```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

```
# 데이터.reindex([인덱스들]) : 인덱스를 바꾸는 함수
```

```
obj = Series([5.4,7.2,-5.3,3.6], index = ['d','b','a','c'])
obj
```

```
d    5.4
b    7.2
a   -5.3
c    3.6
dtype: float64
```

```
obj2 = obj.reindex(['a','b','c','d','e'])
obj2
```

```
a   -5.3
b    7.2
c    3.6
d    5.4
e    NaN
dtype: float64
```

```
obj.reindex(['a','b','c','d','e'],fill_value=0)  #Nan을 0으로 채워줌
```

```
a   -5.3
b    7.2
c    3.6
d    5.4
e    0.0
dtype: float64
```

```
## ffill 메소드 : 앞의 값으로 누락값 채워넣기

obj3 = Series(['blue','purple','yellow'],index = [0,2,4])
obj3.reindex(range(6),method='ffill')
```

```
0      blue
1      blue
2    purple
3    purple
4    yellow
5    yellow
dtype: object
```

```
## bfill 메소드 : 뒤의 값으로 누락값 채워넣기

obj4 = Series(['blue','purple','yellow'],index=[0,2,4])
obj4.reindex(range(6), method = 'bfill')
```

```
0      blue
1    purple
2    purple
3    yellow
4    yellow
5       NaN
dtype: object
```

```
frame = DataFrame(np.arange(9).reshape((3,3)), index = ['a','c','d'],
                  columns = ['Ohio','Texas','California'])
```

```
frame
```

	Ohio	Texas	California
a	0	1	2
c	3	4	5
d	6	7	8

```
frame2 = frame.reindex(['a','b','c','d'])
frame2
```

	Ohio	Texas	California
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	6.0	7.0	8.0

```
states = ['Texas','Utah','California']
frame.reindex(columns=states)
```

	Texas	Utah	California
a	1	NaN	2
c	4	NaN	5
d	7	NaN	8

```
frame.reindex(index=['a','b','c','d'],columns=states)
```

	Texas	Utah	California
a	1.0	NaN	2.0
b	NaN	NaN	NaN
c	4.0	NaN	5.0
d	7.0	NaN	8.0

loc, iloc

- loc : 문자로 행을 가져온다.
- iloc : 숫자로 행을 가져온다. (index loc)

```
frame.loc[['a','b','c','d'], states]
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:1494: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  return self._getitem_tuple(key)
```

	Texas	Utah	California
a	1.0	NaN	2.0
b	NaN	NaN	NaN
c	4.0	NaN	5.0
d	7.0	NaN	8.0

> 하나의 로우 또는 칼럼 삭제하기
>
> - 칼럼 삭제 : drop([칼럼명], axis=1)
> - 로우 삭제 : drop([로우명], axis=0) or drop([로우명])

```
obj = Series(np.arange(5.), index=['a','b','c','d','e'])
new_obj = obj.drop('c')
new_obj
```

```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

```
obj.drop(['d','c'])    # drop([없앨 대상])
```

```
a    0.0
b    1.0
e    4.0
dtype: float64
```

```
obj
```

```
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
dtype: float64
```

```
data = DataFrame(np.arange(16).reshape((4,4)),
                 index = ['Ohio','Colorado','Utah','New York'],
                 columns = ['one','two','three','four'])
data
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data.drop(['Colorado','Utah'])
```

