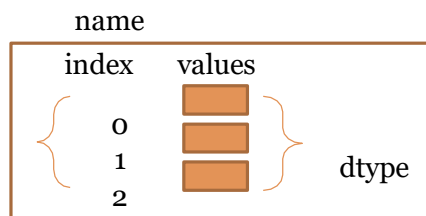
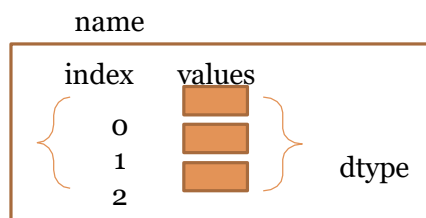
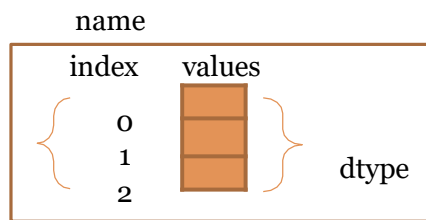
print(s3.index)
print(s3.values)
print(s3.name)
print(s3.dtype)
print(s3.ndim)
print(s3.shape)

Index(['a', 'b', 'c'], dtype='object')
[4 5 6]
Series Data
int64
1
(3,)
```

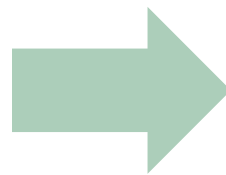
# DataFrame 구조: 2차원

## DataFrame 구조: 2차원

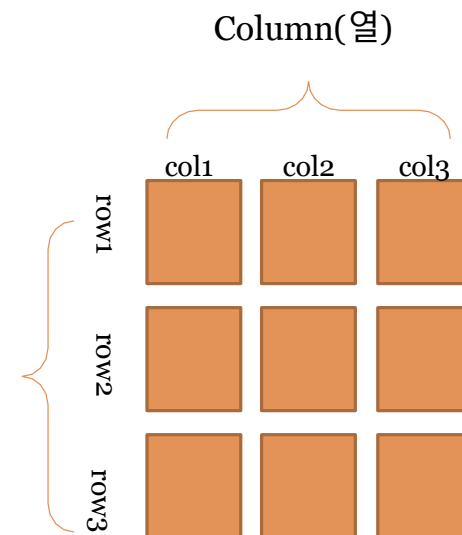
Series 인스턴스들이 DataFrame의 칼럼으로 들어가는 구조 columns는 series 명이 되어야 하고 index는 series의 index로 처리



Series에서  
DataFrame  
전환



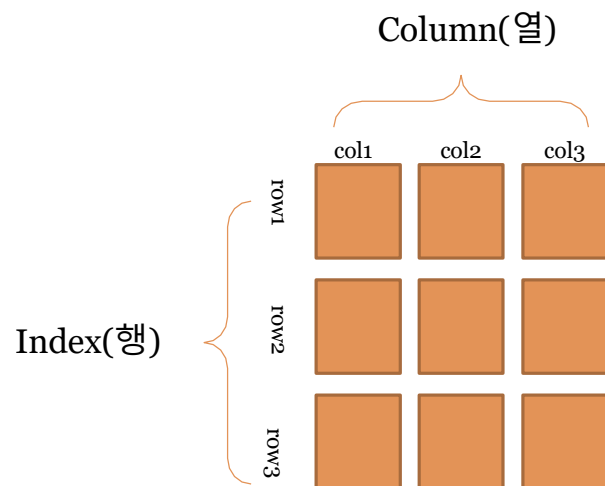
Index(행)



## DataFrame 생성

## DataFrame 생성

$n*m$  행렬구조를 가지는 데이터 구조이로 index와 column이 별도의 명을 가지고있음, column별로 다른 데이터 타입을 가질 수 있음



```
import pandas as pd

data = {'AAA' : [4, 5, 6, 7],
        'BBB' : [10, 20, 30, 40],
        'CCC' : [100, 50, -30, -50]}
df = pd.DataFrame(data,
                  index = ['a', 'b', 'c', 'd'],
                  columns = list(data.keys()))

print(df)
```

	AAA	BBB	CCC
a	4	10	100
b	5	20	50
c	6	30	-30
d	7	40	-50

# DataFrame 생성

$n*m$  행열구조를 가지는 데이터 구조 생성

```
import pandas as pd
```

```
help(pd.DataFrame)
```

Help on class DataFrame in module pandas.core.frame:

```
class DataFrame(pandas.core.generic.NDFrame, pandas.core.arraylike.OpsMixin)
| DataFrame(data=None, index: 'Axes | None' = None, columns: 'Axes | None' = None)
|
| Two-dimensional, size-mutable, potentially heterogeneous tabular data.
|
| Data structure also contains labeled axes (rows and columns).
| Arithmetic operations align on both row and column labels. Can be
| thought of as a dict-like container for Series objects. The primary
| pandas data structure.
|
| Parameters
| -----
| data : ndarray (structured or homogeneous), Iterable, dict, or DataFrame
|       Dict can contain Series, arrays, constants, dataclass or list-like objects.
|       If data is a dict, column order follows insertion-order. If a dict contains
|       keys that are not Series names, new columns will be created. If the keys
|       have an index defined, it is aligned by its index.
```

# DataFrame : arange 생성

n\*m 행열구조를 가지는 데이터 구조 생성

```
import numpy as np
import pandas as pd

obj1 = pd.DataFrame(data = np.arange(16).reshape(4, 4),
                    index = ['a', 'b', 'c', 'd'], columns = ['a', 'b', 'c', 'd'])

print(obj1.index)
print(obj1.columns)
print(obj1.values)
print(obj1.dtypes)

Index(['a', 'b', 'c', 'd'], dtype='object')
Index(['a', 'b', 'c', 'd'], dtype='object')
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
a    int64
b    int64
c    int64
d    int64
dtype: object
```

## Dataframe : dict 생성

data에 dict으로 칼럼정보를 넣고 index로 행에 대한 정보를 넣고 df 생성

```
import numpy as np
import pandas as pd

data = {'name' : ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'year' : [2012, 2012, 2013, 2014, 2014],
        'reports' : [4, 24, 31, 2, 3],
        'coverage' : [25, 94, 57, 62, 70]}
df = pd.DataFrame(data, index = ['Cochice', 'Pima', 'Santa Cruz', 'Maricopa', 'Yuma'])
print(df)
```

	name	year	reports	coverage
Cochice	Jason	2012	4	25
Pima	Molly	2012	24	94
Santa Cruz	Tina	2013	31	57
Maricopa	Jake	2014	2	62
Yuma	Amy	2014	3	70

```
print(obj1.index)
print(obj1.columns)
print(obj1.values)
print(obj1.dtypes)
```

# DataFrame 생성: list/tuple

column단위로 리스트를 만들어서 zip을 이용해서  
순서쌍을 만들고 데이터를 생성

```
import pandas as pd

names = ['Bob', 'Jessica', 'Mary', 'John', 'Mel']
births = [968, 155, 77, 578, 973]
BabyDataSet = list(zip(names, births))

df = pd.DataFrame(data = BabyDataSet, columns = ['Names', 'Births'])
print(df)
```

	Names	Births
0	Bob	968
1	Jessica	155
2	Mary	77
3	John	578
4	Mel	973



# DataFrame 생성: Series

두개의 series타입에서 키값을 추출해서 자동으로 인덱스화 해서 처리

```
import pandas as pd

area_dict = {'California' : 423967, 'Texas' : 695662, 'New York' : 141297, 'Florida' : 170312, 'Illinois' : 149995}
area = pd.Series(area_dict)

pop_dict = {'California' : 1423967, 'Texas' : 1695662, 'New York' : 1141297, 'Florida' : 1170312, 'Illinois' : 1149995}
population = pd.Series(pop_dict)

states = pd.DataFrame({'population' : population,
                      'area' : area})

print(states)
print(states.index)
print(states.values)
```

	population	area
California	1423967	423967
Texas	1695662	695662
New York	1141297	141297
Florida	1170312	170312
Illinois	1149995	149995

```
Index(['California', 'Texas', 'New York', 'Florida', 'Illinois'], dtype='object')
[[1423967  423967]
 [1695662  695662]
 [1141297  141297]
 [1170312  170312]
 [1149995  149995]]
```

# DataFrame 생성: 정리

리스트, 딕셔너리, array, series

```
import pandas as pd
#list 사용
data = [[1, 'Alice'], [2, 'Bob']]
df = pd.DataFrame(data, columns=['ID', 'Name'])
print(df.head())
#dict 사용
data = {'ID': [1, 2], 'Name': ['Alice', 'Bob']}
df = pd.DataFrame(data)
print(df.head())
# np 사용
data = np.array([[1, 'Alice'], [2, 'Bob']])
df = pd.DataFrame(data, columns=['ID', 'Name'])
print(df.head())
# series 사용
s1 = pd.Series([1, 2], name='ID')
s2 = pd.Series(['Alice', 'Bob'], name='Name')
df = pd.DataFrame([s1, s2]).T
print(df.head())
```

	ID	Name
0	1	Alice
1	2	Bob

	ID	Name
0	1	Alice
1	2	Bob

	ID	Name
0	1	Alice
1	2	Bob

# DataFrame 컬럼 추가: 컬럼 복사

Columns를 통해 컬럼명을 할당

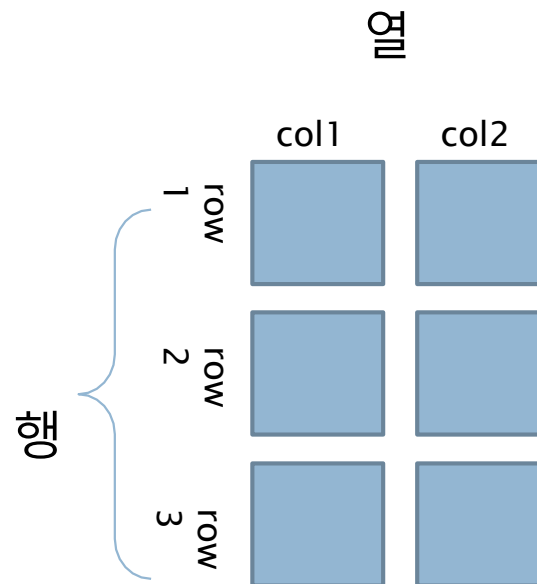
```
import pandas as pd

# Our small data set
d = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Create dataframe
df = pd.DataFrame(d)
df.columns = ['Rev']

# 두번째 컬럼명 및 컬럼값 추가
df['col'] = df['Rev']
print(df)
```

	Rev	col
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9



# DataFrame 행 이름 부여

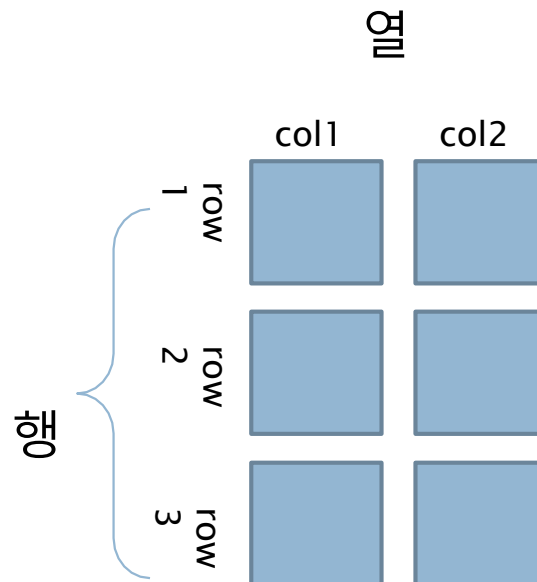
Index함수를 통해 Row명(index)을 부여 가능

```
import pandas as pd

# Our small data set
d = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Create dataframe
df = pd.DataFrame(d)
df.columns = ['Rev']
df['col'] = df['Rev']
i = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df.index = i
print(df)
```

	Rev	col
a	0	0
b	1	1
c	2	2
d	3	3
e	4	4
f	5	5
g	6	6
h	7	7
i	8	8
j	9	9



# DataFrame multiindex 생성

## DataFrame의 index를 multiindex로 정의 후 생성

```
import numpy as np
import pandas as pd

arrays = [['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],
          ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two']]

tuples = list(zip(*arrays))

index = pd.MultiIndex.from_tuples(tuples, names = ['first', 'second'])

df = pd.DataFrame(np.random.randn(3, 8), index = ['A', 'B', 'C'], columns = index)
print(df)
```

first	bar	baz	foo	qux
second	one	two	one	two
A	0.520314	-0.051819	2.436247	1.739253
B	-0.635057	-0.292082	-0.146771	0.139994
C	0.088214	1.163344	-0.223100	0.480989

first	second	two
A	-1.378966	
B	0.148600	
C	-1.180419	

## 연습문제

---

1. NumPy를 사용해 3x3 크기의 임의의 2차원 배열을 생성합니다. 배열의 원소는 1부터 9까지의 숫자입니다.
2. 그 다음, 이 NumPy 배열을 사용하여 Pandas DataFrame을 생성합니다. 각 열의 이름은 'Column1', 'Column2', 'Column3'으로 지정합니다.
3. 생성한 DataFrame을 출력합니다.
4. Df의 type, value, columns, index도 출력해보세요.

# 연습문제 코드

```
# 필요한 라이브러리를 불러옵니다.  
import numpy as np  
import pandas as pd  
  
# 1. 먼저, numpy를 사용해 3x3 크기의 2차원 배열을 생성합니다. 배열의 원소는 1부터 9까지의 숫자입니다.  
numpy_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
  
# 2. 그 다음, 이 numpy 배열을 사용하여 pandas DataFrame을 생성합니다. 각 열의 이름은 'Column1', 'Column2', 'Column3'으로 지정합니다.  
df = pd.DataFrame(numpy_array, columns = ['Column1', 'Column2', 'Column3'])  
  
# 3. 생성한 DataFrame을 출력합니다.  
print(df)
```

	Column1	Column2	Column3
0	1	2	3
1	4	5	6
2	7	8	9

# DATAFRAME 속성



# DataFrame 형태 조회 속성

변수	설명
shape	DataFrame의 행렬 형태를 표시
size	원소들의 갯수
ndim	차원에 대한 정보 표시
dtypes	행과 열에 대한 데이터 타입을 표시
index	행에 대한 접근 표시
columns	칼럼에 대한 접근 표시
axes	행과 열에 대한 축을 접근 표시
values	Numpy 로 변환
empty	DataFrame 내부가 없으면 True 원소가 있으면 False
T	행과 열을 변환

# DataFrame 기본 속성 확인

shape, size, ndim, dtypes 정보 조회

```
import pandas as pd

# DataFrame 생성
data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

# DataFrame 출력
print(df)

# DataFrame 속성들 출력
print("\nShape of DataFrame:", df.shape)
print("Size of DataFrame:", df.size)
print("Number of dimensions of DataFrame:", df.ndim)
print("Data types of DataFrame: \n", df.dtypes)
```

	Name	Age	City
0	John	28	New York
1	Anna	24	Paris
2	Peter	35	Berlin
3	Linda	32	London

```
Shape of DataFrame: (4, 3)
Size of DataFrame: 12
Number of dimensions of DataFrame: 2
Data types of DataFrame:
Name      object
Age       int64
City      object
dtype: object
```

# DataFrame 기본 속성 확인

index, columns, axes, values, empty, transposes  
대한 속성 값들을 확인

```
import pandas as pd

# DataFrame 생성
data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

# DataFrame 출력
print(df)

# DataFrame 속성들 출력
print("\nIndex of DataFrame: \n", df.index)
print("Columns of DataFrame: \n", df.columns)
print("Axes of DataFrame: \n", df.axes)
print("Values of DataFrame: \n", df.values)
print("Is DataFrame empty?", df.empty)
print("Transpose of DataFrame: \n", df.T)
```

	Name	Age	City
0	John	28	New York
1	Anna	24	Paris
2	Peter	35	Berlin
3	Linda	32	London

```
Index of DataFrame:
RangeIndex(start=0, stop=4, step=1)
Columns of DataFrame:
Index(['Name', 'Age', 'City'], dtype='object')
Axes of DataFrame:
[RangeIndex(start=0, stop=4, step=1), Index(['Name', 'Age', 'City'], dtype='object')]
Values of DataFrame:
[['John' 28 'New York']
 ['Anna' 24 'Paris']
 ['Peter' 35 'Berlin']
 ['Linda' 32 'London']]
Is DataFrame empty? False
Transpose of DataFrame:
      0      1      2      3
Name   John   Anna   Peter   Linda
Age      28      24      35      32
City New York Paris Berlin London
```

# Axes 축 이해

Axes(축)은 Index클래스를 가지고, index/columns에 대한 labels구성에 대한 축을 관리 (0: 행, 1: 열)

```
import numpy as np
import pandas as pd
```

```
df = pd.DataFrame(np.arange(16).reshape(4, 4), index = ['a', 'b', 'c', 'd'], columns = ['f', 'g', 'h', 'i'])
```

```
print(df)
```

```
print(" index ", df.axes[0])
```

```
print(" columns ", df.axes[1])
```

```

      f  g  h  i
a    0  1  2  3
b    4  5  6  7
c    8  9 10 11
d   12 13 14 15

```

```
index Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
columns Index(['f', 'g', 'h', 'i'], dtype='object')
```

Index는 행, columns는 열

## 연습문제

---

1. 이름, 나이, 도시를 포함한 데이터프레임을 생성하십시오. 이 데이터프레임은 최소 4명의 사람에 대한 정보를 포함해야 합니다.
2. 데이터프레임의 차원을 확인하십시오 (shape 사용).
3. 데이터프레임의 인덱스를 출력하십시오 (index 사용).
4. 데이터프레임의 열 이름을 출력하십시오 (columns 사용).
5. 데이터프레임의 값을 출력하십시오 (values 사용)

# 연습문제 코드

```
# 필요한 판다스 라이브러리를 불러옵니다.
import pandas as pd

# 1. 이름, 나이, 도시를 포함한 데이터프레임을 생성합니다. 이 데이터프레임은 최소 4명의 사람에 대한 정보를 포함해야 합니다.
data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

# 2. 데이터프레임의 차원을 확인합니다. 이것은 DataFrame의 차원을 나타냅니다.
print("Shape of the DataFrame:", df.shape)

# 3. 데이터프레임의 인덱스를 출력합니다.
print("\nIndex of the DataFrame:")
print(df.index)

# 4. 데이터프레임의 열 이름을 출력합니다.
print("\nColumns of the DataFrame:")
print(df.columns)

# 5. 데이터프레임의 값을 출력합니다. 이것은 DataFrame의 데이터를 numpy array 형태로 보여줍니다.
print("\nValues of the DataFrame:")
print(df.values)

Shape of the DataFrame: (4, 3)

Index of the DataFrame:
RangeIndex(start=0, stop=4, step=1)

Columns of the DataFrame:
Index(['Name', 'Age', 'City'], dtype='object')

Values of the DataFrame:
[['John' 28 'New York']
 ['Anna' 24 'Paris']
 ['Peter' 35 'Berlin']
 ['Linda' 32 'London']]
```

# Indexing, Slicing

# DataFrame indexing

1. [] 연산자를 사용한 열 인덱싱
2. loc를 사용한 레이블 기반 인덱싱
3. iloc를 사용한 정수 위치 기반 인덱싱

```
import pandas as pd

# DataFrame을 생성합니다.
data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

# '[]' 연산자를 사용하여 'Name'열을 인덱싱합니다.
print("Indexing by column name using []:")
print(df['Name'])

# '.loc'를 사용하여 레이블 기반으로 첫 번째 행을 인덱싱합니다.
print("\nLabel-based indexing using .loc:")
print(df.loc[0])

# '.iloc'를 사용하여 위치 기반으로 첫 번째 행을 인덱싱합니다.
print("\nInteger-based indexing using .iloc:")
print(df.iloc[0])
```

Indexing by column name using []:

```
0    John
1    Anna
2    Peter
3    Linda
```

Name: Name, dtype: object

Label-based indexing using .loc:

```
Name      John
Age        28
City    New York
```

Name: 0, dtype: object

Integer-based indexing using .iloc:

```
Name      John
Age        28
City    New York
```

Name: 0, dtype: object



# Indexing [ ] 처리 방법

데이터 타입별 인덱스 접근 방법

Object Type	Selection	Return Value Type
Series	series[label]	scalar value
DataFrame	frame[colname]	Series corresponding to col name

## [ ] 연산자 indexing

[] 연산자는 레이블로 열을 선택하거나 불리언 배열을 사용하여 행을 선택

```
import pandas as pd

data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

# 'Name' 열을 인덱싱합니다.
name = df['Name']
print("Name column:")
print(name)

# 'Age' 열을 인덱싱합니다.
age = df['Age']
print("\nAge column:")
print(age)

# Age가 30 이상인 행 masking 결과
print(df['Age'] > 30)

# Age가 30 이상인 행을 인덱싱합니다.
over_30 = df[df['Age'] > 30]
print("\nRows where Age > 30:")
print(over_30)
```

```
Name column:
0    John
1    Anna
2    Peter
3    Linda
Name: Name, dtype: object
```

```
Age column:
0    28
1    24
2    35
3    32
Name: Age, dtype: int64

0    False
1    False
2     True
3     True
Name: Age, dtype: bool
```

```
Rows where Age > 30:
   Name  Age  City
2  Peter   35 Berlin
3  Linda   32  London
```

# Indexing [ ] 처리 방법 : 논리식

DataFrame 내의 논리식을 표현하면 True 일 경우 출력됨

```
import numpy as np
import pandas as pd

obj1 = pd.DataFrame(data = np.arange(16).reshape(4, 4),
                    index = ['a', 'b', 'c', 'd'], columns = ['a', 'b', 'c', 'd'])

print(obj1['a'])
print(obj1['a'] > 5)
print(obj1[obj1['a'] > 5])
```

```
a    0
b    4
c    8
d   12
Name: a, dtype: int64
a    False
b    False
c     True
d     True
Name: a, dtype: bool
   a  b  c  d
c  8  9 10 11
d 12 13 14 15
```

# Indexing loc 처리 방법

데이터 타입별 인덱스 접근 방법

Object Type	Indexers
Series	<code>s.loc[indexer]</code>
DataFrame	<code>df.loc[row_index,column_index]</code>

# loc indexing

레이블을 기반으로 행과 열을 동시에 선택할 수 있어,  
보다 복잡한 인덱싱과 슬라이싱 가능

```
import pandas as pd

data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

# 첫 번째 행을 인덱싱합니다.
first_row = df.loc[0]
print("First row:")
print(first_row)

# 'Name' 열을 인덱싱합니다.
name_column = df.loc[:, 'Name']
print("\nName column:")
print(name_column)

# 두 번째 행의 'Age' 열을 인덱싱합니다.
specific_value = df.loc[1, 'Age']
print("\nSpecific value in second row, Age column:")
print(specific_value)

# Age가 30 이상인 행을 인덱싱합니다.
over_30 = df.loc[df['Age'] > 30]
print("\nRows where Age > 30")
print(over_30)
```

First row:

Name	John
Age	28
City	New York

Name: 0, dtype: object

Name column:

0	John
1	Anna
2	Peter
3	Linda

Name: Name, dtype: object

Specific value in second row, Age column:

24

Rows where Age > 30

	Name	Age	City
2	Peter	35	Berlin
3	Linda	32	London

# loc slicing

list, numpy slicing와 기본적인 slicing 문법은 동일

