

PYTHON
PANDAS
기초
VERSION 2.X

1 차원 데이터 관리 (SERIES)



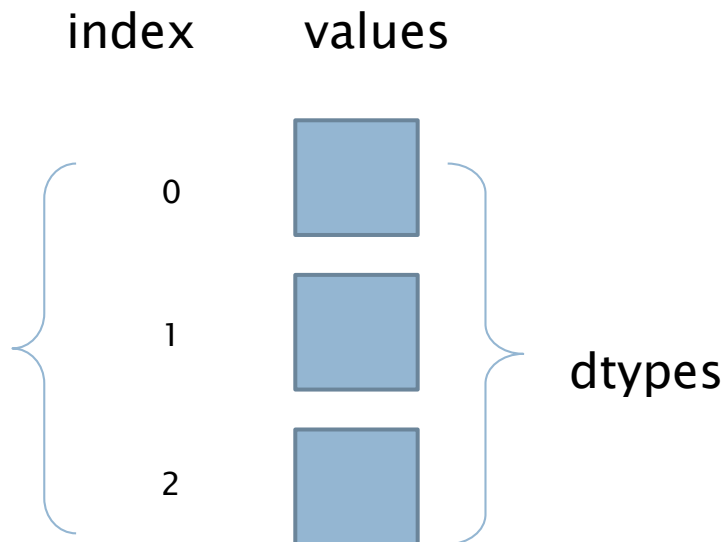


Series 구조

Series 구조

1 차원의 데이터를 관리하는 컨테이너

```
pandas.Series(data,index,dtypes,name,copy)
```



data: 실제 데이터 값
index : 데이터를 접근할 정보
dtypes : 데이터들의 타입
name : Series 인스턴스의 이름

Series 구조 속성

변수	설명
name	Series 인스턴스에 대한 이름
shape	DataFrame의 행렬 형태를 표시
dtypes	행과 열에 대한 데이터 타입을 표시
ndim	차원에 대한 정보 표시
size	원소들의 갯수
ftypes	Return the ftypes (indication of sparse/dense and dtype) in this object.
axes	행과 열에 대한 축을 접근 표시
empty	DataFrame 내부가 없으면 True 원소가 있으면 False
strides	데이터를 구성하는 총 갯수
index	생성된 행에 대한 index 표시
columns	칼럼에 대한 접근 표시
values	실제 data를 Numpy 로 변환

Series 생성 : name 추가

dict를 받아서 Series 인스턴스를 생성

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

```
d = {'a':1,'b':2,'c':3}
s3 = pd.Series(d,name='something')
print(s3, s3.name)
```

```
(a    1
 b    2
 c    3
 Name: something, dtype: int64, 'something')
```

Series 구조 속성: index, values, dtype

Series의 주요 속성인 values는 데이터를 나타내고 index는 series 인스턴스에 대한 index 정보를 가짐

```
import pandas as pd

obj = pd.Series([4,-7,5,3])

print(obj.index)
print(obj.values)

obj.index = ['a','b','c','d']
print(obj.index)
print(obj.values)

obj['a'] = 100
print(type(obj.values),obj.values)
#obj.values = pd.Series([99,99,99,99]).values
#AttributeError: can't set attribute
print(obj.dtype, obj.ftype)
```

```
RangeIndex(start=0, stop=4, step=1)
[ 4 -7  5  3]
Index([u'a', u'b', u'c', u'd'], dtype='object')
[ 4 -7  5  3]
(<type 'numpy.ndarray'>, array([100, -7,  5,  3],
dtype=int64))
(dtype('int64'), 'int64:dense')
```

Series 구조 속성 : shape, ndim

주요 Series 모형과 차원에 대해 출력

```
import pandas as pd

d = {'a':1,'b':2,'c':3}
s3 = pd.Series(d,name='something')
print(s3, s3.name)

print(s3.shape)
print(s3.ndim)
print(s3.size)
```

```
(a  1
 b  2
 c  3
Name: something, dtype: int64, 'something')
(3,)
1
3
```