	one	two	three	four
Ohio	0	1	2	3
New York	12	13	14	15

색인하기, 선택하기, 거르기

- Series의 색인(obj[...])은 NumPy 배열의 색인과 유사하게 동작하는데,
- Series의 색인은 정수가 아니어도 된다는 점이 다르다.
- 또한 라벨이름으로 슬라이싱하는 것은 시작점과 끝점을 포함한다.

```
obj = Series(np.arange(4.), index=['a','b','c','d'])
obj
```

```
a    0.0
b    1.0
c    2.0
d    3.0
dtype: float64
```

```
obj['b']
```

```
1.0
```

```
obj[1]
```

```
1.0
```

```
obj['b':'c']
```

```
b    1.0
c    2.0
dtype: float64
```

```
obj[2:4]   #슬라이싱 [ ]
```

```
c    2.0
d    3.0
dtype: float64
```

```
obj[['b','a','d']]
```

```
b    1.0
a    0.0
d    3.0
dtype: float64
```

```
obj[[1,3]]
```

```
b    1.0
d    3.0
dtype: float64
```

```
obj[obj<2]
```

```
a    0.0
b    1.0
dtype: float64
```

```
obj['b':'c'] = 5
obj
```

```
a    0.0
b    5.0
c    5.0
d    3.0
dtype: float64
```

```
## 시리즈가 두 개 이상이면 데이터프레임!
```

```
data = DataFrame(np.arange(16).reshape((4,4)),
                 index=['Ohio','Colorado','Utah','New York'],
                 columns =['one','two','three','four'])
data
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data['two']
```

```
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int64
```

```
data[['two','three']]
```

	two	three
Ohio	1	2
Colorado	5	6
Utah	9	10
New York	13	14

```
data[:2]
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

```
data[data['three'] >5]
```

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data <5
```

	one	two	three	four
Ohio	True	True	True	True
Colorado	True	False	False	False
Utah	False	False	False	False
New York	False	False	False	False

```
data[data<5] = 0
```

```
data
```

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data.loc['Colorado',['two','three']]
```

```
two      5
three    6
Name: Colorado, dtype: int64
```

```
data.ix[['Colorado','Utah'], [3,0,1]]    # integer position과 label 모두 사용할 수 있다.
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  """Entry point for launching an IPython kernel.
```

	four	one	two
Colorado	7	0	5
Utah	11	8	9

산술연산과 데이터 정렬

```
s1 = Series([7.3, -2.5, 3.4, 1.5], index = ['a','c','d','e'])
s2 = Series([-2.1, 3.3, -7.4, 3.1, 4], index = ['a','c','e','f','g'])
```

```
s1
```

```
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64
```

```
s2
```

```
a   -2.1
c    3.3
e   -7.4
f    3.1
g    4.0
dtype: float64
```

```
s1+s2
```

```
a    5.2
c    0.8
d    NaN
e   -5.9
f    NaN
g    NaN
dtype: float64
```

```
df1 = DataFrame(np.arange(9.).reshape((3,3)),columns=list('bcd'),
                index=['Ohio','Texas','Colorado'])
df2 = DataFrame(np.arange(12.).reshape(4,3),columns=list('dbe'),
                index = ['Utah','Ohio','Texas','Oregon'])
```

```
df1
```

	b	c	d
Ohio	0.0	1.0	2.0
Texas	3.0	4.0	5.0
Colorado	6.0	7.0	8.0

```
df2
```

	d	b	e
Utah	0.0	1.0	2.0
Ohio	3.0	4.0	5.0
Texas	6.0	7.0	8.0
Oregon	9.0	10.0	11.0

```
df1+df2     #겹치는 값만 연산
```

	b	c	d	e
Colorado	NaN	NaN	NaN	NaN
Ohio	4.0	NaN	5.0	NaN
Oregon	NaN	NaN	NaN	NaN
Texas	10.0	NaN	11.0	NaN
Utah	NaN	NaN	NaN	NaN

산술연산 메소드에 채워 넣을 값 지정

```
df1 = DataFrame(np.arange(12.).reshape((3,4)), columns=list('abcd'))
df2 = DataFrame(np.arange(20.).reshape((4,5)), columns=list('abcde'))
```

```
df1
```

	a	b	c	d
0	0.0	1.0	2.0	3.0
1	4.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0

```
df2
```

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	6.0	7.0	8.0	9.0
2	10.0	11.0	12.0	13.0	14.0
3	15.0	16.0	17.0	18.0	19.0