```
import pandas as pd

data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

# 첫 번째부터 세 번째 행을 슬라이싱합니다.
first_three_rows = df.loc[0:2]
print("First three rows:")
print(first_three_rows)

# 'Name'과 'Age' 열을 슬라이싱합니다.
name_and_age = df.loc[:, ['Name', 'Age']]
print("\nName and Age columns:")
print(name_and_age)

# 첫 번째부터 세 번째 행에서 'Name'과 'Age' 열을 슬라이싱합니다.
specific_slice = df.loc[0:2, ['Name', 'Age']]
print("\nSpecific slice - first three rows, Name and Age columns:")
print(specific_slice)
```

First three rows:

	Name	Age	City
0	John	28	New York
1	Anna	24	Paris
2	Peter	35	Berlin

Name and Age columns:

	Name	Age
0	John	28
1	Anna	24
2	Peter	35
3	Linda	32

Specific slice - first three rows, Name and Age columns:

	Name	Age
0	John	28
1	Anna	24
2	Peter	35

# loc slicing

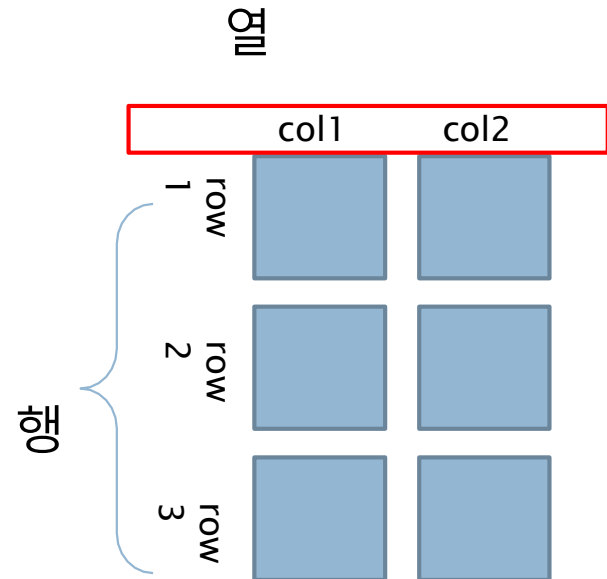
DataFrame은 멀티행을 슬라이싱 방식([ : ])을 사용하지만 이름으로 검색시에는 해당 이름까지 포함해서 처리

```
import pandas as pd

# Our small data set
d = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Create dataframe
df = pd.DataFrame(d)
df.columns = ['Rev']
df['col'] = df['Rev']
i = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df.index = i
print(df.loc['a' : 'c'])
```

	Rev	col
a	0	0
b	1	1
c	2	2



# DataFrame.iloc : indexing

1. 데이터프레임의 행이나 컬럼에 인덱스 값으로 접근.
2. integer location의 약어로, 컴퓨터가 읽을 수 있는 indexing 값으로 데이터에 접근하는 것이다

```
import pandas as pd

data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

# 첫 번째 행을 인덱싱합니다.
first_row = df.iloc[0]
print("First row:")
print(first_row)

# 세 번째 열을 인덱싱합니다.
third_column = df.iloc[:, 2]
print("\nThird column:")
print(third_column)

# 두 번째 행의 첫 번째 열을 인덱싱합니다.
specific_value = df.iloc[1, 0]
print("\nSpecific value:")
print(specific_value)
```

First row:

Name	John
Age	28
City	New York

Name: 0, dtype: object

Third column:

0	New York
1	Paris
2	Berlin
3	London

Name: City, dtype: object

Specific value:

Anna



# DataFrame.iloc : slicing

list, numpy slicing와 기본적인 slicing 문법은 동일

```
import pandas as pd

# DataFrame을 생성합니다.
data = {'Name' : ['John', 'Anna', 'Peter', 'Linda', 'Mike'],
        'Age' : [28, 24, 35, 32, 21],
        'City' : ['New York', 'Paris', 'Berlin', 'London', 'Sydney']}
df = pd.DataFrame(data)

# 첫 두 행을 슬라이싱합니다.
print("First two rows:")
print(df.iloc[0:2])

# 마지막 세 행을 슬라이싱합니다.
print("\nLast three rows:")
print(df.iloc[-3:])

# 두 번째와 세 번째 행을 슬라이싱합니다.
print("\nSecond and third rows:")
print(df.iloc[1:3])

# 두 번째와 세 번째 열을 슬라이싱합니다.
print("\nSecond and third columns:")
print(df.iloc[:, 1:3])

# 첫 두 행과 마지막 두 열을 슬라이싱합니다.
print("\nFirst two rows and last two columns:")
print(df.iloc[0:2, -2:])

# 데이터프레임의 중간 부분을 슬라이싱합니다 (2~4행, 1~2열)
print("\nA slice from the middle of the DataFrame:")
print(df.iloc[1:4, 0:2])
```

First two rows:

	Name	Age	City
0	John	28	New York
1	Anna	24	Paris

Last three rows:

	Name	Age	City
2	Peter	35	Berlin
3	Linda	32	London
4	Mike	21	Sydney

Second and third rows:

	Name	Age	City
1	Anna	24	Paris
2	Peter	35	Berlin

Second and third columns:

	Age	City
0	28	New York
1	24	Paris
2	35	Berlin
3	32	London
4	21	Sydney

First two rows and last two columns:

	Age	City
0	28	New York
1	24	Paris

A slice from the middle of the DataFrame:

	Name	Age
1	Anna	24
2	Peter	35
3	Linda	32

## 연습문제

---

1. `iloc`를 사용하여 처음부터 10행까지 그리고 'A', 'B' 열을 선택하세요.
2. 그 후에, 'A'열에서 값이 50보다 큰 모든 행을 선택하여 출력하세요.

아래의 데이터프레임을 사용하여 시작하세요:

```
import numpy as np
import pandas as pd

np.random.seed(0)
df = pd.DataFrame(np.random.randint(0, 100, size = (100, 4)),
                  columns = list('ABCD'))
```

## 연습문제 코드

```
import numpy as np
import pandas as pd

# Seed for reproducibility
np.random.seed(0)

# 초기 데이터프레임 생성
df = pd.DataFrame(np.random.randint(0, 100, size = (100, 4)), columns = list('ABCD'))

print("Original DataFrame:")
print(df)

#.iloc를 사용하여 처음 10행과 'A', 'B' 열을 선택
df_iloc = df.iloc[0:10, 0:2]

print("\nDataFrame after slicing with iloc:")
print(df_iloc)

# 'A'열에서 값이 50보다 큰 모든 행을 선택
df_mask = df[df['A'] > 50]

print("\nDataFrame after boolean masking:")
print(df_mask)
```

## 연습문제

1. 'Name'이 'Charlie'인 행의 'City' 값을 찾으세요.
2. 'Age'가 30 이상인 사람들의 'Name'과 'City' 정보만을 추출하세요.
3. [] 연산자 사용하여 Name데이터만 추출하세요
4. Iloc을 사용하여 마지막 두 행의 'Age'와 'City' 정보를 추출하세요

아래의 데이터프레임을 사용하여 시작하세요

```
data = {  
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],  
    'Age': [25, 30, 35, 40],  
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']  
}
```

# 연습문제

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}

# 'Name'이 'Charlie'인 행의 'City' 값을 찾으세요.
df = pd.DataFrame(data)
result = df.loc[df['Name'] == 'Charlie', 'City']
print(result)

# 'Age'가 30 이상인 사람들의 'Name'과 'City' 정보만을 추출하세요.
result = df.loc[df['Age'] >= 30, ['Name', 'City']]
print(result)

# [] 연산자 사용하여 Name데이터만 추출하세요
result = df['Name']
print(result)

# 마지막 두 행의 'Age'와 'City' 정보를 추출하세요.
result = df.iloc[-2:, 1:]
print(result)
```

```
2    Chicago
Name: City, dtype: object
      Name      City
1    Bob  Los Angeles
2  Charlie    Chicago
3    David    Houston
0    Alice
1    Bob
2    Charlie
3    David
Name: Name, dtype: object
      Age      City
2    35  Chicago
3    40  Houston
```

# Dataframe 조작

# column 추가

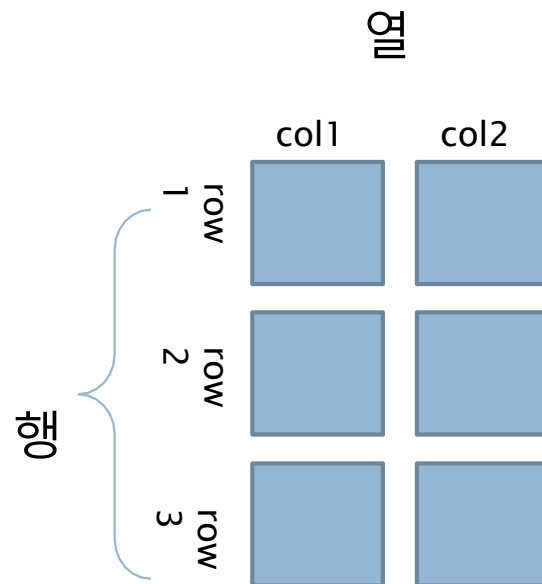
DataFrame은 기존에 없는 column에 값을 scala로 할당시 행에 맞춰 Broadcasting처리

```
import pandas as pd

# Our small data set
d = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Create dataframe
df = pd.DataFrame(d)
df.columns = ['Rev']
df['NewCol'] = 5
print(df)
```

	Rev	NewCol
0	0	5
1	1	5
2	2	5
3	3	5
4	4	5
5	5	5
6	6	5
7	7	5
8	8	5
9	9	5



# column 삭제 : del

기존에 존재한 column을 del로 삭제 가능

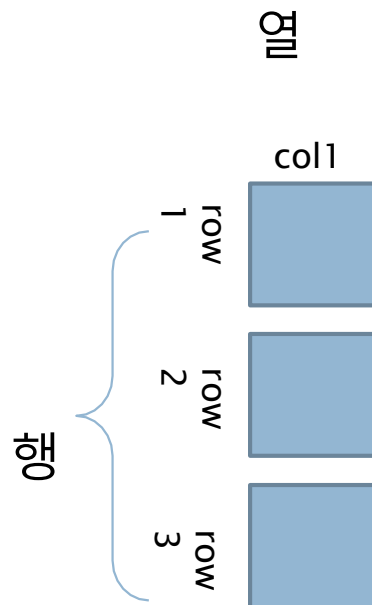
```
import pandas as pd

# Our small data set
d = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Create dataframe
df = pd.DataFrame(d)
df.columns = ['Rev']
df['NewCol'] = 5

del df['NewCol']
print(df)
```

	Rev
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9





# column 삭제 : drop

## 기존에 존재한 column을 drop으로 삭제 가능

```
import pandas as pd

help(pd.DataFrame.drop)
```

Help on function drop in module pandas.core.frame:

drop(self, labels: 'IndexLabel' = None, \*, axis: 'Axis' = 0, index  
Drop specified labels from rows or columns.

Remove rows or columns by specifying label names and correspon  
axis, or by specifying directly index or column names. When us  
multi-index, labels on different levels can be removed by spec  
the level. See the `user guide <advanced.shown\_levels>`  
for more information about the now unused levels.

Parameters

-----  
labels : single label or list-like  
Index or column labels to drop. A tuple will be used as a  
label and not treated as a list-like.  
axis : {0 or 'index', 1 or 'columns'}, default 0  
Whether to drop labels from the index (0 or 'index') or  
columns (1 or 'columns').

```
import pandas as pd

# Our small data set
d = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Create dataframe
df = pd.DataFrame(d)
df.columns = ['Rev']
df['NewCol'] = 5

df1 = df.drop("NewCol", axis = 1)
print(df1)
```

	Rev
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

# column 값 변경

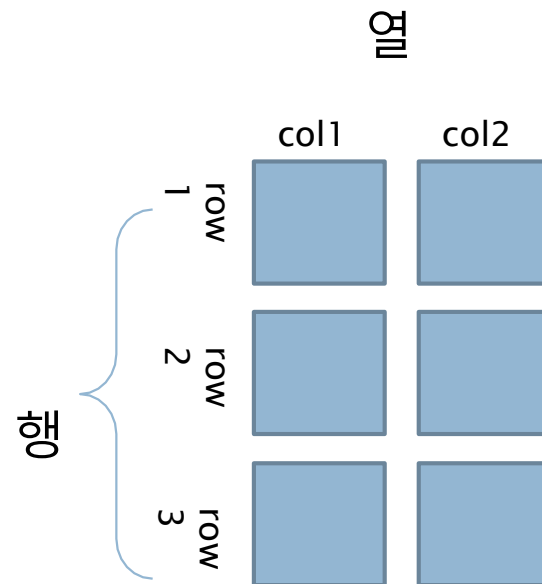
DataFrame은 기존에 존재한 column에 값을 추가할 경우 broadcasting되어 칼럼이 변경

```
import pandas as pd

# Our small data set
d = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Create dataframe
df = pd.DataFrame(d)
df.columns = ['Rev']
df['NewCol'] = 5
df['NewCol'] = df['NewCol'] + 1
print(df)
```

	Rev	NewCol
0	0	6
1	1	6
2	2	6
3	3	6
4	4	6
5	5	6
6	6	6
7	7	6
8	8	6
9	9	6



# column 명 변경

column명은 columns로 초기화 하거나 rename으로 특정 column명만 변경

```
import pandas as pd

data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# 특정 Column 명 변경 (방법 1)
df.rename(columns = {'Name' : 'First Name', 'City' : 'Hometown'}, inplace = True)

print("\nDataFrame after renaming some columns (using rename()):")
print(df)

# 모든 Column 명 변경 (방법 2)
df.columns = ['First Name', 'Age', 'Hometown']

print("\nDataFrame after renaming all columns (using columns):")
print(df)

df.columns = ['성' if x=='First Name' else x for x in df.columns]

print("\nDataFrame after renaming all columns (using columns):")
print(df)
```

Original DataFrame:

	Name	Age	City
0	John	28	New York
1	Anna	24	Paris
2	Peter	35	Berlin
3	Linda	32	London

DataFrame after renaming some columns (using rename()):

	First Name	Age	Hometown
0	John	28	New York
1	Anna	24	Paris
2	Peter	35	Berlin
3	Linda	32	London

DataFrame after renaming all columns (using columns):

	First Name	Age	Hometown
0	John	28	New York
1	Anna	24	Paris
2	Peter	35	Berlin
3	Linda	32	London

DataFrame after renaming all columns (using columns):

	성	Age	Hometown
0	John	28	New York
1	Anna	24	Paris
2	Peter	35	Berlin
3	Linda	32	London

# swap처리

칼럼별 swap 처리를 위해 리스트에 칼럼명을 사용해서 처리

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])

print(df)
df[['f', 'g']] = df[['g', 'f']]
print(df)
```

	f	g	h	i
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

	f	g	h	i
a	1	0	2	3
b	5	4	6	7
c	9	8	10	11
d	13	12	14	15

# Index

행과 열에 들어갈 index 대한 메타데이터 객체화하는 클래스

```
import pandas as pd
```

```
help(pd.Index)
```

Help on class Index in module pandas.core.indexes.base:

```
class Index(pandas.core.base.IndexOpsMixin, pandas.core.base.PandasObject)
|   Index(data=None, dtype=None, copy=False, name=None, tupleize_cols=True, **kwargs) -> 'Index'
|
|   Immutable sequence used for indexing and alignment.
|
|   The basic object storing axis labels for all pandas objects.
|
|   Parameters
|   -----
|   data : array-like (1-dimensional)
|   dtype : NumPy dtype (default: object)
|       If dtype is None, we find the dtype that best fits the data.
|       If an actual dtype is provided, we coerce to that dtype if it's safe.
|       Otherwise, an error will be raised.
|   copy : bool
|       Make a copy of input ndarray.
|   name : object
|       Name to be stored in the index.
|   tupleize_cols : bool (default: True)
|       When True, attempt to create a MultiIndex if possible.
```

# Index 생성하기: int

## Integer를 기반으로 Index 생성하기

```
import pandas as pd

idx1 = pd.Index([1, 2, 3, 4])
idx2 = pd.Index([3, 4, 5, 6])
print(idx1.difference(idx2))
print(idx1, idx2)
idx3 = pd.Index([1, 2], dtype = 'int64')
print(idx3)

Int64Index([1, 2], dtype='int64')
Int64Index([1, 2, 3, 4], dtype='int64') Int64Index([3, 4, 5, 6], dtype='int64')
Int64Index([1, 2], dtype='int64')
```

# Index 생성하기: str

---

## String를 기반으로 Index 생성하기

```
import pandas as pd

idx4 = pd.Index(['a', 'b', 'c'])
print(idx4)

Index(['a', 'b', 'c'], dtype='object')
```

# Index 생성하기: DateTime

## DateTime를 기반으로 Index 생성하기

```
import pandas as pd
```

```
idx5 = pd.Index(pd.date_range('20130101', periods = 3))
```

```
print(idx5)
```

```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03'], dtype='datetime64[ns]', freq='D')
```



# DataFrame에 index 적용

## DataFrame 생성전 직접 Index를 만들고 반영

```
import pandas as pd

inx = pd.Index(['a', 'b', 'c', 'd'])
iny = pd.Index(['A', 'B'])
data = {'A': [1, 2, 3, 4],
        'B': [5, 6, 7, 8]}

df = pd.DataFrame(data, index = inx, columns = iny)
print(df.values)
print(df.index)
print(df.columns)
print(df.describe())
```

```
[[1 5]
 [2 6]
 [3 7]
 [4 8]]
Index(['a', 'b', 'c', 'd'], dtype='object')
Index(['A', 'B'], dtype='object')
      A      B
count  4.000000  4.000000
mean   2.500000  6.500000
std    1.290994  1.290994
min    1.000000  5.000000
25%    1.750000  5.750000
50%    2.500000  6.500000
75%    3.250000  7.250000
max    4.000000  8.000000
```

# index 초기화

## DataFrame 인덱스를 default(integer 형식)으로 초기화

```
import pandas as pd

data = {'Name' : ['John', 'Anna', 'Peter', 'Linda'],
        'Age' : [28, 24, 35, 32],
        'City' : ['New York', 'Paris', 'Berlin', 'London']}
df = pd.DataFrame(data)

# 임의의 인덱스를 설정합니다.
df.index = ['a', 'b', 'c', 'd']

print("Before reset_index:")
print(df)

# 인덱스를 초기화합니다.
df_reset = df.reset_index()

print("\nAfter reset_index:")
print(df_reset)
```

Before reset\_index:

	Name	Age	City
a	John	28	New York
b	Anna	24	Paris
c	Peter	35	Berlin
d	Linda	32	London

After reset\_index:

	index	Name	Age	City
0	a	John	28	New York
1	b	Anna	24	Paris
2	c	Peter	35	Berlin
3	d	Linda	32	London

# index class 속성

Index class 속성	description
values	인덱스의 값
name	인덱스의 이름
nlevels	인덱스의 레벨 수 (MultiIndex의 경우 사용)
ndim	차원
dtype	데이터 타입
hasnans	인덱스에 NaN 값이 있는지 여부
has_duplicates	인덱스에 중복 값이 있는지 여부

# 속성을 활용한 index 확인

## Index 내의 분류 및 중복여부 확인

```
import pandas as pd

# 임의의 인덱스 설정
df.index = ['a', 'b', 'c', 'd']

# Index 객체 가져오기
index = df.index

print("Index values:", index.values)    # 인덱스의 값
print("Index name:", index.name)        # 인덱스의 이름
print("Index nlevels:", index.nlevels)  # 인덱스의 레벨 수 (MultiIndex의 경우 사용)
print("Index ndim:", index.ndim)        # 인덱스의 차원
print("Index dtype:", index.dtype)      # 인덱스의 데이터 타입
print("Index hasnans:", index.hasnans)  # 인덱스에 NaN 값이 있는지 여부
print("Index has_duplicates:", index.has_duplicates) # 인덱스에 중복 값이 있는지 여부

Index values: ['a' 'b' 'c' 'd']
Index name: None
Index nlevels: 1
Index ndim: 1
Index dtype: object
Index hasnans: False
Index has_duplicates: False
```

## 연습문제

---

1. 데이터프레임의 'A'와 'B' column을 swap하세요.
2. 데이터프레임의 index를 'x', 'y', 'z', 'k'로 변경하세요.
3. D column 을 데이터프레임에서 삭제하세요.

가장 처음 시작할 때, 아래의 데이터프레임을 사용하세요:

```
df = pd.DataFrame({'A' : ['foo', 'bar', 'baz', 'qux'],  
                   'B' : ['one', 'one', 'two', 'three'],  
                   'C' : np.random.randn(4),  
                   'D' : np.random.randn(4)})
```

## 연습문제 코드

```
import numpy as np
import pandas as pd

# 초기 데이터프레임 생성
df = pd.DataFrame({'A' : ['foo', 'bar', 'baz', 'qux'],
                   'B' : ['one', 'one', 'two', 'three'],
                   'C' : np.random.randn(4),
                   'D' : np.random.randn(4)})

print("Original DataFrame:")
print(df)

# 데이터프레임의 'A'와 'B'행을 swap
# df = df.reindex(['B', 'A', 'C', 'D'], axis = 1)
df[['A', 'B']] = df[['B', 'A']]
print("\nDataFrame after swapping 'A' and 'B':")
print(df)

# 데이터프레임의 index를 변경
df.index = ['x', 'y', 'z', 'k']
print("\nDataFrame after changing index:")
print(df)

# 'D' 열을 데이터프레임에서 삭제
df = df.drop('D', axis = 1)
print("\nDataFrame after deleting 'D':")
print(df)
```

# DATAFRAME 메소드

# 타입 변환 후 생성 : astype

타입을 변경해서 다른 dataframe 생성

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'], columns = ['f', 'g', 'h', 'i'])
print(df)

df1 = df.astype(np.float64)
print(df1)
print(df1['f'])
```

	f	g	h	i
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

	f	g	h	i
a	0.0	1.0	2.0	3.0
b	4.0	5.0	6.0	7.0
c	8.0	9.0	10.0	11.0
d	12.0	13.0	14.0	15.0

a	0.0
b	4.0
c	8.0
d	12.0

Name: f, dtype: float64



## 카피 후 생성 : copy

copy 메소드를 이용해서 생성하면 다른 인스턴스가 생성되지만 값을 비교(==)와 인스턴스비교(is)는 다른 결과가 나옴

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])

print(df)

df0 = df.copy()
print(df0)
print(id(df), id(df0))
print((df0 == df).all())
print(df0 is df)
```

```
      f  g  h  i
a    0  1  2  3
b    4  5  6  7
c    8  9 10 11
d   12 13 14 15
      f  g  h  i
a    0  1  2  3
b    4  5  6  7
c    8  9 10 11
d   12 13 14 15
140213730078592 140213730074704
f      True
g      True
h      True
i      True
dtype: bool
False
```

## Iterable 처리: iterrows

Dataframe을 iterable 하게 처리하면 행명과 행값들의 쌍(index, Series)으로 조회

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])
print(df.iterrows())
for row, val in df.iterrows():
    print(row, val)
```

```
<generator object DataFrame.iterrows at 0x7f85dd8a3d80>
a f    0
g    1
h    2
i    3
Name: a, dtype: int64
b f    4
g    5
h    6
i    7
Name: b, dtype: int64
c f    8
g    9
h   10
i   11
Name: c, dtype: int64
d f   12
g   13
h   14
i   15
Name: d, dtype: int64
```

# 칼럼 삽입: insert

## Insert 메소드를 이용해서 새로운 칼럼을 삽입

```
import pandas as pd
```

```
help(pd.DataFrame.insert)
```

Help on function insert in module pandas.core.frame:

```
insert(self, loc: 'int', column: 'Hashable', value: 'Scalar |
Insert column into DataFrame at specified location.
```

Raises a ValueError if `column` is already contained in the DataFrame unless `allow\_duplicates` is set to True.

Parameters

-----

loc : int

Insertion index. Must verify  $0 \leq \text{loc} \leq \text{len}(\text{columns})$

column : str, number, or hashable object

Label of the inserted column.

value : Scalar, Series, or array-like

allow\_duplicates : bool, optional, default lib.no\_default

```
import numpy as np
import pandas as pd
```

```
df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'], columns = ['f', 'g', 'h', 'i'])
```

```
print(df)
```

```
df.insert(4, 'j', [99, 999, 999, 9999])
```

```
print(df)
```

	f	g	h	i
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

	f	g	h	i	j
a	0	1	2	3	99
b	4	5	6	7	999
c	8	9	10	11	999
d	12	13	14	15	9999

# 칼럼 삭제: pop

## Pop 메소드를 이용해서 칼럼을 꺼낸 후 삭제하기

```
import pandas as pd

help(pd.DataFrame.pop)
```

Help on function pop in module pandas.core.frame:

pop(self, item: 'Hashable') -> 'Series'  
Return item and drop from frame. Raise KeyError if not found.

Parameters

-----

item : label  
Label of column to be popped.

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'], columns = ['f', 'g', 'h', 'i'])
print(df)
df.insert(4, 'j', [99, 999, 999, 9999])
print(df)

df.pop('j')
print(df)
```

	f	g	h	i
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

	f	g	h	i	j
a	0	1	2	3	99
b	4	5	6	7	999
c	8	9	10	11	999
d	12	13	14	15	9999

	f	g	h	i
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

# isnull/notnull

Null 여부 체크하여 boolean 표현으로 확인

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])

print(df)
df2 = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print(df2)
print(df2['f'].isnull())
print(df2['f'].notnull())
```

	f	g	h	i
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

	f	g	h	i
a	0.0	1.0	2.0	3.0
b	4.0	5.0	6.0	7.0
c	8.0	9.0	10.0	11.0
d	12.0	13.0	14.0	15.0
e	NaN	NaN	NaN	NaN
f	NaN	NaN	NaN	NaN
g	NaN	NaN	NaN	NaN
h	NaN	NaN	NaN	NaN

a	False
b	False
c	False
d	False
e	True
f	True
g	True
h	True

Name: f, dtype: bool

a	True
b	True
c	True
d	True
e	False
f	False
g	False
h	False

Name: f, dtype: bool

# Replace : 원소 한 개 변경

## DataFrame 내의 원소를 검색한 후에 대치시킴

```
import numpy as np
import pandas as pd

obj1 = pd.DataFrame(data = np.arange(16).reshape(4, 4),
                    index = ['a', 'b', 'c', 'd'], columns = ['a', 'b', 'c', 'd'])

obj1.replace(to_replace = 0, value = 999, inplace = True)
print(obj1)
print(obj1.replace(to_replace = 2, value = 888, inplace = True))
print(obj1)
print(obj1['d'].replace(3, 777, inplace = True))
print(obj1)
```

```
   a  b  c  d
a  999  1  2  3
b   4  5  6  7
c   8  9 10 11
d  12 13 14 15
None
```

```
   a  b  c  d
a  999  1 888  3
b   4  5  6  7
c   8  9 10 11
d  12 13 14 15
None
```

```
   a  b  c  d
a  999  1 888 777
b   4  5  6  7
c   8  9 10 11
d  12 13 14 15
```

# Replace : 원소 여러 개 변경

## DataFrame 내의 원소를 검색한 후에 대치시킴

```
import numpy as np
import pandas as pd

obj1 = pd.DataFrame(data = np.arange(16).reshape(4, 4),
                    index = ['a', 'b', 'c', 'd'], columns = ['a', 'b', 'c', 'd'])
obj1.replace(to_replace = (0, 1), value = 999, inplace = True)
print(obj1)
obj1.replace(to_replace = [3, 4, 5], value = 888, inplace = True)
print(obj1)
obj1.replace((10, 11), 777, inplace = True)
print(obj1)
```

	a	b	c	d
a	999	999	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

	a	b	c	d
a	999	999	2	888
b	888	888	6	7
c	8	9	10	11
d	12	13	14	15

	a	b	c	d
a	999	999	2	888
b	888	888	6	7
c	8	9	777	777
d	12	13	14	15

# DataFrame sort\_value

DataFrame 내의 원소에 대한 sorting 처리.  
Inplace를 통해 기존 dataframe을 변환

```
import numpy as np
import pandas as pd

data = np.random.permutation(np.arange(16))

obj1 = pd.DataFrame(data = data.reshape(4, 4),
                    index = ['ar', 'br', 'cr', 'dr'], columns = ['a', 'b', 'c', 'd'])

print(obj1)
obj1.sort_values('a', inplace = True)
print(obj1)
```

	a	b	c	d
ar	12	3	4	2
br	1	6	5	13
cr	7	11	14	10
dr	9	8	0	15
	a	b	c	d
br	1	6	5	13
cr	7	11	14	10
dr	9	8	0	15
ar	12	3	4	2



# DataFrame head 검색

default=5까지 처음 5열을 검색

```
import pandas as pd

help(pd.DataFrame.head)
```

Help on function head in module pandas.core.generic:

```
head(self: 'NDFrameT', n: 'int' = 5) -> 'NDFrameT'
    Return the first `n` rows.
```

This function returns the first `n` rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

For negative values of `n`, this function returns all rows except the last `|n|` rows, equivalent to ``df[:n]``.

If `n` is larger than the number of rows, this function returns

```
import pandas as pd

# Our small data set
d = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Create dataframe
df = pd.DataFrame(d)
df.columns = ['Rev']
df['col'] = df['Rev']
i = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df.index = i
print(df.head())
```

	Rev	col
a	0	0
b	1	1
c	2	2
d	3	3
e	4	4

# DataFrame tail 검색

default=5까지 끝에서 5열을 검색

```
import pandas as pd

help(pd.DataFrame.tail)
```

Help on function tail in module pandas.core.generic:

```
tail(self: 'NDFrameT', n: 'int' = 5) -> 'NDFrameT'
    Return the last `n` rows.
```

This function returns last `n` rows from the object bas position. It is useful for quickly verifying data, for after sorting or appending rows.

For negative values of `n`, this function returns all r the first `|n|` rows, equivalent to ``df[|n|:]``.

If n is larger than the number of rows, this function r

```
import pandas as pd

# Our small data set
d = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Create dataframe
df = pd.DataFrame(d)
df.columns = ['Rev']
df['col'] = df['Rev']
i = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df.index = i
print(df.tail())
```

	Rev	col
f	5	5
g	6	6
h	7	7
i	8	8
j	9	9

## 연습문제

---

1. 주어진 iterable을 데이터프레임으로 변환하세요.
2. `replace` 함수를 사용하여 데이터프레임 내의 특정 값들을 대체하세요.  
2 -> 20, 3 -> 30
3. `sort_values` 함수를 사용하여 C 열을 기준으로 데이터를 정렬하세요.
4. `iterrows` 함수를 사용하여 3번째로 큰 행을 찾아 출력하세요.

```
data_iterable = [('A', [1, 2, 3, 4]), ('B', [4, 3, 2, 1]), ('C', [6, 7, 8, 10])]
```

## 연습문제 코드

```
import pandas as pd

# 주어진 iterable을 데이터프레임으로 변환
data_iterable = [('A', [1, 2, 3, 4]), ('B', [4, 3, 2, 1]), ('C', [6, 7, 8, 10])]
df = pd.DataFrame(dict(data_iterable))

# replace 함수를 사용하여 데이터프레임 내의 특정 값들을 대체
df['A'] = df['A'].replace(2, 20)
df['B'] = df['B'].replace(3, 30)

# sort_values 함수를 사용하여 'C'열을 기준으로 데이터를 정렬
df = df.sort_values('C', ascending = False)

# 'C'열에서 3번째로 큰 값을 가진 데이터를 출력
third_largest_value = df['C'].unique()[2]

for index, row in df.iterrows():
    if row['C'] == third_largest_value:
        print(row)
        break
```

```
A    20
B    30
C     7
Name: 1, dtype: int64
```

# 연산/통계

# 산술연산자 이용

산술 연산자, 산술 메소드	description
<b>+ , add()</b>	덧셈을 수행합니다.
<b>- , sub()</b>	뺄셈을 수행합니다.
<b>* , mul()</b>	곱셈을 수행합니다.
<b>/ , div()</b>	나눗셈을 수행합니다.
<b>// , floordiv()</b>	나눗셈의 몫을 반환합니다.
<b>% , mod()</b>	나눗셈의 나머지를 반환합니다.
<b>** , pow()</b>	제곱을 계산합니다.

# 산술연산자 이용

## 산술연산자를 이용한 dataframe간 연산

```
import pandas as pd

# 예제 DataFrame 생성
data1 = {'A': [1, 2, 3],
         'B': [4, 5, 6],
         'C': [7, 8, 9]}
data2 = {'A': [10, 11, 12],
         'B': [13, 14, 15],
         'C': [16, 17, 18]}

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
print("df1:")
print(df1)
print("#df2:")
print(df2)

# 덧셈 연산을 수행합니다.
addition = df1 + df2
subtraction = df1 - df2
multiplication = df1 * df2
division = df1 / df2
modulus = df1 % df2
floor_division = df1 // df2
exponentiation = df1 ** df2

print(addition)
print(subtraction)
print(multiplication)
print(division)
print(modulus)
print(floor_division)
print(exponentiation)
```

```
df1:
   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

df2:
   A  B  C
0 10 13 16
1 11 14 17
2 12 15 18

   A  B  C
0 11 17 23
1 13 19 25
2 15 21 27

   A  B  C
0 -9 -9 -9
1 -9 -9 -9
2 -9 -9 -9

   A  B  C
0 10 52 112
1 22 70 136
2 36 90 162

   A      B      C
0 0.100000 0.307692 0.437500
1 0.181818 0.357143 0.470588
2 0.250000 0.400000 0.500000

   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

   A  B  C
0  0  0  0
1  0  0  0
2  0  0  0

   A      B      C
0      1      67108864      33232930569601
1    2048      6103515625      2251799813685248
2  531441  470184984576  150094635296999121
```

# 산술연산자 이용

## 산술메소드를 이용한 dataframe간 연산

```
import pandas as pd

# 예제 DataFrame 생성
data1 = {'A' : [1, 2, 3],
         'B' : [4, 5, 6],
         'C' : [7, 8, 9]}
data2 = {'A' : [10, 11, 12],
         'B' : [13, 14, 15],
         'C' : [16, 17, 18]}

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
print("df1:")
print(df1)
print("#df2:")
print(df2)

# 덧셈 연산을 수행합니다.
addition = df1.add(df2)
subtraction = df1.sub(df2)
multiplication = df1.mul(df2)
division = df1.div(df2)
modulus = df1.mod(df2)
floor_division = df1.floordiv(df2)
exponentiation = df1.pow(df2)

print(addition)
print(subtraction)
print(multiplication)
print(division)
print(modulus)
print(floor_division)
print(exponentiation)
```

```
df1:
   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

df2:
   A  B  C
0 10 13 16
1 11 14 17
2 12 15 18

   A  B  C
0 11 17 23
1 13 19 25
2 15 21 27

   A  B  C
0 -9 -9 -9
1 -9 -9 -9
2 -9 -9 -9

   A  B  C
0 10 52 112
1 22 70 136
2 36 90 162

   A      B      C
0 0.100000 0.307692 0.437500
1 0.181818 0.357143 0.470588
2 0.250000 0.400000 0.500000

   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

   A  B  C
0  0  0  0
1  0  0  0
2  0  0  0

   A      B      C
0    1    67108864    33232930569601
1   2048    6103515625    2251799813685248
2  531441   470184984576   150094635296999121
```



# 칼럼간 산술연산

## 산술연산에 대한 처리

```
import numpy as np
import pandas as pd
import inspect as ins

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])

print("dataframe add ", df['f'].add(df['f']))
print("dataframe sub ", df['f'].sub(df['f']))
print("dataframe sub ", df['f'].mul(df['f']))
print("dataframe sub ", df['f'].div(df['f']))
print("dataframe truediv ", df['f'].truediv(df['f']))
print("dataframe floordiv ", df['f'].floordiv(df['f']))
print("dataframe mod ", df['f'].mod(df['f']))
```

```
dataframe add a      0
b      8
c     16
d     24
Name: f, dtype: int64
dataframe sub a      0
b      0
c      0
d      0
Name: f, dtype: int64
dataframe sub a      0
b     16
c     64
d    144
Name: f, dtype: int64
dataframe sub a      NaN
b     1.0
c     1.0
d     1.0
Name: f, dtype: float64
dataframe truediv a      NaN
b     1.0
c     1.0
d     1.0
Name: f, dtype: float64
dataframe floordiv a      0
b      1
c      1
d      1
Name: f, dtype: int64
dataframe mod a      0
b      0
c      0
d      0
Name: f, dtype: int64
```

# Dataframe broadcasting

스칼라 값과의 브로드캐스팅	스칼라 값을 모든 요소에 적용
DataFrame간 브로드캐스팅	동일한 크기가 아닌 DataFrame 간의 연산을 수행할 때, 열 이름이 일치하는 요소끼리 연산이 수행
Series와의 브로드캐스팅	Series 객체의 인덱스와 DataFrame의 열 이름이 일치하는 경우, Series의 값이 해당 열의 모든 요소에 브로드캐스팅

```
import numpy as np
import pandas as pd

# 스칼라 값과의 브로드캐스팅
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
scalar = 10
result_scalar = df + scalar
print("Broadcasting with scalar:")
print(result_scalar)
df['D'] = [1, 2, 3]
# 다른 DataFrame과의 브로드캐스팅
df_other = pd.DataFrame({'A': [10, 20, 30], 'B': [40, 50, 60]})
result_df = df + df_other
print("\nBroadcasting with another DataFrame:")
print(result_df)

# Series와의 브로드캐스팅
series = pd.Series([100, 200, 300], index=['A', 'B', 'C'])
result_series = df + series
print("\nBroadcasting with a Series:")
print(result_series)
```

Broadcasting with scalar:

	A	B
0	11	14
1	12	15
2	13	16

Broadcasting with another DataFrame:

	A	B	D
0	11	44	NaN
1	22	55	NaN
2	33	66	NaN

Broadcasting with a Series:

	A	B	C	D
0	101.0	204.0	NaN	NaN
1	102.0	205.0	NaN	NaN
2	103.0	206.0	NaN	NaN

# DataFrame describe

Data의 통계량을 요약

Column별 count, mean, std, min, max, percentile 제공

```
import pandas as pd

# 예제 DataFrame 생성
data = {
    'Name': ['John', 'Anna', 'Peter', 'Linda'],
    'Age': [28, 24, 35, 32],
    'City': ['New York', 'Paris', 'Berlin', 'London']
}
df = pd.DataFrame(data)

print("DataFrame:")
print(df)

# describe() 메서드를 사용하여 DataFrame의 요약 통계량 계산
summary = df.describe()

print("\nSummary Statistics:")
print(summary)
```

DataFrame:

	Name	Age	City
0	John	28	New York
1	Anna	24	Paris
2	Peter	35	Berlin
3	Linda	32	London

Summary Statistics:

	Age
count	4.000000
mean	29.750000
std	4.787136
min	24.000000
25%	27.000000
50%	30.000000
75%	32.750000
max	35.000000

# Describe내 값을 메소드로 확인

describe() 에 결과를 mean()메소드로 확인

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(30).reshape(5, 6))
print(df.describe())
print("df mean ", df.mean())
```

	0	1	2	3	4	5
count	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000
mean	12.000000	13.000000	14.000000	15.000000	16.000000	17.000000
std	9.486833	9.486833	9.486833	9.486833	9.486833	9.486833
min	0.000000	1.000000	2.000000	3.000000	4.000000	5.000000
25%	6.000000	7.000000	8.000000	9.000000	10.000000	11.000000
50%	12.000000	13.000000	14.000000	15.000000	16.000000	17.000000
75%	18.000000	19.000000	20.000000	21.000000	22.000000	23.000000
max	24.000000	25.000000	26.000000	27.000000	28.000000	29.000000

```
df mean 0    12.0
1     13.0
2     14.0
3     15.0
4     16.0
5     17.0
dtype: float64
```

# 합, 평균, 표준편차, 분산 : 열

## 열에 대한 합, 평균, 표준편차, 분산 처리

```
import numpy as np
import pandas as pd
import inspect as ins

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])

print(df)
print(df.sum(axis = 0))
print(df.mean(axis = 0))
print(df.std(axis = 0))
print(df.var(axis = 0))
```

```
f    g    h    i
a    0    1    2    3
b    4    5    6    7
c    8    9   10   11
d   12   13   14   15
f    24
g    28
h    32
i    36
dtype: int64
f    6.0
g    7.0
h    8.0
i    9.0
dtype: float64
f    5.163978
g    5.163978
h    5.163978
i    5.163978
dtype: float64
f    26.666667
g    26.666667
h    26.666667
i    26.666667
dtype: float64
```

# 합, 평균, 표준편차, 분산 : 행

행에 대한 합, 평균, 표준편차, 분산 처리

```
import numpy as np
import pandas as pd
import inspect as ins

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])
print(df.sum(axis = 1))
print(df.mean(axis = 1))
print(df.std(axis = 1))
print(df.var(axis = 1))
```

```
a      6
b     22
c     38
d     54
dtype: int64
a      1.5
b      5.5
c      9.5
d     13.5
dtype: float64
a      1.290994
b      1.290994
c      1.290994
d      1.290994
dtype: float64
a      1.666667
b      1.666667
c      1.666667
d      1.666667
dtype: float64
```

# min/max : 열

## 열에 대한 min/max 처리

```
import numpy as np
import pandas as pd
import inspect as ins

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])

print(df)
print(df.min(axis = 0))
print(df.max(axis = 0))
```

```
   f  g  h  i
a  0  1  2  3
b  4  5  6  7
c  8  9 10 11
d 12 13 14 15
f    0
g    1
h    2
i    3
dtype: int64
f    12
g    13
h    14
i    15
dtype: int64
```

# min/max : 행

## 행에 대한 min/max 처리

```
import numpy as np
import pandas as pd
import inspect as ins

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])

print(df)
print(df.min(axis = 1))
print(df.max(axis = 1))
```

```
   f  g  h  i
a  0  1  2  3
b  4  5  6  7
c  8  9 10 11
d 12 13 14 15

a    0
b    4
c    8
d   12
dtype: int64
a    3
b    7
c   11
d   15
dtype: int64
```



# All

## 논리 연산에 대한 행(axis=1), 열(axis=0)에 대한 처리

```
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])

print(df)
print((df == df).all(axis = 0))
print((df == df).all(axis = 1))
```

```
   f  g  h  i
a  0  1  2  3
b  4  5  6  7
c  8  9 10 11
d 12 13 14 15
f   True
g   True
h   True
i   True
dtype: bool
a   True
b   True
c   True
d   True
dtype: bool
```

# any

행과 열의 논리 연산을 한 결과에 대해 축약형 논리 값 표시

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])

print(df)
print((df > 5).any(axis = 0))
print((df > 5).any(axis = 1))
```

```
   f  g  h  i
a  0  1  2  3
b  4  5  6  7
c  8  9 10 11
d 12 13 14 15
```

```
f    True
g    True
h    True
i    True
dtype: bool
a    False
b     True
c     True
d     True
dtype: bool
```

```
      f      g      h      i
a  False  False  False  False
b  False  False   True   True
c   True   True   True   True
d   True   True   True   True
```

# equals()

---

계산된 결과가 동등한지 처리하는 메소드

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])
print(df.equals(df))
```

True

# Data import, export

Import : `read_csv('파일경로/파일이름.csv')`

export : `data.to_csv('파일경로/파일명.csv', index = False)`

```
import pandas as pd

# 데이터프레임 생성
data = {'Name': ['Tom', 'Nick', 'John', 'Tom'],
        'Age': [20, 21, 19, 18],
        'Gender': ['Male', 'Male', 'Male', 'Male']}

df = pd.DataFrame(data)

# 데이터프레임을 'my_data.csv' 파일로 저장
df.to_csv('my_data.csv', index = False)

# 'my_data.csv' 파일을 읽어들이어 새로운 데이터프레임 생성
new_df = pd.read_csv('my_data.csv')

print(new_df)
```

	Name	Age	Gender
0	Tom	20	Male
1	Nick	21	Male
2	John	19	Male
3	Tom	18	Male

# 시계열 데이터

Index가 시계열 데이터인 데이터 (2021-03-26 21:06:29.35)

Data\_range : 시작, 끝 시간을 통해 index 생성

Resample : 데이터를 특정 주기로 추출

```
import numpy as np
import pandas as pd

date_idx = pd.date_range("2023/1/01", "2023/7/01", freq='W')
l = len(date_idx)
df = pd.DataFrame(np.random.randn(l, 1), index = date_idx)
print(df.head())
resample = df.resample("m")
print(resample.first())
print(resample.mean())
```

	0	1	2	3	4	5
2023-01-01	0.053235	1.308637	-0.297778	1.080034	1.254741	-0.553223
2023-01-08	-0.511498	1.294382	0.564162	0.098907	-0.195665	-1.516977
2023-01-15	0.019408	-0.090466	-1.005783	0.293538	-0.174458	-1.481576
2023-01-22	0.157972	-1.030956	0.454619	-0.797458	-1.842199	-0.333161
2023-01-29	-0.256425	0.827676	-0.705554	1.852422	-1.308226	-1.783192

	6	7	8	9	...	16
2023-01-01	-1.531054	0.251069	1.775010	-1.244792	...	1.089860 -1.10
2023-01-08	-0.733258	-0.223582	-0.685662	-0.033860	...	1.208476 0.4
2023-01-15	-0.267106	0.048387	-1.256633	0.363592	...	-1.116174 -0.4
2023-01-22	0.307325	0.682755	0.662647	-2.029424	...	0.207683 -1.2
2023-01-29	0.993480	-2.077730	0.865706	1.424244	...	-0.350070 0.3

	18	19	20	21	22	23
2023-01-01	-0.252243	0.301210	0.607103	0.525562	-0.185933	1.924403
2023-01-08	-0.392589	0.192110	-0.572805	-0.158747	0.986850	1.474007
2023-01-15	0.306365	-0.742534	0.312897	0.560748	-0.790920	-1.035521
2023-01-22	-0.806385	-0.175612	0.213103	0.241251	1.578347	-0.119765
2023-01-29	-1.416976	1.204031	0.153536	1.341130	1.495042	-0.644656

	24	25
2023-01-01	1.805569	0.271337
2023-01-08	0.629361	-0.297793
2023-01-15	1.984777	0.263940
2023-01-22	-1.388890	0.398037
2023-01-29	-1.168753	0.133488

[5 rows x 26 columns]						
	0	1	2	3	4	5
2023-01-31	0.053235	1.308637	-0.297778	1.080034	1.254741	-0.553223
2023-02-28	0.091863	-1.565904	1.068123	0.485601	-0.033029	2.227276
2023-03-31	2.812615	1.281153	0.369817	-0.894421	1.009199	0.853354
2023-04-30	-1.026697	0.790263	1.815823	-0.103652	0.629095	0.251795
2023-05-31	-1.770744	0.681426	-0.397271	-1.309571	-1.574932	-0.346916
2023-06-30	1.517156	0.109748	-1.925071	-0.194772	2.028717	-0.831925

## 연습문제

---

1. 모든 열에 대해 각 값에서 열의 평균을 빼는 산술 연산을 적용하세요.
2. 그 후에, 각 열의 표준편차를 계산하세요.
3. 마지막으로, 'A'열에서 값이 50보다 큰 값이 있는지 any 함수를 사용하여 확인하세요.