```
df1+df2
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	NaN
1	9.0	11.0	13.0	15.0	NaN
2	18.0	20.0	22.0	24.0	NaN
3	NaN	NaN	NaN	NaN	NaN

```
df1.add(df2, fill_value=0)
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	4.0
1	9.0	11.0	13.0	15.0	9.0
2	18.0	20.0	22.0	24.0	14.0
3	15.0	16.0	17.0	18.0	19.0

DataFrame과 Series 연산

```
arr = np.arange(12.).reshape((3,4))
arr
```

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

```
arr[0]
```

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

```
arr - arr[0]
```

```
array([[0., 0., 0., 0.],
       [4., 4., 4., 4.],
       [8., 8., 8., 8.]])
```

```
frame = DataFrame(np.arange(12.).reshape((4,3)), columns=list('bde'),
                  index=['Utah','Ohio','Texas','Oregon'])
```

```
series = frame.iloc[0]
series
```

```
b    0.0
d    1.0
e    2.0
Name: Utah, dtype: float64
```

```
frame
```

	b	d	e
Utah	0.0	1.0	2.0
Ohio	3.0	4.0	5.0
Texas	6.0	7.0	8.0
Oregon	9.0	10.0	11.0

```
frame - series
```

	b	d	e
Utah	0.0	0.0	0.0
Ohio	3.0	3.0	3.0
Texas	6.0	6.0	6.0
Oregon	9.0	9.0	9.0

```
series2 = Series(range(3), index=['b','e','f'])
series2
```

```
b    0
e    1
f    2
dtype: int64
```

```
frame+series2
```

	b	d	e	f
Utah	0.0	NaN	3.0	NaN
Ohio	3.0	NaN	6.0	NaN
Texas	6.0	NaN	9.0	NaN
Oregon	9.0	NaN	12.0	NaN

```
series3 = frame['d']
series3
```

```
Utah      1.0
Ohio      4.0
Texas     7.0
Oregon    10.0
Name: d, dtype: float64
```

```
frame.sub(series3, axis=0)
```

	b	d	e
Utah	-1.0	0.0	1.0
Ohio	-1.0	0.0	1.0
Texas	-1.0	0.0	1.0
Oregon	-1.0	0.0	1.0

함수 적용과 매핑

- Pandas 객체에도 Numpy의 유니버설 함수를 적용할 수 있다.
- 유니버설 함수 : 배열의 각 원소에 적용되는 메소드

```
frame = DataFrame(np.random.randn(4,3),columns=list('bde'),
                  index=['Utah','Ohio','Texas','Oregon'])
```

```
frame
```

	b	d	e
Utah	0.335188	0.658969	-1.935243
Ohio	-0.338729	0.854647	-1.044901
Texas	0.140435	0.724125	-0.349138
Oregon	-1.262255	1.122823	-0.160988

```
np.abs(frame)  # abs : 절댓값
```

	b	d	e
Utah	0.335188	0.658969	1.935243
Ohio	0.338729	0.854647	1.044901
Texas	0.140435	0.724125	0.349138
Oregon	1.262255	1.122823	0.160988

```
f = lambda x : x.max() - x.min()    # lambda 함수
```

```
frame.apply(f)
```

```
b    1.597443
d    0.463854
e    1.774255
dtype: float64
```

```
def f(x) :
    return Series([x.min(), x.max()], index=['min','max'])

frame.apply(f)
```

	b	d	e
min	-1.262255	0.658969	-1.935243
max	0.335188	1.122823	-0.160988

```
#실수값을 문자열 포맷으로 변환
#반올림

format = lambda x : '%.2f' % x
frame.applymap(format)
```

	b	d	e
Utah	0.34	0.66	-1.94
Ohio	-0.34	0.85	-1.04
Texas	0.14	0.72	-0.35
Oregon	-1.26	1.12	-0.16