```
np.random.seed(0)
df = pd.DataFrame(np.random.randint(0,100,size=(100, 4)),
                  columns=list('ABCD'))
```

```
# 모든 열에 대해 각 값에서 열의 평균을 빼는 산술 연산을 적용
df = df.apply(lambda x: x - x.mean())
```

# 연습문제 코드

```
import numpy as np
import pandas as pd

# Seed for reproducibility
np.random.seed(0)

# 초기 데이터프레임 생성
df = pd.DataFrame(np.random.randint(0, 100, size = (100, 4)), columns = list('ABCD'))

print("Original DataFrame:")
print(df)

# 모든 열에 대해 각 값에서 열의 평균을 빼는 산술 연산을 적용
df = df.apply(lambda x: x - x.mean())

print("\nDataFrame after arithmetic operation:")
print(df)

# 각 열의 표준편차를 계산
std_dev = df.std()

print("\nStandard deviation of each column:")
print(std_dev)

# 'A'열에서 값이 50보다 큰 값이 있는지 any 함수를 사용하여 확인
check = (df['A'] > 50).any()

print("\nCheck if any value in column 'A' is greater than 50:")
print(check)
```

# 데이터 병합



# 행과 열기준으로 연결 1

## 행 / 열기준으로 두 객체를 연결

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4, 4),
                  index = ['a', 'b', 'c', 'd'],
                  columns = ['f', 'g', 'h', 'i'])
print(pd.concat([df, df]))
print(pd.concat([df, df], axis = 1))
```

	f	g	h	i
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

	f	g	h	i	f	g	h	i
a	0	1	2	3	0	1	2	3
b	4	5	6	7	4	5	6	7
c	8	9	10	11	8	9	10	11
d	12	13	14	15	12	13	14	15

## 행과 열기준으로 연결 2

Axis=0 일 때 Column이 맞지 않는 경우 Nan값 포함하여 concat.

```
import pandas as pd

# 초기 데이터프레임 생성
df1 = pd.DataFrame({'A': ['B0', 'B1', 'B2', 'B3'],
                    'B': ['C0', 'C1', 'C2', 'C3'],
                    'C': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])

df2 = pd.DataFrame({'A': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'E': ['E4', 'E5', 'E6', 'E7']},
                    index=[4, 5, 2, 3])

print("Original DataFrames:")
print(df1)
print(df2)

# concat 함수를 사용하여 df1과 df2를 연결
df_concat = pd.concat([df1, df2])

print("\nDataFrame after concatenation:")
print(df_concat)
```

Original DataFrames:

	A	B	C
0	B0	C0	D0
1	B1	C1	D1
2	B2	C2	D2
3	B3	C3	D3

	A	C	E
4	B4	C4	E4
5	B5	C5	E5
2	B6	C6	E6
3	B7	C7	E7

DataFrame after concatenation:

	A	B	C	E
0	B0	C0	D0	NaN
1	B1	C1	D1	NaN
2	B2	C2	D2	NaN
3	B3	C3	D3	NaN
4	B4	NaN	C4	E4
5	B5	NaN	C5	E5
2	B6	NaN	C6	E6
3	B7	NaN	C7	E7

## 행과 열기준으로 연결 3

Axis=0 일 때 Column의 순서가 맞지 않으면 자동 맞춤

```
import pandas as pd

# 초기 데이터프레임 생성
df1 = pd.DataFrame({'A': ['B0', 'B1', 'B2', 'B3'],
                    'B': ['C0', 'C1', 'C2', 'C3'],
                    'C': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])

df2 = pd.DataFrame({'A': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'E': ['E4', 'E5', 'E6', 'E7']},
                    index=[4, 5, 2, 3])

print("Original DataFrames:")
print(df1)
print(df2)

# concat 함수를 사용하여 df1과 df2를 연결
df_concat = pd.concat([df1, df2])

print("\nDataFrame after concatenation:")
print(df_concat)
```

Original DataFrames:

	A	B	C
0	B0	C0	D0
1	B1	C1	D1
2	B2	C2	D2
3	B3	C3	D3

	A	C	E
4	B4	C4	E4
5	B5	C5	E5
2	B6	C6	E6
3	B7	C7	E7

DataFrame after concatenation:

	A	B	C	E
0	B0	C0	D0	NaN
1	B1	C1	D1	NaN
2	B2	C2	D2	NaN
3	B3	C3	D3	NaN
4	B4	NaN	C4	E4
5	B5	NaN	C5	E5
2	B6	NaN	C6	E6
3	B7	NaN	C7	E7

# 병합

## Data를 특정 기준열에 맞춰 병합

```
import pandas as pd

# 초기 데이터프레임 생성
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                    'B': ['B0', 'B1', 'B2'],
                    'key': ['K0', 'K1', 'K2']})

df2 = pd.DataFrame({'C': ['C0', 'C1', 'C2'],
                    'D': ['D0', 'D1', 'D2'],
                    'key': ['K0', 'K2', 'K3']})

print("Original DataFrames:")
print(df1)
print(df2)

# merge 함수를 사용하여 df1과 df2를 'key' 열을 기준으로 병합
df_merge = pd.merge(df1, df2, on='key')

print("\nDataFrame after merge:")
print(df_merge)
```

Original DataFrames:

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2

	C	D	key
0	C0	D0	K0
1	C1	D1	K2
2	C2	D2	K3

DataFrame after merge:

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A2	B2	K2	C1	D1

# Inner join

subject\_id에 값으로 일치하는 것만 처리

```
import pandas as pd
import inspect as ins

raw_data = {'subject_id' : ['1', '2', '3', '4', '5'],
            'first_name' : ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
            'last_name' : ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}
df_a = pd.DataFrame(raw_data)

raw_data = {'subject_id' : ['4', '5', '6', '7', '8'],
            'first_name' : ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
            'last_name' : ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}
df_b = pd.DataFrame(raw_data)

print(df_a)
print(df_b)
print(pd.merge(df_a, df_b, on = 'subject_id'))
print(pd.merge(df_a, df_b, on = 'subject_id', how = 'inner'))
```

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black

# Outer join

열기준(subject\_id)으로 모든 것을 표시

```
import pandas as pd

raw_data = {'subject_id' : ['1', '2', '3', '4', '5'],
            'first_name' : ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
            'last_name' : ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}
df_a = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name', 'last_name'])

raw_data = {'subject_id' : ['4', '5', '6', '7', '8'],
            'first_name' : ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
            'last_name' : ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}
df_b = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name', 'last_name'])

print(pd.merge(df_a, df_b, on = 'subject_id', how = 'outer'))
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black
5	6	NaN	NaN	Bran	Balwner
6	7	NaN	NaN	Bryce	Brice
7	8	NaN	NaN	Betty	Btisan

## 연습문제

---

1. `merge` 함수를 사용하여 `df1`과 `df2`를 'key' 열을 기준으로 병합하세요..
2. Key열을 index 값으로 설정하고
3. `concat` 함수를 사용하여 `df1`과 `df2`를 가로 방향으로 연결하세요.

```
df1 = pd.DataFrame({  
    'A': ['A0', 'A1', 'A2'],  
    'B': ['B0', 'B1', 'B2'],  
    'key': ['K0', 'K1', 'K2']  
})
```

```
df2 = pd.DataFrame({  
    'C': ['C0', 'C1', 'C2'],  
    'D': ['D0', 'D1', 'D2'],  
    'key': ['K0', 'K1', 'K2']  
})
```

## 연습문제 코드

```
import pandas as pd

# 초기 데이터프레임 생성
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                    'B': ['B0', 'B1', 'B2'],
                    'key': ['K0', 'K1', 'K2']})

df2 = pd.DataFrame({'C': ['C0', 'C1', 'C2'],
                    'D': ['D0', 'D1', 'D2'],
                    'key': ['K0', 'K1', 'K2']})

print("Original DataFrames:")
print(df1)
print(df2)

# merge 함수를 사용하여 df1과 df2를 'key' 열을 기준으로 병합
df_merge = pd.merge(df1, df2, on='key')

print("\nDataFrame after merge:")
print(df_merge)

# concat 함수를 사용하여 df1과 df2를 가로 방향으로 연결
df_concat = pd.concat([df1.set_index('key'), df2.set_index('key')], axis=1)
print("\nDataFrame after concatenation:")
print(df_concat)
```



# Multi index 데이터프레임

# MultiIndex

## Index나 column에 대한 메타데이터에 대한 객체화

```
import pandas as pd
```

```
help(pd.MultiIndex)
```

Help on class MultiIndex in module pandas.core.indexes.multi:

```
class MultiIndex(pandas.core.indexes.base.Index)
| MultiIndex(levels=None, codes=None, sortorder=None, names=None, dtype=None, copy=False, name=None,
|
| A multi-level, or hierarchical, index object for pandas objects.
|
| Parameters
| -----
| levels : sequence of arrays
|     The unique labels for each level.
| codes : sequence of arrays
|     Integers for each level designating which label at each location.
| sortorder : optional int
|     Level of sortedness (must be lexicographically sorted by that
|     level).
| names : optional sequence of objects
|     Names for each of the index levels. (name is accepted for compat).
| copy : bool, default False
|     Copy the meta-data.
| verify_integrity : bool, default True
|     Check that the levels/codes are consistent and valid.
```

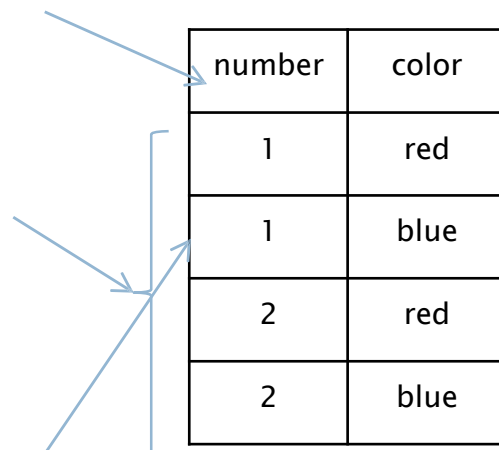
# 표에 대한 메타데이터 관리

실제 데이터를 접근할 때 별도의 메타데이터로 관리가 필요할 경우

names는 각 열의 이름을 관리

levels는 각 열의 대표값을 list로 관리

codes는 각 열의 실제 위치를 관리



number	color
1	red
1	blue
2	red
2	blue



객체화

```
MultiIndex(levels = [[1, 2],
                     [u'blue', u'red']],
           codes = [[0, 0, 1, 1],
                    [1, 0, 1, 0]],
           names = [u'number', u'color'])
```

# Index에 대한 객체

Levels, Codes/labels, names으로 분리해서 접근할 수 있는 정보를 관리

## Index(행)

names

Levels에 대한 명

levels

Index에 대한 이름관리

Codes

Index에 대한 위치관리

Column(열)

	col1	col2	col3
1 row			
2 row			
3 row			

## Column(열)

names

Levels에 대한 명

levels

Column 에 대한 이름관리

Codes

Column 에 대한 위치관리

# MultiIndex 생성하기: tuple

Tuple 형태로 전달시 계층에 대한 이름(levels), 각 이름 별 계층 위치(codes) 그리고 level에 대한 이름(names)

```
import pandas as pd

tuples = [(1, u'red'), (1, u'blue'), (2, u'red'), (2, u'blue')]
mi = pd.MultiIndex.from_tuples(tuples, names = ('number', 'color'))
print(mi)
print(mi.levels)
print(mi.codes)
print(mi.names)

MultiIndex([(1, 'red'),
            (1, 'blue'),
            (2, 'red'),
            (2, 'blue')],
           names=['number', 'color'])
[[1, 2], ['blue', 'red']]
[[0, 0, 1, 1], [1, 0, 1, 0]]
['number', 'color']
```

## Multindex 생성하기: array

List로 lable 형태로 전달시 계층에 대한 이름 (levels), 각 이름별 계층 위치(codes) 그리고 level에 대한 이름(names)

```
import pandas as pd

arrays = [[1, 1, 2, 2], ['red', 'blue', 'red', 'blue']]
mi1 = pd.MultiIndex.from_arrays(arrays, names = ('number', 'color'))
print(mi)
print(mi.levels)
print(mi.codes)
print(mi.names)
```

```
MultiIndex([(1, 'red'),
            (1, 'blue'),
            (2, 'red'),
            (2, 'blue')],
           names=['number', 'color'])
[[1, 2], ['blue', 'red']]
[[0, 0, 1, 1], [1, 0, 1, 0]]
['number', 'color']
```

# Multindex 생성하기: product

List를 level 형태로 전달시 계층에 대한 이름 (levels), 각 이름별 계층 위치(codes) 그리고 level에 대한 이름(names)

```
import pandas as pd

numbers = [0, 1, 2]
colors = ['green', 'purple']
mi = pd.MultiIndex.from_product([numbers, colors], names = ['number', 'color'])
print(mi)
print(mi.levels)
print(mi.codes)
print(mi.names)
```

```
MultiIndex([(0, 'green'),
            (0, 'purple'),
            (1, 'green'),
            (1, 'purple'),
            (2, 'green'),
            (2, 'purple')],
           names=['number', 'color'])
[[0, 1, 2], ['green', 'purple']]
[[0, 0, 1, 1, 2, 2], [0, 1, 0, 1, 0, 1]]
['number', 'color']
```

## DataFrame : 상위 column 조회

DataFrame은 column 구조에 따라 구분해서 접근해서 조회함

```
import numpy as np
import pandas as pd

arrays = [['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],
          ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two']]
tuples = list(zip(*arrays))

index = pd.MultiIndex.from_tuples(tuples, names = ['first', 'second'])

df = pd.DataFrame(np.random.randn(3, 8), index = ['A', 'B', 'C'], columns = index)
print(type(df['bar']))
print(df['bar'])
```

```
<class 'pandas.core.frame.DataFrame'>
second      one      two
A      0.294810 -0.197397
B      1.705716 -1.782617
C      1.241464 -0.518545
```



# DataFrame 조회 : 하위 칼럼

DataFrame은 column 구조에 따라 순차적 접근해서 조회함

```
import numpy as np
import pandas as pd

arrays = [['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],
          ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two']]
tuples = list(zip(*arrays))

index = pd.MultiIndex.from_tuples(tuples, names = ['first', 'second'])

df = pd.DataFrame(np.random.randn(3, 8), index = ['A', 'B', 'C'], columns = index)
print(df['bar', 'one'])
print(df['bar']['one'])
```

```
A    1.077595
B    1.011048
C   -1.240030
Name: (bar, one), dtype: float64
A    1.077595
B    1.011048
C   -1.240030
Name: one, dtype: float64
```

## 연습문제

1. 아래와 같은 multi column dataframe을 만들고
2. 레벨 0 인덱스가 2인 모든 행을 선택하세요
3. 이러한 행에서 'A'와 'B'열의 평균을 계산하세요.

```
import pandas as pd
import numpy as np

np.random.seed(0)

index = pd.MultiIndex.from_tuples([(i, j) for i in range(5) for j in range(5)])
df = pd.DataFrame(np.random.randint(0, 100, size=(25, 4)), index=index, columns=list('ABCD'))
```

## 연습문제 코드

```
import pandas as pd
import numpy as np

# Seed for reproducibility
np.random.seed(0)

# 초기 멀티 컬럼 데이터프레임 생성
index = pd.MultiIndex.from_tuples([(i, j) for i in range(5) for j in range(5)])
df = pd.DataFrame(np.random.randint(0, 100, size=(25, 4)), index=index, columns=list('ABCD'))

print("Original DataFrame:")
print(df)

# 레벨 0 인덱스가 2인 모든 행을 선택
df_indexed = df.loc[2]

print("\nDataFrame after indexing with level 0 index equal to 2:")
print(df_indexed)

# 선택된 행에서 'A'와 'B'열의 평균을 계산
average = df_indexed[['A', 'B']].mean()

print("\nAverage of columns 'A' and 'B' for the selected rows:")
print(average)
```

# Groupby

# Groupby

## DataFrame에 대해 group화해서 칼럼들에 대한 연산 처리

```
import pandas as pd

help(pd.DataFrame.groupby)
```

Help on function groupby in module pandas.core.frame:

groupby(self, by=None, axis: 'Axis' = 0, level: 'IndexLabel | None' = None, as\_index: 'bool' = True, sort: 'bool' = True, Group DataFrame using a mapper or by a Series of columns.

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

Parameters  
-----

by : mapping, function, label, or list of labels  
Used to determine the groups for the groupby.  
If ``by`` is a function, it's called on each value of the object's index. If a dict or Series is passed, the Series or dict VALUES will be used to determine the groups (the Series' values are first aligned; see ``.align()`` method). If a list or ndarray of length equal to the selected axis is passed (see the 'groupby user guide' <[https://pandas.pydata.org/pandas-docs/stable/user\\_guide/groupby.html#splitting-an-object-into-groups](https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html#splitting-an-object-into-groups)>`\_), the values are used as-is to determine the groups. A label or list of labels may be passed to group by the columns in ``self``. Notice that a tuple is interpreted as a (single) key.

# Groupby : 1칼럼

하나의 칼럼을 기준으로 group화해서 칼럼들에 대한 연산 처리

```
import pandas as pd
import inspect as ins

d = {'one' : [1, 1, 1, 1, 1],
      'two' : [2, 2, 2, 2, 2],
      'letter' : ['a', 'a', 'b', 'b', 'c']}
```

```
# Create dataframe
df1 = pd.DataFrame(d)
print(df1)
one = df1.groupby('letter')
print(one)
print(one.sum())
```

```
   one  two letter
0    1    2     a
1    1    2     a
2    1    2     b
3    1    2     b
4    1    2     c
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f342f82f9a0>
   one  two
letter
a      2    4
b      2    4
c      1    2
```

	letter	one	two
0	a	1	2
1	a	1	2
2	b	1	2
3	b	1	2
4	c	1	2



	one	two
letter		
a	2	4
b	2	4
c	1	2

# Groupby : 여러 칼럼 1

칼럼기준을 그룹을 연계하면 인덱스가 multi index  
로 변환

```
import pandas as pd
import inspect as ins

d = {'one' : [1, 1, 1, 1, 1],
      'two' : [2, 2, 2, 2, 2],
      'letter' : ['a', 'a', 'b', 'b', 'c']}

# Create dataframe
df1 = pd.DataFrame(d)
print(df1)
letterone = df1.groupby(['letter', 'one']).sum()
print(letterone)
```

```
   one  two letter
0    1    2     a
1    1    2     a
2    1    2     b
3    1    2     b
4    1    2     c

      two
letter one
a        1    4
b        1    4
c        1    2
```

	letter	one	two
0	a	1	2
1	a	1	2
2	b	1	2
3	b	1	2
4	c	1	2



		two
letter	one	
a	1	4
b	1	4
c	1	2

# Groupby : 여러 칼럼 2

as\_index=False로 처리해서 groupby메소드 이후에도 index구성이 변하지 않도록 처리

```
import pandas as pd
import inspect as ins

d = {'one' : [1, 1, 1, 1, 1],
      'two' : [2, 2, 2, 2, 2],
      'letter' : ['a', 'a', 'b', 'b', 'c']}

# Create dataframe
df1 = pd.DataFrame(d)
lettertwo = df1.groupby(['letter', 'one'], as_index = False).sum()
print(lettertwo)
print(lettertwo.index)
```

```
   letter  one  two
0      a    1    4
1      b    1    4
2      c    1    2
RangeIndex(start=0, stop=3, step=1)
```

	letter	one	two
0	a	1	2
1	a	1	2
2	b	1	2
3	b	1	2
4	c	1	2



	letter	one	two
0	a	1	4
1	b	1	4
2	c	1	2



## mean/size 조회

특정 열을 기준을 가지고 특정 열에 대한 값을 처리

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.arange(30).reshape(5, 6))
df1 = df.copy()

df1[0] = ['A', 'B', 'B', 'C', 'C']
df1[1] = ['1', '2', '2', '3', '3']

print(df1[2].groupby(df1[0]).obj)
print(df1[2].groupby(df1[0]).mean())
print(df1[2].groupby(df1[0]).size())
```

```
0    2
1    8
2   14
3   20
4   26
Name: 2, dtype: int64
0
A    2.0
B   11.0
C   23.0
Name: 2, dtype: float64
0
A    1
B    2
C    2
Name: 2, dtype: int64
```

## Groupby + mean: 2개 그룹

Groupby + mean 메소드를 사용해서 2개 그룹에 대한 평균값을 계산 groupby 내의 파라미터를 칼럼구분([ , ]) 처리

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(30).reshape(5, 6))

df1 = df.copy()
df1[0] = ['A', 'B', 'B', 'C', 'C']

df1[1] = ['1', '2', '2', '3', '3']
print(df1.groupby([0, 1]).mean())
```

	2	3	4	5
0 1				
A 1	2.0	3.0	4.0	5.0
B 2	11.0	12.0	13.0	14.0
C 3	23.0	24.0	25.0	26.0

# DataFrameGroupby : iterable

DataFrameGroupby 객체도 iterable 이므로 행이 name, 데이터가 group으로 출력됨

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(30).reshape(5, 6))
df1 = df.copy()

df1[0] = ['A', 'B', 'B', 'C', 'C']
df1[1] = ['1', '2', '2', '3', '3']

for name, group in df1.groupby([0, 1]):
    # print the name of the regiment
    print(name)
    # print the data of that regiment
    print(group)
```

```
('A', '1')
   0  1  2  3  4  5
0  A  1  2  3  4  5
('B', '2')
   0  1  2  3  4  5
1  B  2  8  9 10 11
2  B  2 14 15 16 17
('C', '3')
   0  1  2  3  4  5
3  C  3 20 21 22 23
4  C  3 26 27 28 29
```

# Groupby에 대한 describe 확인

## Groupby에 대한 describe() 처리

```
import numpy as np
import pandas as pd

data = {'Platoon' : ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C'],
        'Casualties' : [1, 4, 5, 7, 5, 5, 6, 1, 4, 5, 6, 7, 4, 6, 4, 6]}
df = pd.DataFrame(data)

print(df.groupby("Platoon").describe())
```

	Casualties							
	count	mean	std	min	25%	50%	75%	max
Platoon								
A	6.0	4.5	1.974842	1.0	4.25	5.0	5.0	7.0
B	5.0	4.4	2.073644	1.0	4.00	5.0	6.0	6.0
C	5.0	5.4	1.341641	4.0	4.00	6.0	6.0	7.0