```
frame['e'].map(format)
```

정렬과 순위

```
#ascending=False 내림차순
#ascending=True  오름차순
```

```
obj = Series(range(4),index=list('dabc'))
obj.sort_index()
```

```
a    1
b    2
c    3
d    0
dtype: int64
```

```
frame = DataFrame(np.arange(8).reshape((2,4)),index=['three','one'],
                  columns=list('dabc'))
frame
```

	d	a	b	c
three	0	1	2	3
one	4	5	6	7

```
frame.sort_index()
```

	d	a	b	c
one	4	5	6	7
three	0	1	2	3

```
frame.sort_index(axis=1)
```

	a	b	c	d
three	1	2	3	0
one	5	6	7	4

```
frame.sort_index(axis=1, ascending=False)
```

	d	c	b	a
three	0	3	2	1
one	4	7	6	5

```
obj = Series([4,7,-3,2])
obj.sort_values()
```

```
2   -3
3    2
0    4
1    7
dtype: int64
```

```
obj = Series([4, np.nan, 7, np.nan, -3, 2])
obj.sort_values()
```

```
4   -3.0
5    2.0
0    4.0
2    7.0
1    NaN
3    NaN
dtype: float64
```

```
frame = DataFrame({'b' : [4,7,-3,2], 'a' : [0,1,0,1]})
frame
```

	b	a
0	4	0
1	7	1
2	-3	0
3	2	1

```
frame.sort_values(by='b')   #b 기준
```

	b	a
2	-3	0
3	2	1
0	4	0
1	7	1

```
frame.sort_values(by=['a','b'])
```

	b	a
2	-3	0
0	4	0
3	2	1
1	7	1

```
obj = Series([7,-5,7,4,2,0,4])
```

rank() : 순위

rank method

- method='average' : 동점 관측치 간의 그룹 내에서 평균 순위 부여 (default)
- method='min' : 동점 관측치 그룹 내에서 최소 순위 부여
- method='max' : 동점 관측치 그룹 내에서 최대 순위 부여
- method='first' : 동점 관측치 중에서 먼저 나타나는 관측치부터 순위 부여
- method='dense' : 최소값('min')과 같은 방법으로 순위부여를 하지만, 'min'과는 다르게 그룹 간 순위가 '1'씩 증가한다.

```
obj.rank()   # 랭킹
```

```
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
dtype: float64
```

```
obj.rank(method='first')
```

```
0    6.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

```
obj.rank(ascending=False, method='max')
```

```
0    2.0
1    7.0
2    2.0
3    4.0
4    5.0
5    6.0
6    4.0
dtype: float64
```

```
obj.rank(ascending=False,method='dense')
```

```
0    1.0
1    5.0
2    1.0
3    2.0
4    3.0
5    4.0
6    2.0
dtype: float64
```

```
frame = DataFrame({'b' : [4.3,7,-3,2],
                   'a' : [0,1,0,1],
                   'c' : [-2,5,8,-2.5]})
frame
```

	b	a	c
0	4.3	0	-2.0
1	7.0	1	5.0
2	-3.0	0	8.0
3	2.0	1	-2.5

```
frame.rank(axis=1)
```

	b	a
0	2.0	1.0
1	2.0	1.0
2	1.0	2.0
3	2.0	1.0

중복 색인

```
obj = Series(range(5), index=list('aabbc'))
```

```
obj
```

```
a    0
a    1
b    2
b    3
c    4
dtype: int64
```

```
obj.index.is_unique
```

```
False
```

```
obj['c']
```

```
4
```

```
obj[['a','c']]
```

```
a    0
a    1
c    4
dtype: int64
```

```
df =DataFrame(np.random.randn(4,3), index=list('aabb'))
df
```

	0	1	2
a	0.277569	0.293562	0.775577
a	1.642211	0.140085	0.699753
b	0.204539	-0.850062	1.356970
b	-0.352696	-1.084041	-0.953166

```
df.loc['b']
```

	0	1	2
b	0.204539	-0.850062	1.356970
b	-0.352696	-1.084041	-0.953166