## 연습문제

---

1. 아래와 같은 dataframe을 만들고
2. Animal 컬럼을 기준으로 데이터프레임을 그룹화하세요.
3. 그 다음, 각 그룹의 Max Speed와 Weight 컬럼의 평균을 계산하세요.

```
import pandas as pd
```

```
df = pd.DataFrame({  
    'Animal': ['Falcon', 'Falcon', 'Parrot', 'Parrot'],  
    'Max Speed': [380., 370., 24., 26.],  
    'Weight': [1, 1.5, 0.3, 0.5]  
})
```

# 연습문제 코드

```
import pandas as pd

# 초기 데이터프레임 생성
df = pd.DataFrame({
    'Animal': ['Falcon', 'Falcon', 'Parrot', 'Parrot'],
    'Max Speed': [380., 370., 24., 26.],
    'Weight': [1, 1.5, 0.3, 0.5]
})

print("Original DataFrame:")
print(df)

# 'Animal' 컬럼을 기준으로 데이터프레임을 그룹화하고, 각 그룹의 'Max Speed'와 'Weight' 컬럼의 평균을 계산
grouped = df.groupby('Animal').mean()

print("\nDataFrame after groupby and mean calculation:")
print(grouped)
```

Original DataFrame:

	Animal	Max Speed	Weight
0	Falcon	380.0	1.0
1	Falcon	370.0	1.5
2	Parrot	24.0	0.3
3	Parrot	26.0	0.5

DataFrame after groupby and mean calculation:

	Animal	Max Speed	Weight
	Falcon	375.0	1.25
	Parrot	25.0	0.40

# DATAFRAME APPLY

# Apply

Apply 메소드는 내부 함수를 모든 행, 열에 대해 계산을 처리함

`df.apply(func)`

Column(열)

	col1	col2	col3
1 row			
2 row			
3 row			

Index(행)

Apply 메소드



`func(df 원소값)`을 넣어 전체 값을 변환

Column(열)

	col1	col2	col3
1 row			
2 row			
3 row			

Index(행)



# Dataframe 모든 원소에 적용

Apply 메소드는 Dataframe 각 열, 행에 함수를 적용  
(series에 적용)

```
import numpy as np
import pandas as pd

dfx = pd.DataFrame(np.arange(16).reshape(4, 4))
print(dfx)

print(dfx.apply(sum))
print(dfx.apply(np.mean))
```

```
   0  1  2  3
0  0  1  2  3
1  4  5  6  7
2  8  9 10 11
3 12 13 14 15
0    24
1    28
2    32
3    36
dtype: int64
0    6.0
1    7.0
2    8.0
3    9.0
dtype: float64
```

# 사용자 함수 정의 확인

함수를 직접 정의하여 dataframe에 apply

```
import numpy as np
import pandas as pd

# 데이터프레임 생성
df = pd.DataFrame({'A': [1, 2, 3, 4],
                   'B': [10, 20, 30, 40],
                   'C': [100, 200, 300, 400]})

# 함수 정의 (제곱 값을 반환)
def square(x):
    return x**2

# apply 함수를 사용하여 각 열에 함수 적용
df = df.apply(square)

print(df)
```

	A	B	C
0	1	100	10000
1	4	400	40000
2	9	900	90000
3	16	1600	160000

## Dataframe apply 적용

Platoon 칼럼기준으로 Casulties 값을 가지고 합산, 평균, 표준편차, 분산을 계산

```
import numpy as np
import pandas as pd

data = {'Platoon' : ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C'],
        'Casualties' : [1, 4, 5, 7, 5, 5, 6, 1, 4, 5, 6, 7, 4, 6, 4, 6]}
df = pd.DataFrame(data)

print(df.groupby('Platoon')['Casualties'].apply(sum))
print(df.groupby('Platoon')['Casualties'].apply(np.count_nonzero))
print(df.groupby('Platoon')['Casualties'].apply(np.mean))
print(df.groupby('Platoon')['Casualties'].apply(np.std))
print(df.groupby('Platoon')['Casualties'].apply(np.var))
```

# Dataframe apply 적용

Platoon 칼럼기준으로 Casualties 값을 가지고 합산, 평균, 표준편차, 분산을 계산

```
Platoon
A    27
B    22
C    27
Name: Casualties, dtype: int64
Platoon
A     6
B     5
C     5
Name: Casualties, dtype: int64
Platoon
A    4.5
B    4.4
C    5.4
Name: Casualties, dtype: float64
Platoon
A    1.802776
B    1.854724
C    1.200000
Name: Casualties, dtype: float64
Platoon
A    3.25
B    3.44
C    1.44
Name: Casualties, dtype: float64
```

# Name 칼럼에 apply 메소드 적용

문자열로 저장된 칼럼에 대해 소문자를 대문자로 전환

```
import numpy as np
import pandas as pd

data = {'name' : ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'year' : [2012, 2012, 2013, 2014, 2014],
        'reports' : [4, 24, 31, 2, 3],
        'coverage' : [25, 94, 57, 62, 70]}
df = pd.DataFrame(data, index = ['Cochice', 'Pima', 'Santa Cruz', 'Maricopa', 'Yuma'])

capitalizer = lambda x: x.upper()
print(df['name'].apply(capitalizer))
```

```
Cochice      JASON
Pima         MOLLY
Santa Cruz   TINA
Maricopa     JAKE
Yuma         AMY
Name: name, dtype: object
```

# map

사전에 정의한 내용을 변수에 적용할 수 있는 기능  
 Series에서만 적용 가능  
 머신러닝 모델에 학습하기 전 입력데이터의 형태를  
 모두 숫자로 변환하기 위해 많이 사용되는 함수

```
import pandas as pd
initial_list = ['A', 'B', 'A', 'C', 'C', 'A', 'B', 'A', 'D', 'D', 'A', 'D']
initial_series = pd.Series(initial_list)
initial_series
initial_dict = {'A' : 1, 'B' : 2, 'C' : 3, 'D' : 4}
initial_dict

initial_after = initial_series.map(initial_dict)
pd.DataFrame({'변환 전':initial_series, '변환 후':initial_after})
```

	변환 전	변환 후
0	A	1
1	B	2
2	A	1
3	C	3
4	C	3
5	A	1
6	B	2
7	A	1
8	D	4
9	D	4
10	A	1
11	D	4

# Applymap

Dataframe에서만 사용 가능  
문자열 칼럼은 변경없이 숫자타입일 경우는 100을 곱셈함

```
import numpy as np
import pandas as pd

data = {'name' : ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'year' : [2012, 2012, 2013, 2014, 2014],
        'reprots' : [4, 24, 31, 2, 3],
        'coverage' : [25, 94, 57, 62, 70]}
df = pd.DataFrame(data, index = ['Cochice', 'Pima', 'Santa Cruz', 'Maricopa', 'Yuma'])

def times100(x):
    if type(x) is str:
        return x
    elif x:
        return 100 * x
    else:
        return
print(df.applymap(times100))
```

	name	year	reprots	coverage
Cochice	Jason	201200	400	2500
Pima	Molly	201200	2400	9400
Santa Cruz	Tina	201300	3100	5700
Maricopa	Jake	201400	200	6200
Yuma	Amy	201400	300	7000

# Applymap vs apply

Applymap : dataframe 각 원소에 적용

Apply : dataframe 각 행, 열에 적용

```
import pandas as pd

# 데이터프레임 생성
df = pd.DataFrame({'A': [1, 2, 3, 4],
                    'B': [10, 20, 30, 40],
                    'C': [100, 200, 300, 400]})

# apply에 적용 가능하지만 applymap에 적용 불가능한 함수 정의 (각 열의 평균 계산)
def col_mean(col):
    return col.mean()

# apply 함수를 사용하여 각 열에 함수 적용
result_apply = df.apply(col_mean)

print("Result with apply:")
print(result_apply)

# applymap 함수를 사용하여 각 열에 함수 적용 시도 (오류 발생)
try:
    result_applymap = df.applymap(col_mean)
except Exception as e:
    print("\nError with applymap:")
    print(str(e))
```

Result with apply:

```
A      2.5
B     25.0
C    250.0
dtype: float64
```

Error with applymap:

```
'int' object has no attribute 'mean'
```



## 연습문제

---

1. 아래와 같은 dataframe을 만들고
2. 사용자 정의 함수 `subtract_mean`를 생성하여 각 열의 값에서 해당 열의 평균을 뺀 값을 반환하세요.
3. 그 다음, `apply` 함수를 사용하여 `subtract_mean` 함수를 데이터프레임의 각 열에 적용하세요.

```
import pandas as pd

df = pd.DataFrame({
    'A': [1, 2, 3, 4],
    'B': [10, 20, 30, 40],
    'C': [100, 200, 300, 400]
})
```

# 연습문제 코드

```
import pandas as pd

# 초기 데이터프레임 생성
df = pd.DataFrame({'A': [1, 2, 3, 4],
                   'B': [10, 20, 30, 40],
                   'C': [100, 200, 300, 400]})

print("Original DataFrame:")
print(df)

# 사용자 정의 함수 정의
def subtract_mean(column):
    return column - column.mean()

# apply 함수를 사용하여 각 열에 함수 적용
df = df.apply(subtract_mean)

print("\nDataFrame after applying user-defined function:")
print(df)
```

Original DataFrame:

	A	B	C
0	1	10	100
1	2	20	200
2	3	30	300
3	4	40	400

DataFrame after applying user-defined function:

	A	B	C
0	-1.5	-15.0	-150.0
1	-0.5	-5.0	-50.0
2	0.5	5.0	50.0
3	1.5	15.0	150.0

# Quiz

---

- **주식 데이터 분석**
- **문제:** 당신은 주식 투자를 좋아하는 데이터 과학자입니다. 최근에 몇 가지 주식에 대한 일일 가격 데이터를 수집하였고, 이를 바탕으로 아래 작업을 수행하세요:
  1. Pandas를 사용하여 데이터 파일을 읽고 DataFrame으로 변환하는 코드를 작성하세요.
  2. 회사별로 평균 종가를 계산하고, 가장 높은 평균 종가를 가진 회사 이름을 찾으세요.
  3. 특정 회사 ( ' Apple ' )의 최저 저가와 최고 고가를 찾으시오.
  4. ' Apple ' 회사의 일별 가격 변동률을 계산하고, 가장 큰 변동률을 가진 날짜를 찾으세요.

가격 변동률은

$(\text{Close} - \text{Open}) / \text{Open} \times 100\%$

# Quiz 데이터

```
import pandas as pd

#데이터 파일 읽기:
filename = 'stock_data.csv'
df = pd.read_csv(filename)
print("1", df.head)

# 가장 높은 평균 종가를 가진 회사 찾기
avg_close = df.groupby('Company')['Close'].mean()
sorted_avg_close = avg_close.sort_values(ascending=False)
highest_avg_close_company = sorted_avg_close.index[0]
print("2", highest_avg_close_company)

# 'Apple' 회사의 최저 저가와 최고 고가 찾기
apple_data = df[df['Company'] == 'Apple']
lowest_low = apple_data['Low'].min()
highest_high = apple_data['High'].max()
print("3", lowest_low, highest_high)

# 'Apple' 회사의 일별 가격 변동을 계산 및 가장 큰 변동을 가진 날짜 찾기:
apple_data = df[df['Company'] == 'Apple']
apple_data['Change'] = (apple_data['Close'] - apple_data['Open']) / apple_data['Open'] * 100
sorted_change = apple_data['Change'].sort_values(ascending=False)
max_change_date = apple_data.loc[sorted_change.index[0], 'Date']
print("4", max_change_date)
```

# Quiz 데이터 실행 결과

```
1 <bound method NDFrame.head of
0    2023-01-01    Apple  137.454012  138.233984  126.044793  126.335211
1    2023-01-01    Google  186.617615  192.812211  183.077252  189.806636
2    2023-01-01  Microsoft  102.058449  102.975472   94.728457   96.249668
3    2023-01-02    Apple  152.475643  154.635368  145.767799  147.223945
4    2023-01-02    Google  161.185289  163.017099  154.624479  156.904829
..      ..      ..      ..      ..      ..      ..
295  2023-04-09    Google  129.166258  132.641167  117.984049  122.284661
296  2023-04-09  Microsoft  177.985099  178.509750  176.533104  177.743329
297  2023-04-10    Apple  198.666259  199.378737  190.461771  192.956212
298  2023-04-10    Google  161.815573  167.113315  160.183267  167.064461
299  2023-04-10  Microsoft  151.771164  170.026748  150.018030  169.587415
```

```
[300 rows x 6 columns]>
```

```
2 Google
3 73.54412574081596 209.2434840871398
4 2023-03-28
```

감사합니다  
[2차시]

# 데이터 시각화

## 데이터 시각화

---

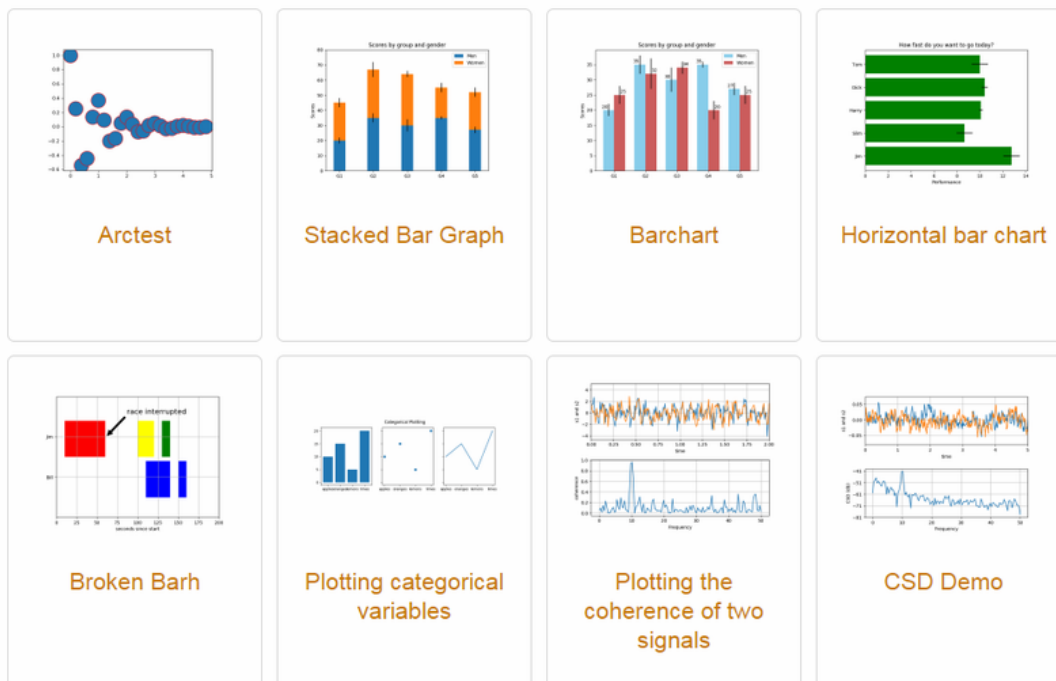
1. 많은 양의 데이터를 한눈에 볼 수 있다.
2. 데이터 분석에 대한 전문 지식이 없어도, 누구나 쉽게 데이터 인사이트를 찾을 수 있다.
3. 요약 통계보다 정확한 데이터 분석 결과를 도출할 수 있다.
4. 효과적인 데이터 인사이트 공유로 데이터 기반의 의사결정을 할 수 있다.



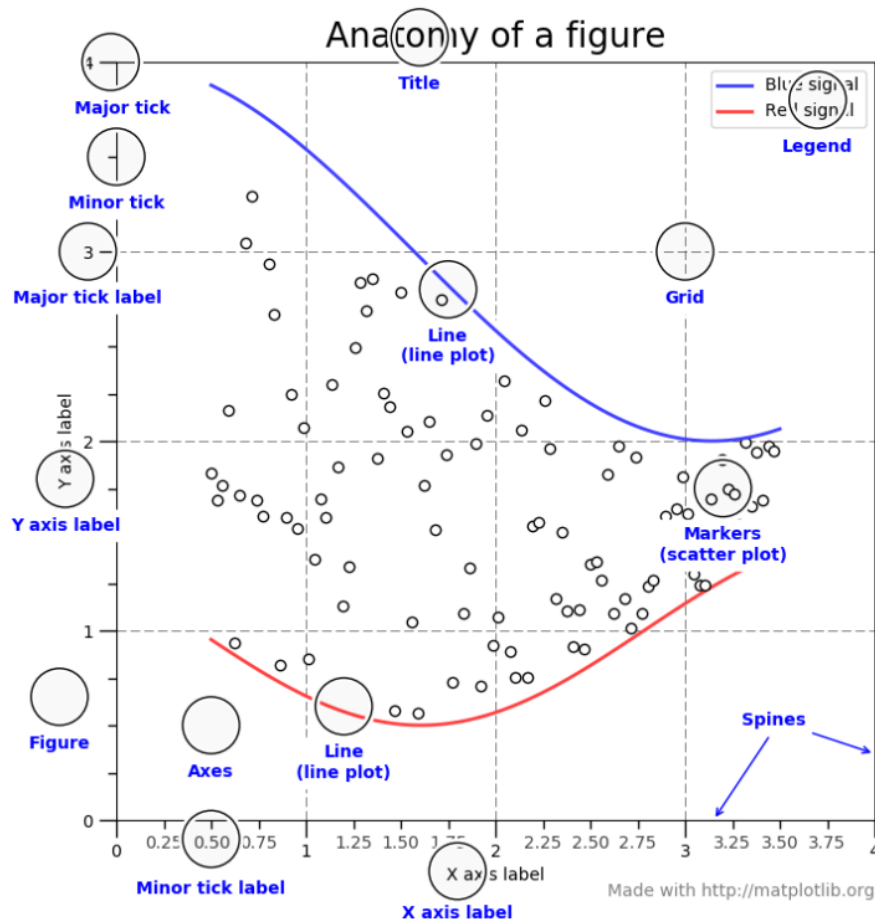
# Matplotlib

Matplotlib은 데이터 시각화 파이썬 라이브

최근 가장 많이 사용되는 시각화 파이썬 라이브러리



# Matplotlib : 구성요소

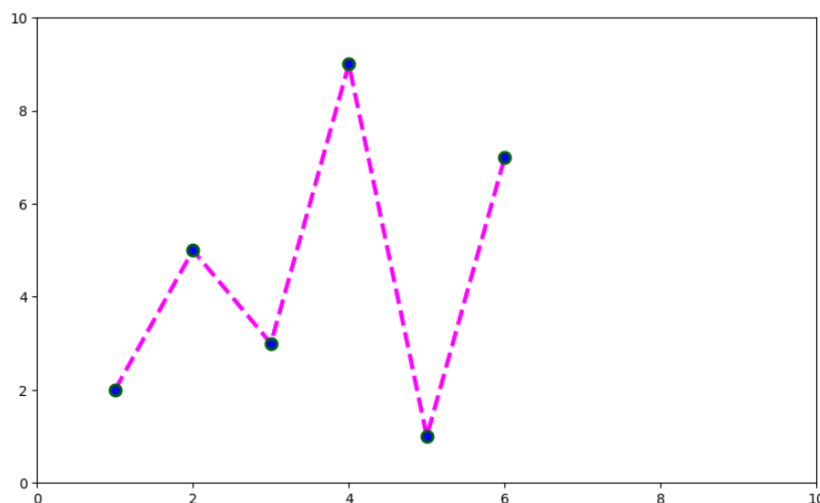


1. Figure : 그래프가 그려질 공간
2. Axes : 그래프가 그려지는 영역
3. X axis : X 축
4. Y axis : Y 축
5. Tick : 눈금
6. Spines : 그래프 테두리
7. Line : 선
8. Markers : 점
9. Grid : 그래프 격자
10. Title : 그래프 제목
11. Label : 그래프의 이름
12. Legend : 범례

# Matplotlib : 예제

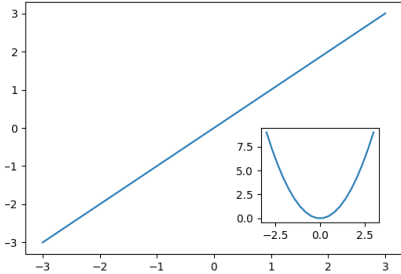
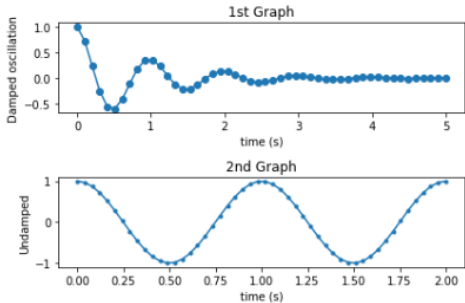
```
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure(figsize = (10, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set(xlim = [0, 10], ylim = [0, 10])
line, = ax.plot([1, 2, 3, 4, 5, 6], [2, 5, 3, 9, 1, 7])
line.set(color = 'magenta', linewidth = 3,
          linestyle = '--', marker = 'o',
          markersize = 8, markeredgewidth = 2,
          markerfacecolor = 'blue', markeredgecolor = 'darkgreen')
plt.show()
```



## 2. 데이터 시각화

# Matplotlib : interface

object-oriented	State-based
<p>figure와 axes 객체를 직접 생성하고 이들의 메소드를 사용하여 그래프를 그림</p>	<p>현재 figure나 현재 axes와 같은 "현재 상태"에 의존하는 함수</p>
<pre>import matplotlib.pyplot as plt  x = [1, 2, 3, 4, 5] y = [2, 3, 5, 7, 11]  fig, ax = plt.subplots() # figure와 axes 객체 생성 ax.plot(x, y) # axes 객체의 메소드를 사용하여 플롯 생성 plt.show()</pre>	<pre>import matplotlib.pyplot as plt  x = [1, 2, 3, 4, 5] y = [2, 3, 5, 7, 11]  plt.plot(x, y) plt.show()</pre>
<p>Plot이 복잡해질수록 강력하고 유연해짐</p>  <p>The figure displays a main plot with a linear relationship (y = x) and an inset plot showing a parabolic relationship (y = x^2). The main plot has x-axis from -3 to 3 and y-axis from -3 to 3. The inset plot has x-axis from -2.5 to 2.5 and y-axis from 0.0 to 7.5.</p>	<p>간단하지만 현재 상태의 figure와, axe에 의존 따로 정의하여 변경은 가능</p>  <p>The figure contains two subplots. The top subplot, titled '1st Graph', shows 'Damped oscillation' with 'time (s)' on the x-axis (0 to 5) and amplitude on the y-axis (-0.5 to 1.0). The bottom subplot, titled '2nd Graph', shows 'Undamped' oscillation with 'time (s)' on the x-axis (0.00 to 2.00) and amplitude on the y-axis (-1 to 1).</p>

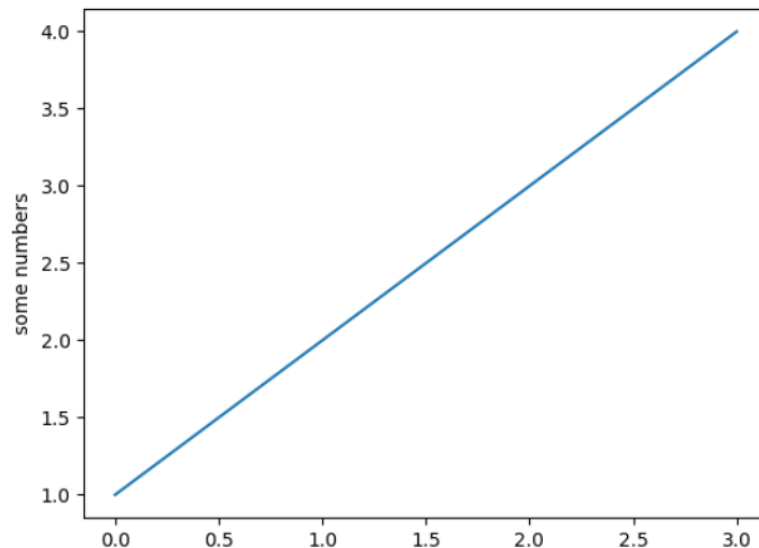
inline 실행

## JUPYTER 내에서 그래프 보기

%matplotlib inline 명령을 먼저 실행해야  
jupyter notebook 내에서 그래프가 보임

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

plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



## notebook 실행

## notebook 실행

%matplotlib notebook 명령을 먼저 실행해야  
jupyter notebook 내에 qt 창이 실행

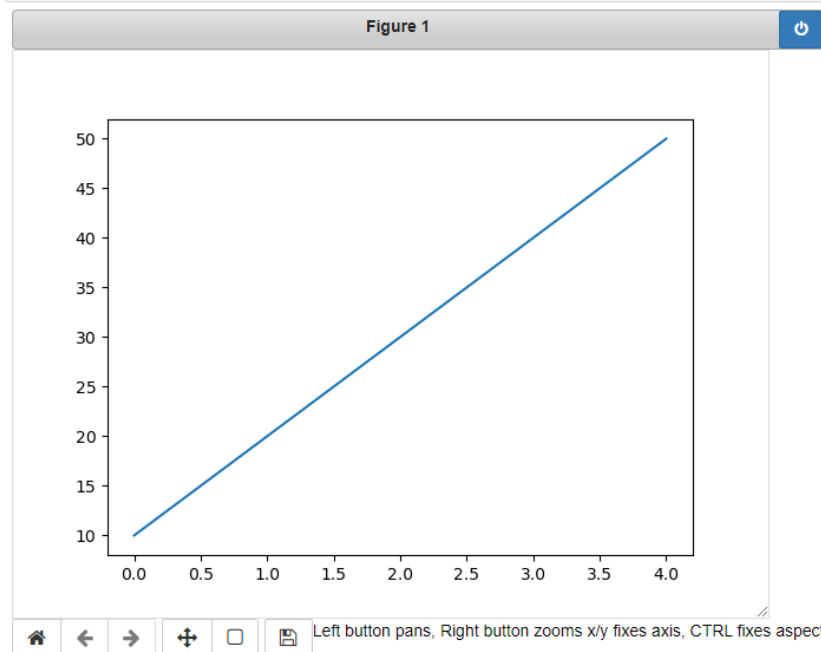
```
In [1]: %matplotlib notebook
```

```
In [2]: import matplotlib.pyplot as plt
```

```
y = [10, 20, 30, 40, 50]
```

```
plt.plot(y)
```

```
plt.show()
```



Inline을 다시 돌아가려  
면 %matplotlib inline을  
실행시켜주면 됨

# Axes 객체 생성 : plt.subplot

## Axes 객체 생성

subplot 함수를 이용해서 Axes 객체를 생성

```
import matplotlib.pyplot as plt

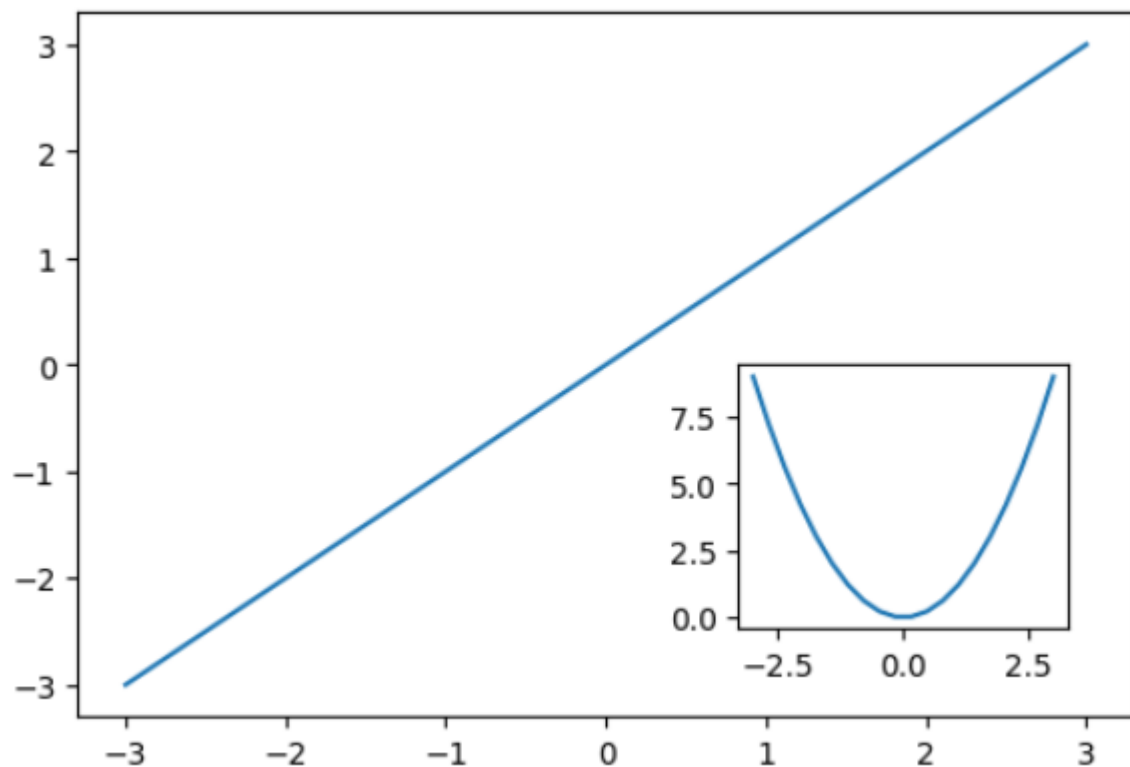
x = np.linspace(-3, 3, 20)
y1 = x
y2 = x**2

fig = plt.figure(figsize = (6, 4))

plt.axes([0.1, 0.1, 0.8, 0.8])
plt.plot(x, y1)

plt.axes([0.6, 0.2, 0.25, 0.3])
plt.plot(x, y2)

plt.show()
```



## Plot 함수

# Plot 함수

선 그래프를 그리는 함수. Plot함수로 그리면  
Line2D class의 인스턴스가 생김

```
import matplotlib.pyplot as plt
```

```
help(plt.plot)
```

Help on function plot in module matplotlib.pyplot:

```
plot(*args, scalex=True, scaley=True, data=None, **kwargs)
```

Plot y versus x as lines and/or markers.

Call signatures::

```
plot([x], y, [fmt], *, data=None, **kwargs)
```

```
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

The coordinates of the points or line nodes are given by \*x\*, \*y\*.

The optional parameter \*fmt\* is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the \*Notes\* section below.

```
>>> plot(x, y)          # plot x and y using default line style and color
>>> plot(x, y, 'bo')    # plot x and y using blue circle markers
>>> plot(y)             # plot y using x as index array 0..N-1
>>> plot(y, 'r+')       # ditto, but with red plusses
```



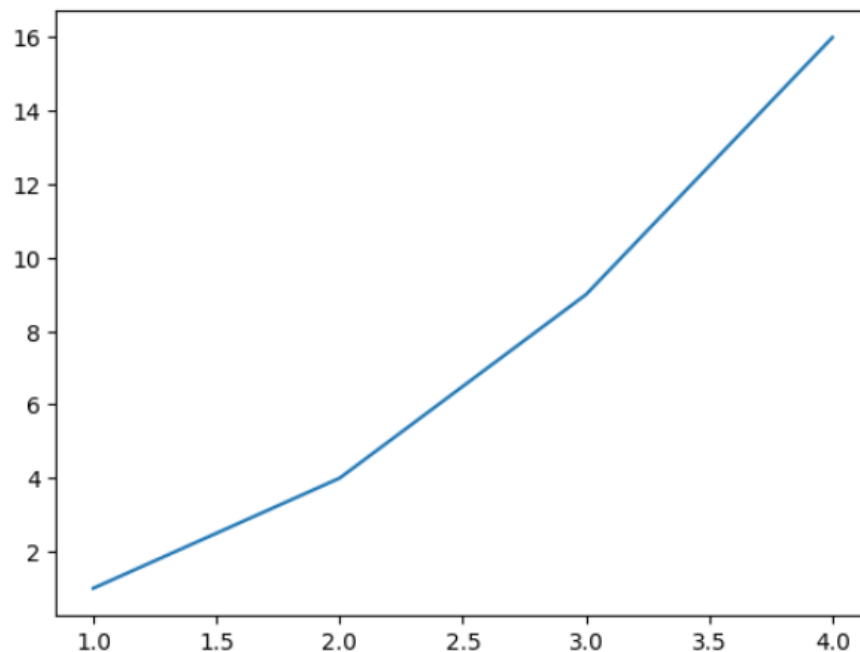
## plot 함수 : x축과 y축 1

## plot 함수 예제 1

x축과 y축 넣고 그래프 보기 (미리 Figure, axe가 호출되지 않았다면 plot함수가 자동 호출)

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
```



## sin/cos 함수 그래프

## plot 함수 예제 2

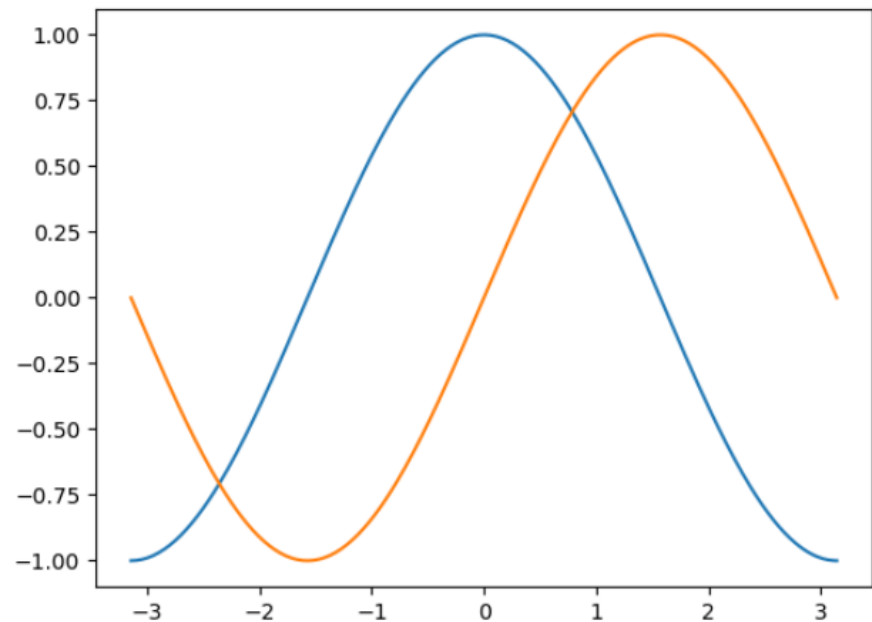
## 두 그래프를 동시에 그리기

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint = True)
C, S = np.cos(X), np.sin(X)

plt.plot(X, C)
plt.plot(X, S)

plt.show()
```



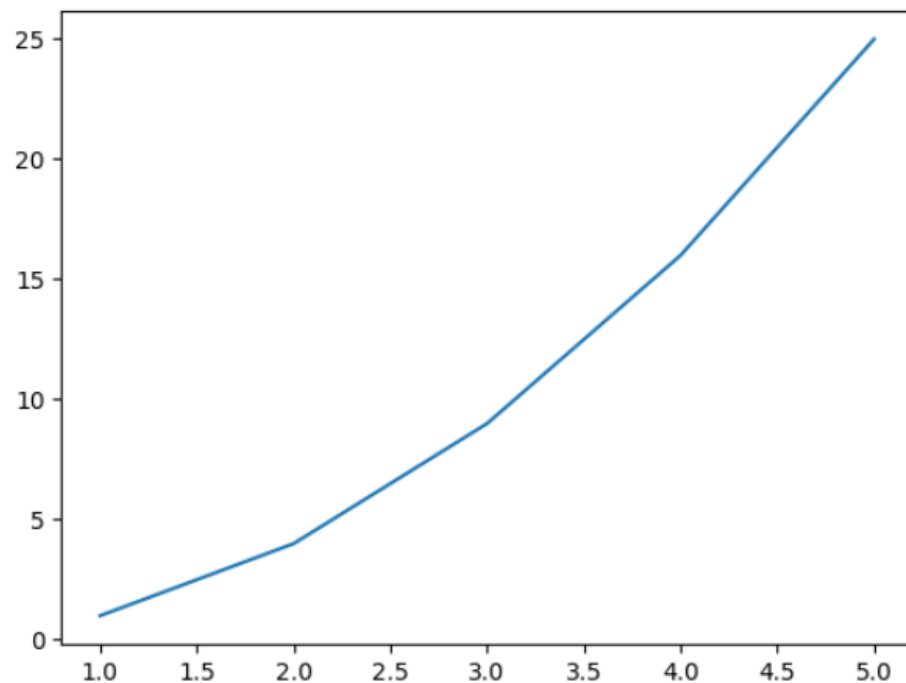
## plot 함수 결과 확인하기

plot 함수로 그린 결과를 조회하면 Line2D 인스턴스가 생김

```
import numpy as np
import matplotlib.pyplot as plt

# Make an array of x values
x = [1, 2, 3, 4, 5]
# Make an array of y values for each x value
y = [1, 4, 9, 16, 25]
# Use pyplot to plot x and y
mo = plt.plot(x, y)
print(mo)
# show the plot on the screen
plt.show()
```

[<matplotlib.lines.Line2D object at 0x7fc610f230d0>]



## 두 리스트를 매칭

x 와 y 축의 원소를 맞춰 정의한 후 plot 함수에 넣고 color, marker, linestyle을 부여

```
import numpy as np
import matplotlib.pyplot as plt

years = [x for x in range(1950, 2011, 10)]
print(years)

gdp = [y for y in np.random.randint(300, 10000, size = 7)]

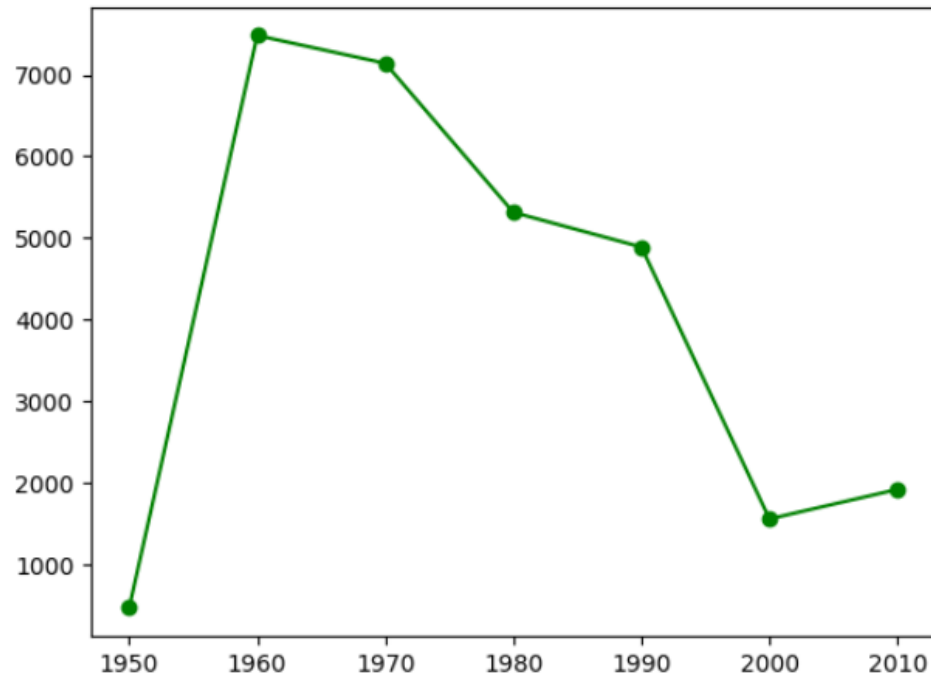
print(gdp)

plt.plot(years, gdp, color = 'green', marker = 'o', linestyle = 'solid')
plt.show()
```

## 두 리스트를 매칭 : 그래프

앞장의 선언대로 년도는 x축, gdp는 y축으로 구성되고  
맞는 접은 원, 각 점마다 선으로 연결된 그래프를 그림

[1950, 1960, 1970, 1980, 1990, 2000, 2010]  
[468, 7481, 7140, 5314, 4889, 1558, 1920]



## 2. 데이터 시각화

Line2D : line color -&gt; 문자

line color -&gt; 문자

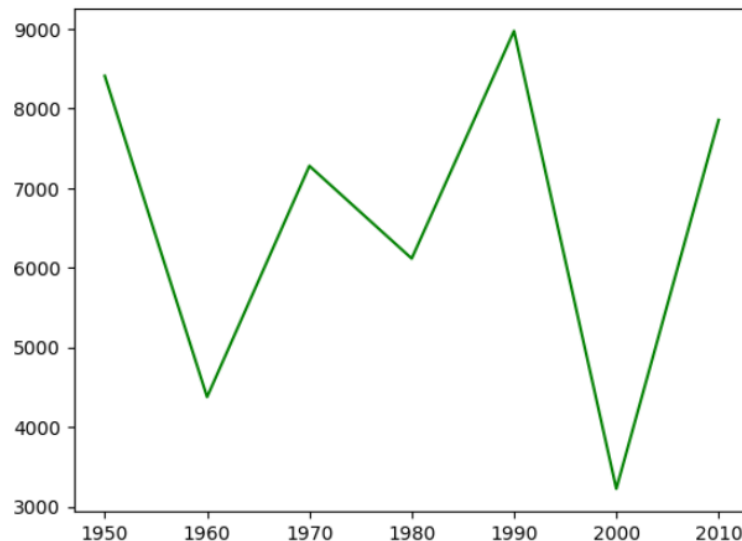
## Line2D에 대한 line color 처리

```
import numpy as np
import matplotlib.pyplot as plt

years = [x for x in range(1950, 2011, 10)]
gdp = [y for y in np.random.randint(300, 10000, size = 7)]

plt.plot(years, gdp, color = 'g')

plt.show()
```



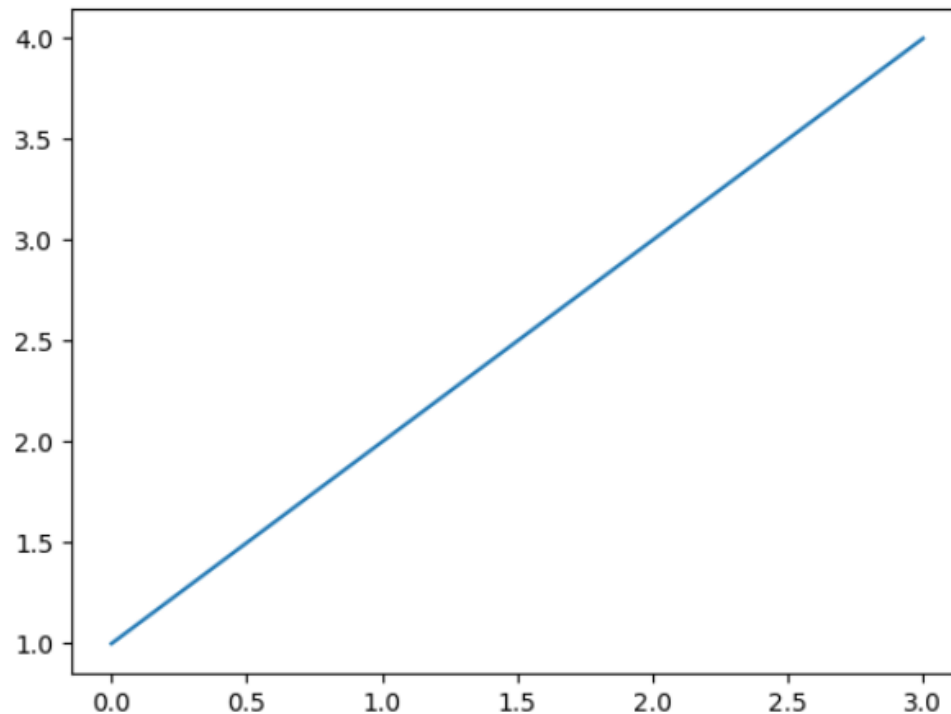
- b: blue
- g: green
- r: red
- c: cyan
- m: magenta
- y: yellow
- k: black
- w: white

# Line2D : line color defaults

## line color defaults

line color의 default 값은 blue

```
import matplotlib.pyplot as plt  
  
plt.plot([1, 2, 3, 4])  
plt.show()
```



## 2. 데이터 시각화

## plot 함수 : linewidth

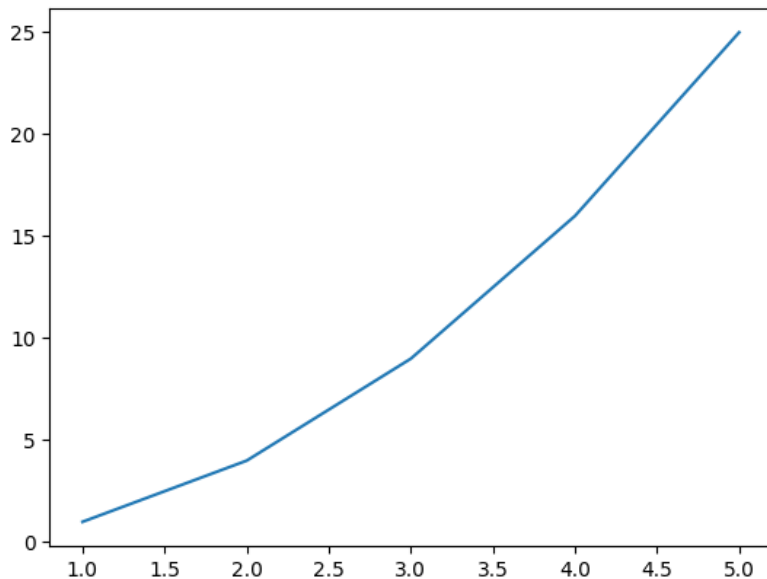
## plot 함수 : linewidth

Plot 함수에 line을 굵게 하려면 linewidth에 값을 부여

```
import numpy as np
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
plt.plot(x, y)

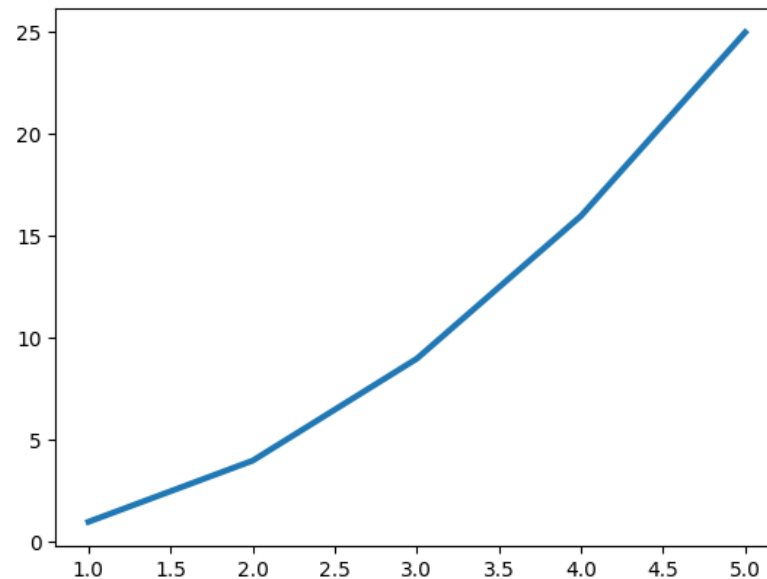
plt.show()
```



```
import numpy as np
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
plt.plot(x, y, linewidth = 3.0)

plt.show()
```





## Line2D : marker

## marker

## Line2D에 대한 marker 처리

marker	description
"."	point
","	pixel
"o"	circle
"v"	triangle_down
"^"	triangle_up
"<"	triangle_left
">"	triangle_right
"1"	tri_down
"2"	tri_up
"3"	tri_left
"4"	tri_right
"8"	octagon
"s"	square
"p"	pentagon
"*"	star
"h"	hexagon1
"H"	hexagon2
"+"	plus
"x"	x
"D"	diamond
"d"	thin_diamond

marker	description
" "	vline
"_"	hline
TICKLEFT	tickleft
TICKRIGHT	tickright
TICKUP	tickup
TICKDOWN	tickdown
CARETLEFT	caretleft
CARETRIGHT	caretright
CARETUP	caretup
CARETDOWN	caretdown
"None"	nothing
None	nothing
" "	nothing
" "	nothing
'\$...\$'	render the string using mathtext.
verts	a list of (x, y) pairs used for Path vertices. The center of the marker is located at (0,0) and the size is normalized.
path	a <a href="#">Path</a> instance.
(numsides, style, angle)	see below

## marker 처리 예시

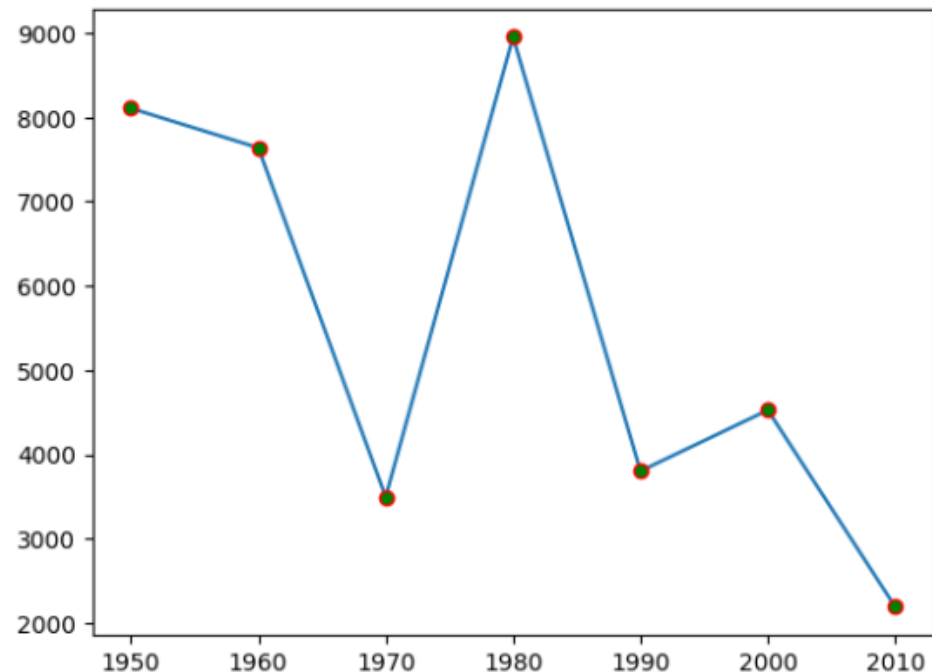
## marker에 대한 정보를 세팅해서 처리

```
import numpy as np
import matplotlib.pyplot as plt

years = [x for x in range(1950, 2011, 10)]
gdp = [y for y in np.random.randint(300, 10000, size = 7)]

plt.plot(years, gdp, marker = 'o',
         markersize = 6, markeredgewidth = 1,
         markeredgewidth = 'red', markerfacecolor = 'green')

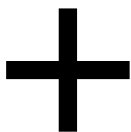
plt.show()
```



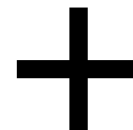
# color와 style 조합 표시하기

line color와 linestyle을 조합해서 문자열로 조합해서 처리하기

- b: blue
- g: green
- r: red
- c: cyan
- m: magenta
- y: yellow
- k: black
- w: white



marker	description
"."	point
","	pixel
"o"	circle
"v"	triangle_down
"^"	triangle_up
"<"	triangle_left
">"	triangle_right
"1"	tri_down
"2"	tri_up
"3"	tri_left
"4"	tri_right
"8"	octagon
"s"	square
"p"	pentagon
"*"	star
"h"	hexagon1
"H"	hexagon2
"+"	plus
"x"	x
"D"	diamond
"d"	thin_diamond



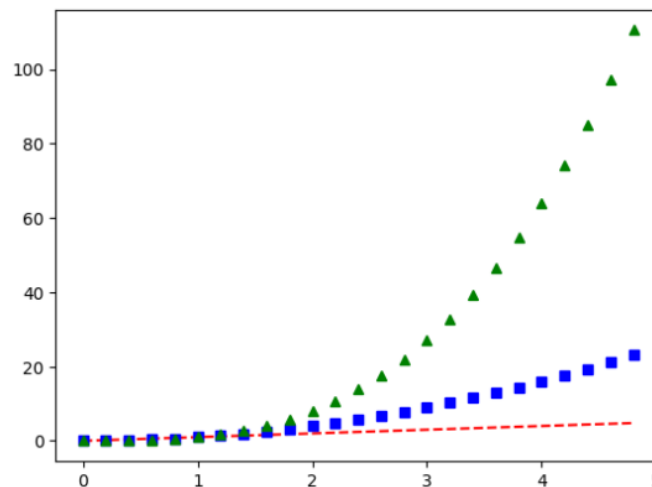
linestyle	description
"" or "solid"	solid line
"--" or "dashed"	dashed line
"-." or "dashdot"	dash-dotted line
"..." or "dotted"	dotted line
"None"	draw nothing
.....	draw nothing

## 여러 선에 색과 스타일 조합

‘r—’: 빨간색과 — 조합, ‘bs’:는 파란색과 사각형 조합, ‘g^’: 초록색과 삼각형 조합

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



$y = x$   
 $y = x^{**2}$   
 $y = x^{**3}$   
 에 대한 함수의  
 그래프를 표현

## axis 함수 이해하기

## axis 함수 이해하기

axis 함수는 리스트의 값을 그대로 표시하고  
앞의 2자리는 x축, 뒤에 2자리는 y축을 표시

```
import matplotlib.pyplot as plt
```

```
help(plt.axis)
```

Help on function axis in module matplotlib.pyplot:

```
axis(arg=None, /, *, emit=True, **kwargs)
```

Convenience method to get or set some axis properties.

Call signatures::

```
xmin, xmax, ymin, ymax = axis()
xmin, xmax, ymin, ymax = axis([xmin, xmax, ymin, ymax])
xmin, xmax, ymin, ymax = axis(option)
xmin, xmax, ymin, ymax = axis(**kwargs)
```

Parameters

-----

xmin, xmax, ymin, ymax : float, optional

The axis limits to be set. This can also be achieved using ::

```
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
```

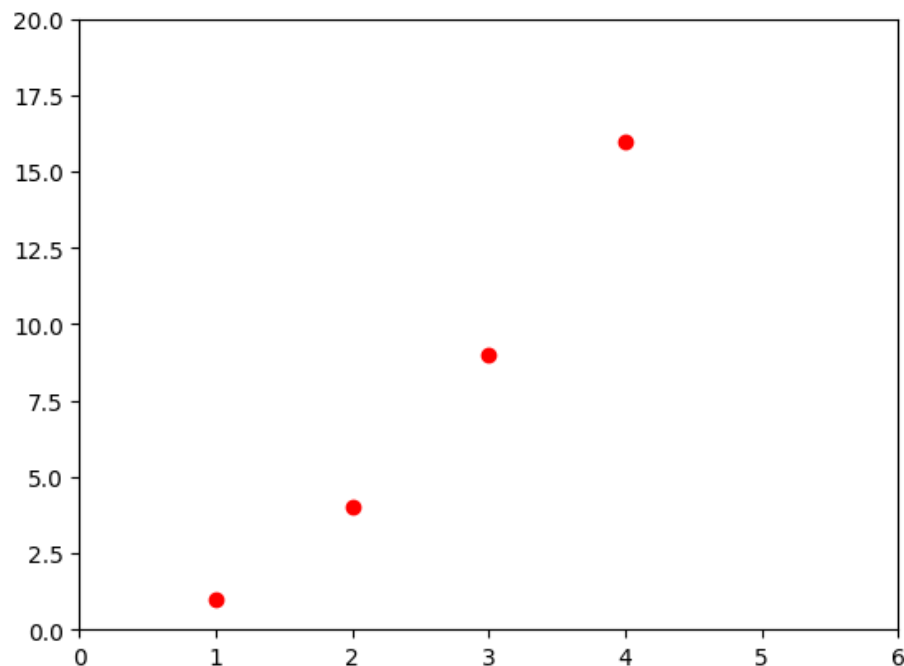
## axis 함수 실행 예시

## axis 함수 실행 예시

x축을 0,6으로 제한하고 y축을 0,20으로 제한

```
import matplotlib.pyplot as plt

a = plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
b = plt.axis([0, 6, 0, 20])
plt.show()
print(a)
print(type(b), b)
```



```
[<matplotlib.lines.Line2D object at 0x7fc5f18de200>]  
<class 'tuple'> (0.0, 6.0, 0.0, 20.0)
```

## xticks 함수 : x축 넣기

## xticks 함수 : x축 넣기

## xticks 함수를 이용해서 세부 값을 부여

```
import matplotlib.pyplot as plt
```

```
help(plt.xticks)
```

Help on function xticks in module matplotlib.pyplot:

```
xticks(ticks=None, labels=None, *, minor=False, **kwargs)
```

Get or set the current tick locations and labels of the x-axis.

Pass no arguments to return the current values without modifying them.

Parameters

-----  
ticks : array-like, optional

The list of xtick locations. Passing an empty list removes all xticks.

labels : array-like, optional

The labels to place at the given \*ticks\* locations. This argument can only be passed if \*ticks\* is passed as well.

minor : bool, default: False

If ``False``, get/set the major ticks/labels; if ``True``, the minor ticks/labels.

\*\*kwargs

`.Text` properties can be used to control the appearance of the labels.

## yticks 함수 : y축 넣기

### yticks 함수를 이용해서 세부 값을 부여

```
import matplotlib.pyplot as plt
```

```
help(plt.yticks)
```

Help on function yticks in module matplotlib.pyplot:

```
yticks(ticks=None, labels=None, *, minor=False, **kwargs)
```

Get or set the current tick locations and labels of the y-axis.

Pass no arguments to return the current values without modifying them.

Parameters

ticks : array-like, optional

The list of ytick locations. Passing an empty list removes all yticks.

labels : array-like, optional

The labels to place at the given \*ticks\* locations. This argument can only be passed if \*ticks\* is passed as well.

minor : bool, default: False

If ``False``, get/set the major ticks/labels; if ``True``, the minor ticks/labels.

\*\*kwargs

`.Text` properties can be used to control the appearance of the labels.



## 좌표축 범위 제한

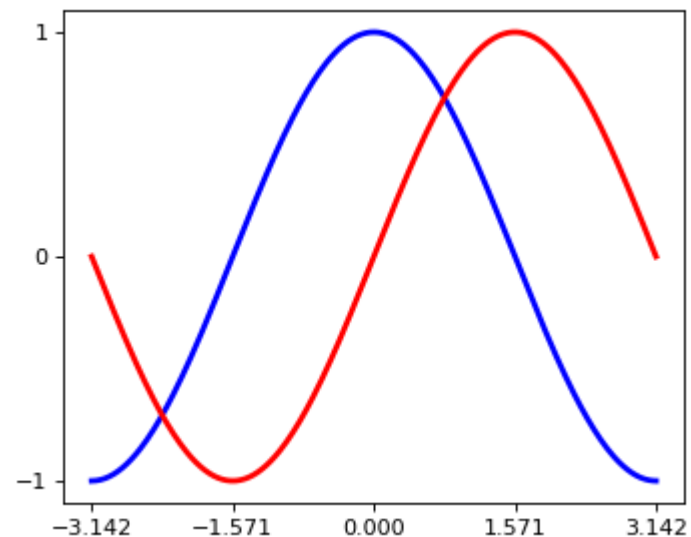
## 좌표 축 범위 제한

xticks, yticks를 이용해서 좌표축 범위 제한

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint = True)
C, S = np.cos(X), np.sin(X)
plt.figure(figsize = (5, 4), dpi = 80)
plt.plot(X, C, color = 'blue', linewidth = 2.5, linestyle = "-")
plt.plot(X, S, color = 'red', linewidth = 2.5, linestyle = "-")

plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.yticks([-1, 0, +1])
plt.show()
```



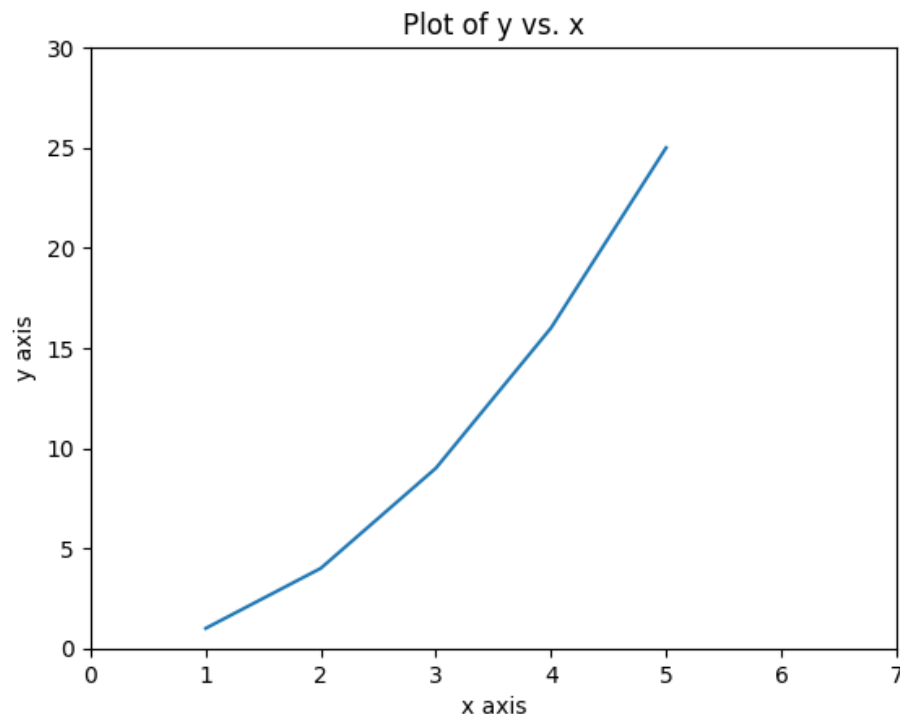
## lim 함수 : 축 넣기

## lim 함수 : 축 넣기

xlim, ylim 함수를 이용해서 축내의 범위 값을 부여

```
import numpy as np
import matplotlib.pyplot as plt

# Make an array of x values
x = [1, 2, 3, 4, 5]
# Make an array of y values for each x value
y = [1, 4, 9, 16, 25]
# Use pyplot to plot x and y
plt.plot(x, y)
# Give plot a title
plt.title('Plot of y vs. x')
# Make axis Labels
plt.xlabel('x axis')
plt.ylabel('y axis')
# Set axis limits
plt.xlim(0.0, 7.0)
plt.ylim(0.0, 30.)
# Show the plot on the screen
plt.show()
```



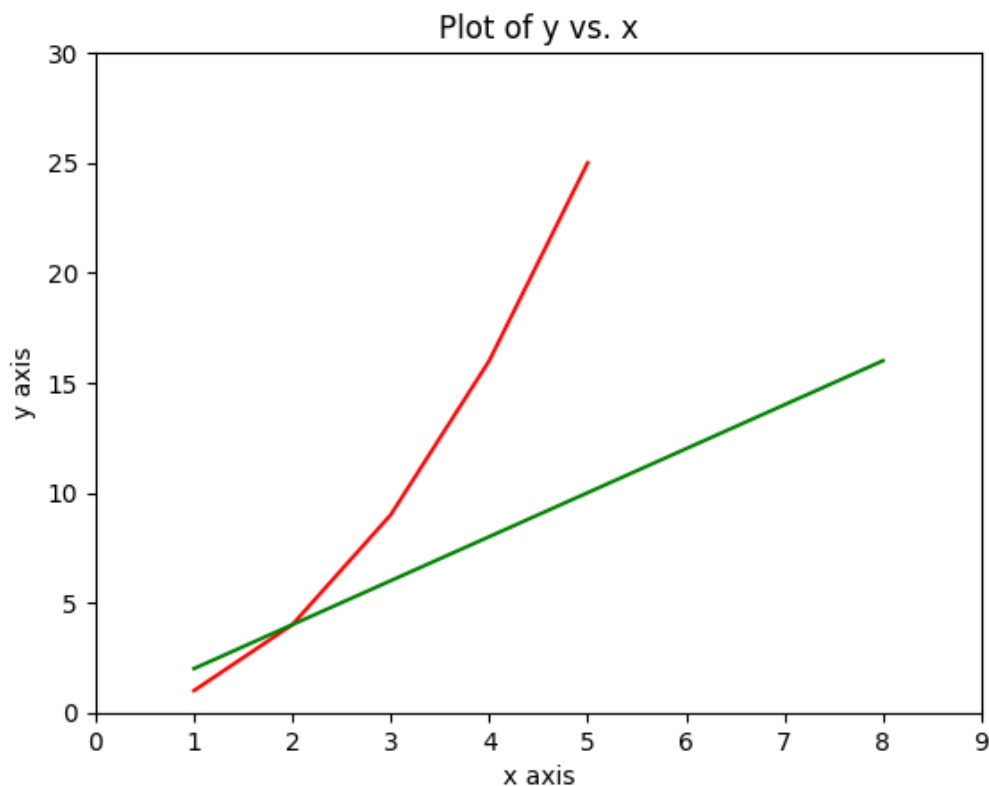
## Xlabel/ylabel

## xlabel/ylabel

## x축과 y축 label 붙이기

```
import numpy as np
import matplotlib.pyplot as plt

# Make x, y arrays for each graph
x1 = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
x2 = [1, 2, 4, 6, 8]
y2 = [2, 4, 8, 12, 16]
# Use pyplot to plot x and y
plt.plot(x1, y1, 'r')
plt.plot(x2, y2, 'g')
# Give plot a title
plt.title('Plot of y vs. x')
# Make axis Labels
plt.xlabel('x axis')
plt.ylabel('y axis')
# Set axis limits
plt.xlim(0.0, 9.0)
plt.ylim(0.0, 30.)
# Show the plot on the screen
plt.show()
```



## xlabel 함수 : font/color

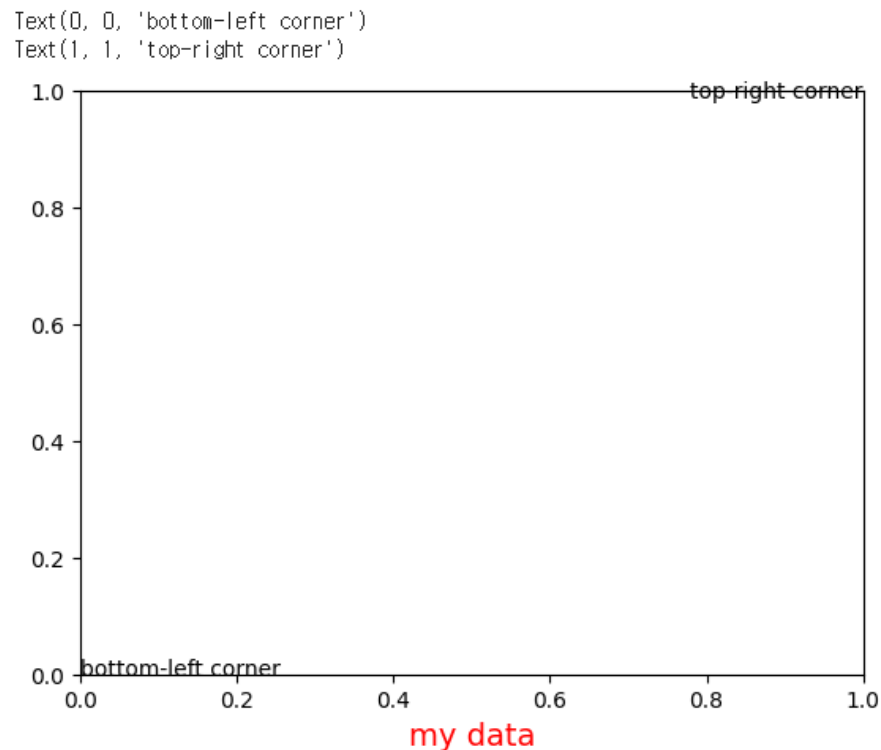
## xlabel 함수 : font/color

x축 그래프에 label에 fontsize와 font color 변 경하기

```
import matplotlib.pyplot as plt

txt = plt.text(0, 0, 'bottom-left corner')
txt1 = plt.text(1, 1, 'top-right corner', ha = 'right', va = 'center')
print(txt)
print(txt1)
t = plt.xlabel('my data', fontsize = 14, color = 'red')

plt.show()
```



## 2. 데이터 시각화

## plot 함수 : label

## plot 함수 : label

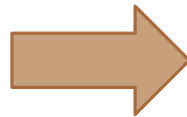
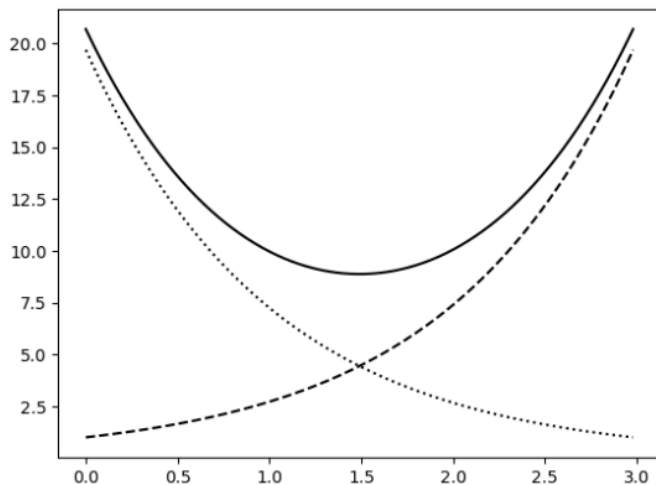
## Plot 함수에 legend함수 처리를 위해 label을 정의

```
import numpy as np
import matplotlib.pyplot as plt

# Make same fake data
a = b = np.arange(0, 3, .02)
c = np.exp(a)
d = c[::-1]

# Create plots with pre-defined labels
plt.plot(a, c, 'k--', label = 'Model length')
plt.plot(a, d, 'k:', label = 'Data length')
plt.plot(a, c + d, 'k', label = 'Total message length')

plt.show()
```



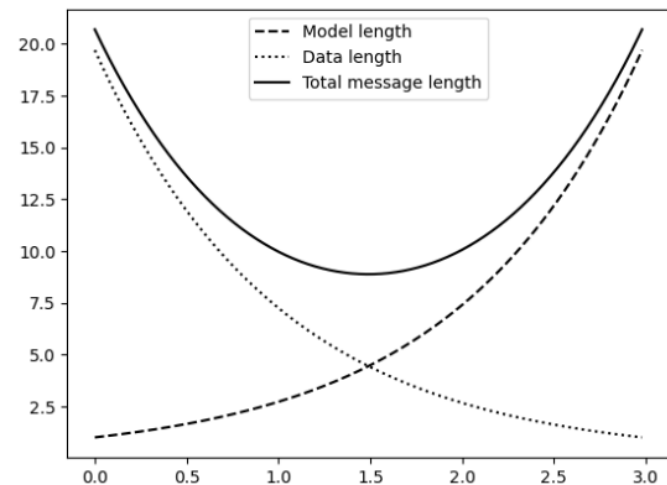
legend 함수  
호출하면 범  
주 표시

```
import numpy as np
import matplotlib.pyplot as plt

# Make same fake data
a = b = np.arange(0, 3, .02)
c = np.exp(a)
d = c[::-1]

# Create plots with pre-defined labels
plt.plot(a, c, 'k--', label = 'Model length')
plt.plot(a, d, 'k:', label = 'Data length')
plt.plot(a, c + d, 'k', label = 'Total message length')

plt.legend()
plt.show()
```



## plot(label) 이용 : 1

## Legend 함수

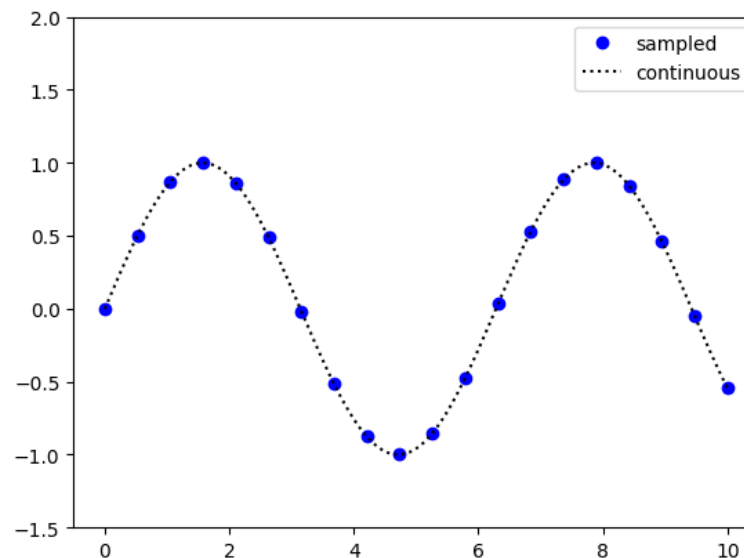
plot함수의 label을 이용해서 그래프에 범주를 표시

```
import numpy as np
import matplotlib.pyplot as plt

x1 = np.linspace(0, 10, 20)
y1 = np.sin(x1)

x2 = np.linspace(0, 10, 1000)
y2 = np.sin(x2)
plt.plot(x1, y1, 'bo', label = 'sampled')
plt.plot(x2, y2, ':k', label = 'continuous')
plt.legend()

plt.ylim(-1.5, 2.0)
plt.show()
```



## 범주 위치 지정

## Legend 함수

## legend 생성시 위치 배정 및 색깔 입히기

```
import numpy as np
import matplotlib.pyplot as plt

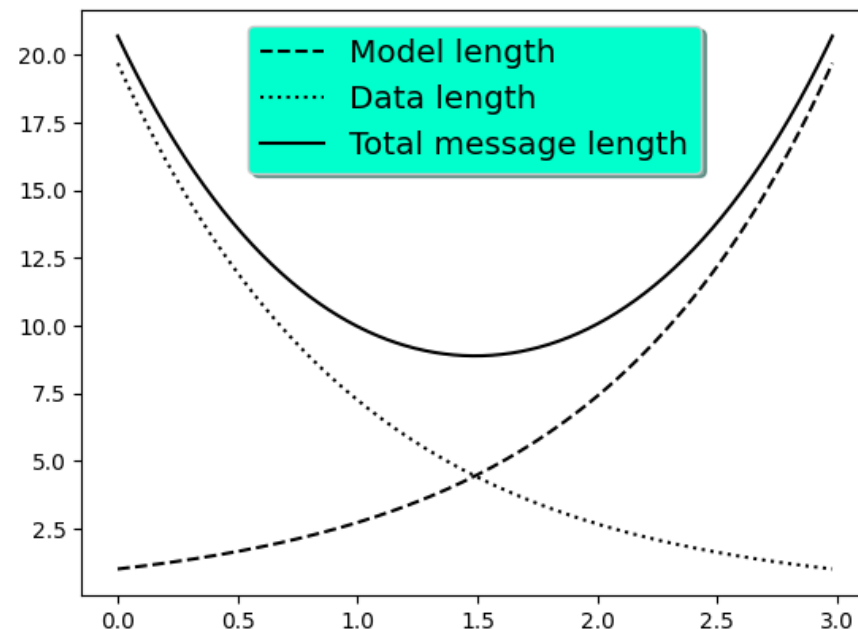
# Make some fake data
a = b = np.arange(0, 3, .02)
c = np.exp(a)
d = c[::-1]

# Create plots with pre-defined Labels
plt.plot(a, c, 'k--', label = 'Model length')
plt.plot(a, d, 'k:', label = 'Data length')
plt.plot(a, c + d, 'k', label = 'Total message length')

legend = plt.legend(loc = 'upper center', shadow = True, fontsize = 'x-large')

# Put a nicer background color on the legend
legend.get_frame().set_facecolor('#00FFCC')

plt.show()
```



## plot(label) 이용 : 2

## Legend 함수

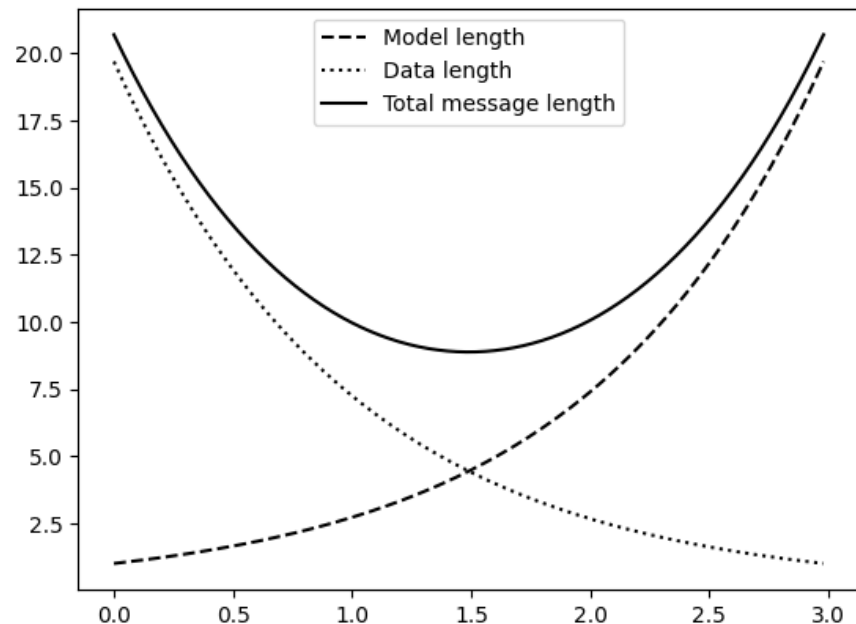
plot함수의 label을 이용해서 그래프에 범주를 표시

```
import numpy as np
import matplotlib.pyplot as plt

# Make some fake data
a = b = np.arange(0, 3, .02)
c = np.exp(a)
d = c[::-1]

# Create plots with pre-defined Labels
plt.plot(a, c, 'k--', label = 'Model length')
plt.plot(a, d, 'k:', label = 'Data length')
plt.plot(a, c + d, 'k', label = 'Total message length')

plt.legend()
plt.show()
```





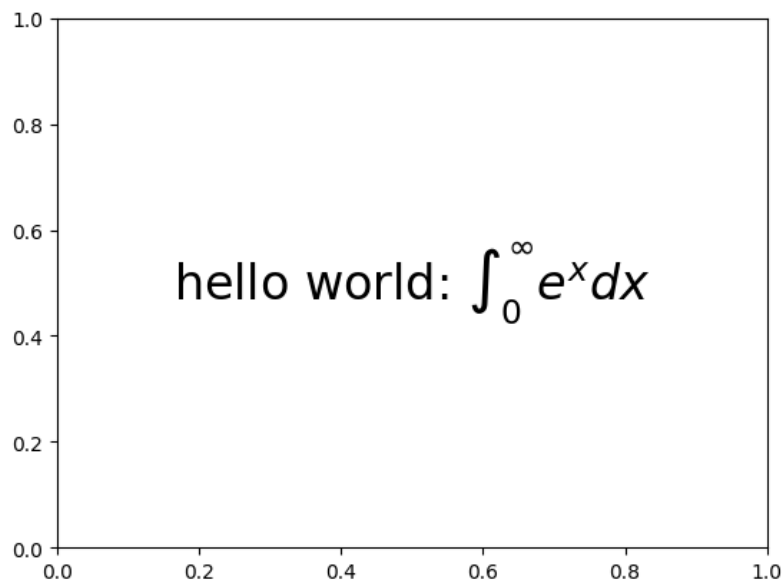
## text 함수 : 기초

## text 함수

그래프 내에 특정 좌표에 문자열이 들어가도록 입력해서 표시

```
import numpy as np
import matplotlib.pyplot as plt

plt.text(0.5, 0.5, 'hello world:  $\int_0^{\infty} e^x dx$ ', size = 24, ha = 'center', va = 'center')
plt.show()
```



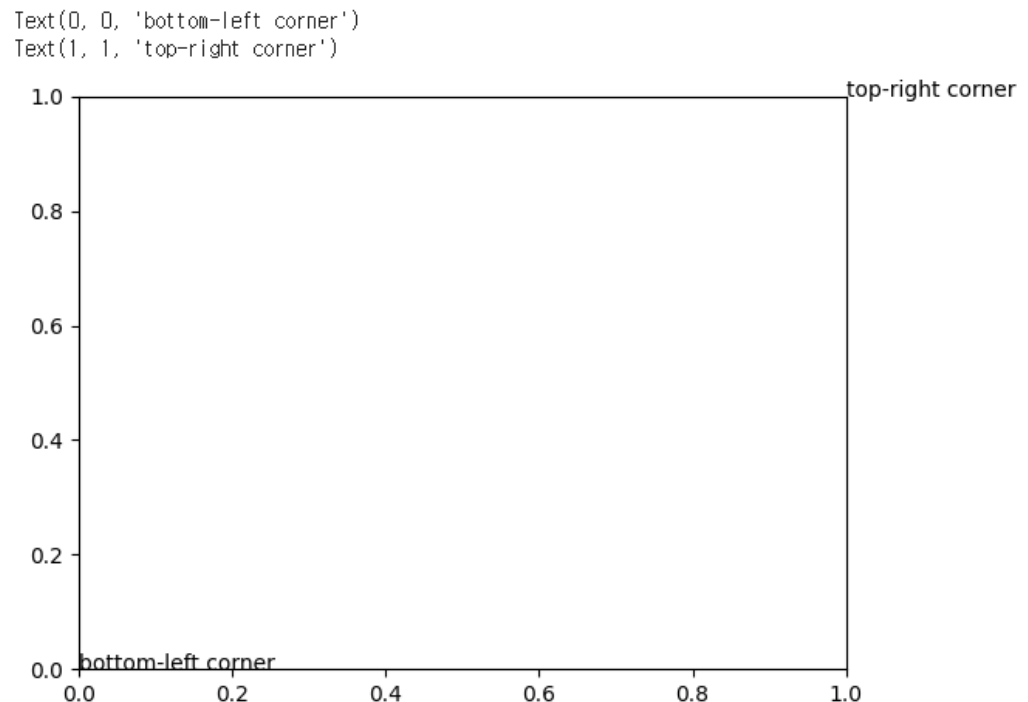
# text 함수 : 좌표에 따른 표시

## text 함수

text함수는 Text 클래스의 객체를 생성하고 그 위치 값을 좌표로 해서 문자열을 출력함

```
import matplotlib.pyplot as plt

txt = plt.text(0, 0, 'bottom-left corner')
txt1 = plt.text(1, 1, 'top-right corner')
print(txt)
print(txt1)
plt.show()
```



## 2. 데이터 시각화

text 함수 : text 붙이기

## text 함수 : text 붙이기

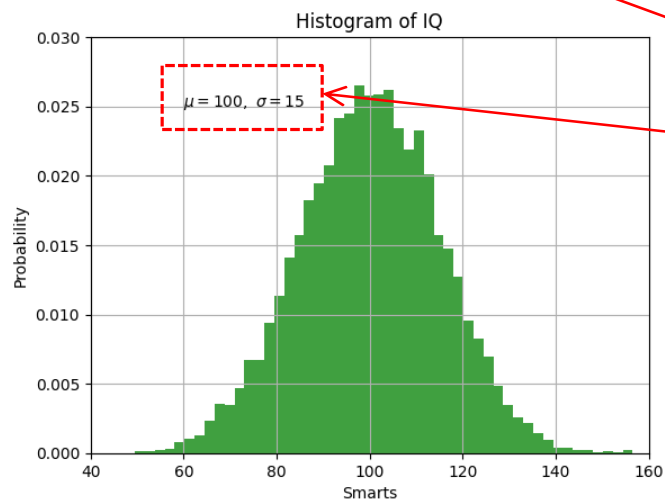
## 그래프 내에 text를 사용해서 입력하기

```

import numpy as np
import matplotlib.pyplot as plt

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
# the histogram of the data
n, bins, patches = plt.hist(x, 50, density = True, facecolor = 'g', alpha = 0.75)
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100, \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()

```



텍스트에 대해 입력

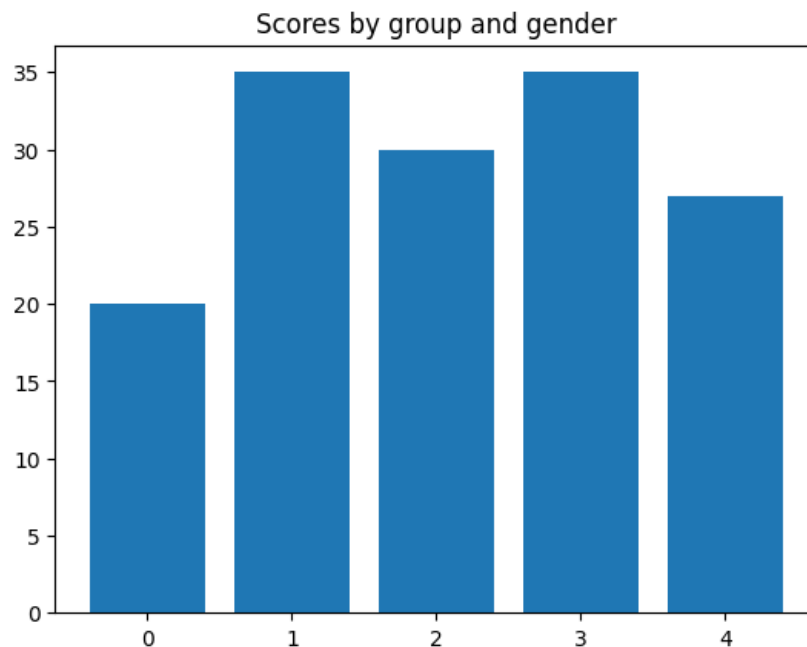
# title 함수 : 제목 붙이기

## 그래프에 제목을 표시

```
import numpy as np
import matplotlib.pyplot as plt

N = 5
menMeans = (20, 35, 30, 35, 27)
ind = np.arange(N)
print(ind)
plt.bar(ind, menMeans)
plt.title('Scores by group and gender')
plt.show()
```

[0 1 2 3 4]



# annotate 함수 : 기초

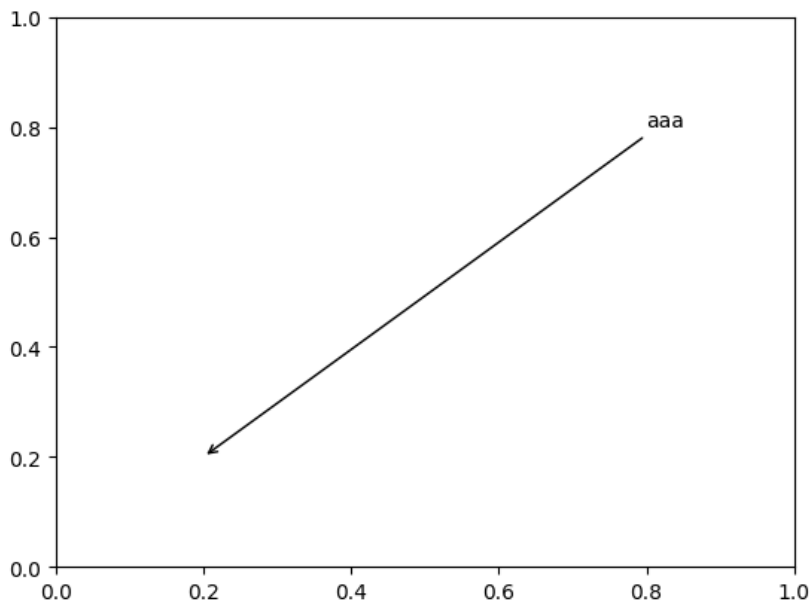
## annotate 함수

annotate 함수는 문장열, xy(화살표 끝 지시), xytext(문자열 시작 위치), arrowprops(화살표) 그래프에 주석을 표시

```
import matplotlib.pyplot as plt

plt.annotate("aaa",
             xy = (0.2, 0.2), xycoords = 'data',
             xytext = (0.8, 0.8), textcoords = 'data',
             arrowprops = dict(arrowstyle = "->", connectionstyle = "arc3"))

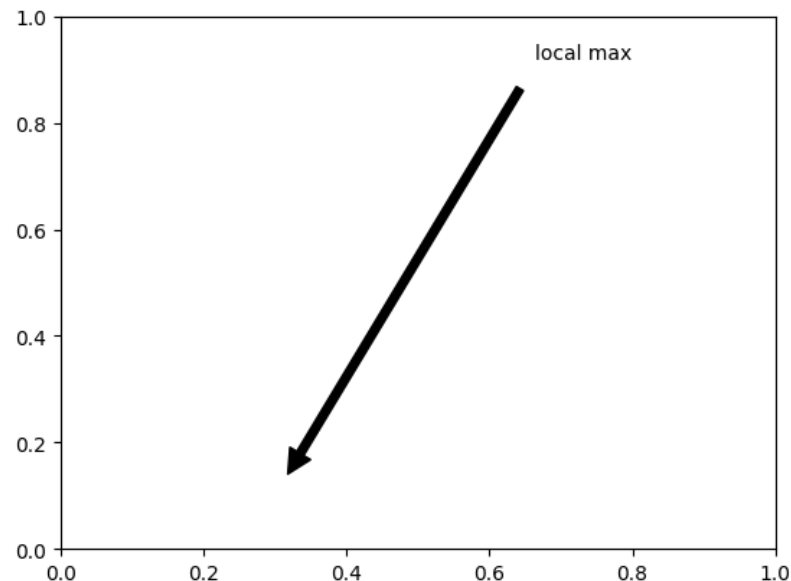
plt.show()
```



```
import matplotlib.pyplot as plt

plt.annotate('local max',
             xy = (0.3, 0.1), xycoords = 'data',
             xytext = (0.8, 0.95), textcoords = 'axes fraction',
             arrowprops = dict(facecolor = 'black', shrink = 0.05),
             horizontalalignment = 'right', verticalalignment = 'top')

plt.show()
```



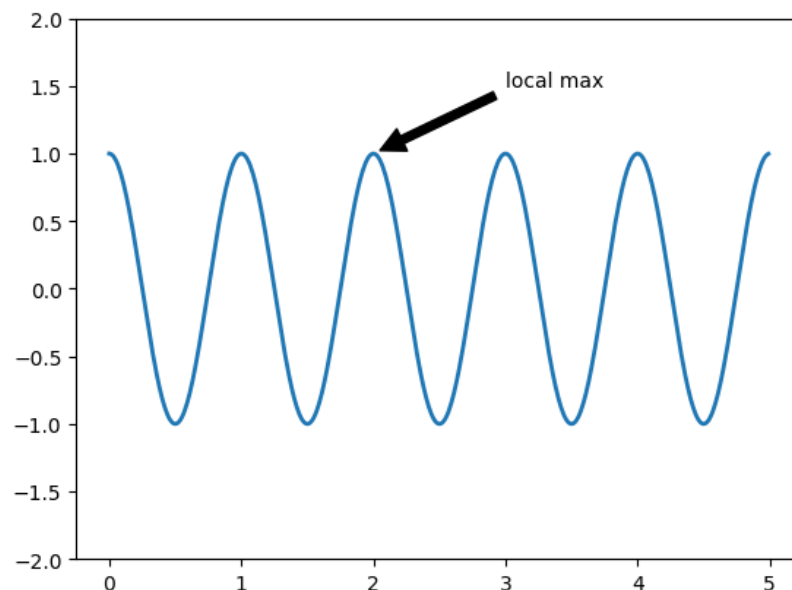
## annotate 함수 : 실행

## annotate 함수 : 실행

annotate 함수는 그래프에 주석을 표시

```
import numpy as np
import matplotlib.pyplot as plt

ax = plt.subplot(111)
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2 * np.pi * t)
line, = plt.plot(t, s, lw = 2)
plt.annotate('local max',
             xy = (2, 1), xytext = (3, 1.5),
             arrowprops = dict(facecolor = 'black', shrink = 0.05))
plt.ylim(-2, 2)
plt.show()
```



# 연습문제

---

- 다음 작업을 수행하세요:
  1. x 축의 값으로 사용될 0에서 10까지의 숫자를 생성하세요.
  2. 첫 번째 함수로  $\sin(x)$ , 두 번째 함수로  $\cos(x)$ 를 사용합니다. 이 두 함수에 대해 plot을 생성하세요.
  3. plot에 "Sine and Cosine functions"라는 제목을 추가하세요.
  4. x축과 y축에 각각 "x values", "y values"라는 레이블을 추가하세요.
  5. 범례를 추가하여 첫 번째 그래프를 " $\sin(x)$ ", 두 번째 그래프를 " $\cos(x)$ "로 표시하세요.
  6. x축의 범위를 0에서 10으로, y축의 범위를 -1에서 1로 설정하세요.

# 연습문제 코드

```
import numpy as np
import matplotlib.pyplot as plt

# Data for plotting
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create a figure
plt.figure()

# Plot sin(x)
plt.plot(x, y1, label = 'sin(x)')

# Plot cos(x)
plt.plot(x, y2, label = 'cos(x)')

# Set labels
plt.xlabel('x values')
plt.ylabel('y values')

# Set title
plt.title('Sine and Cosine functions')

# Set legend
plt.legend()

# Set x and y limits
plt.xlim(0, 10)
plt.ylim(-1, 1)

# Show the plot
plt.show()
```



감사합니다  
[2차시]