Series 구조 속성 : strides, base

주요 Series 속성에 대해 출력

```
import pandas as pd

d = {'a':1,'b':2,'c':3}
s3 = pd.Series(d,name='something')

print(s3.strides)
print(s3.base)
```

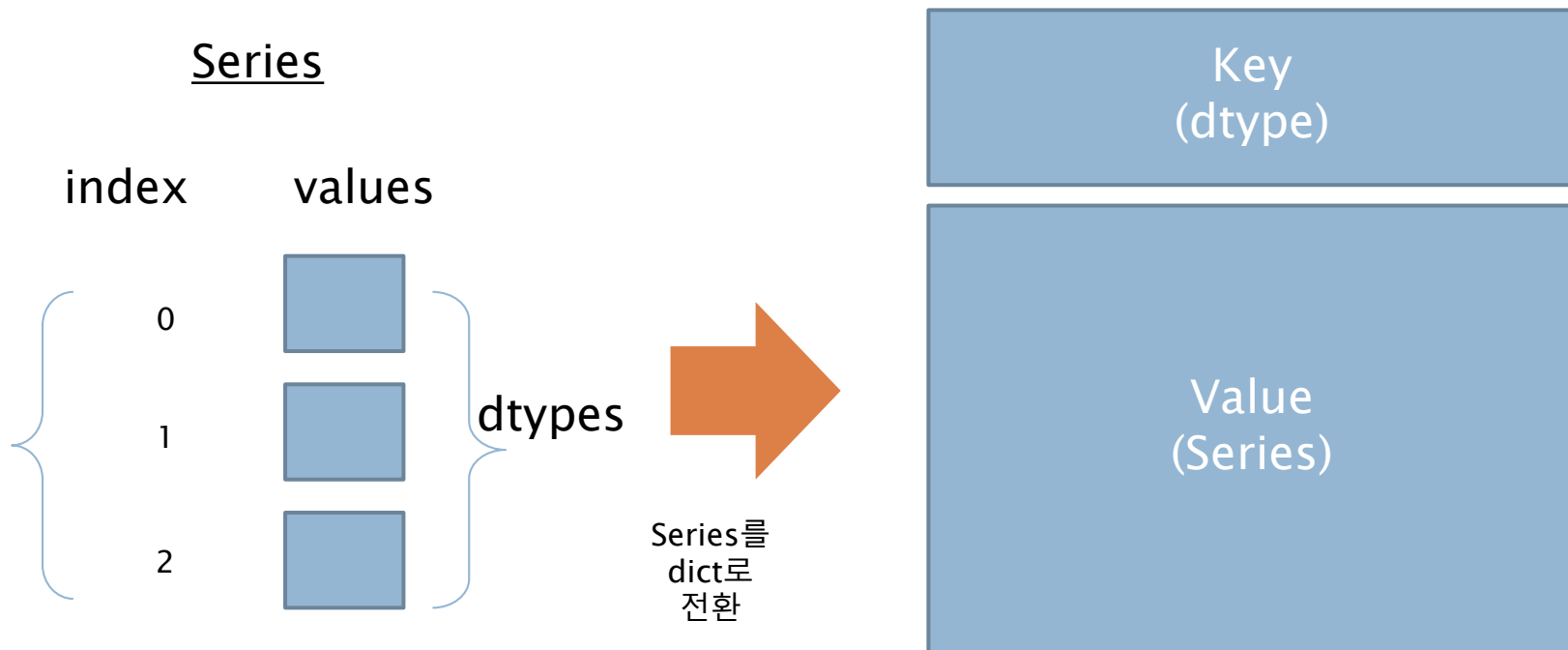
```
a    1
b    2
c    3
Name: something, dtype: int64
(8,)
[1 2 3]
```

Series 변환 속성

변수	설명
blocks	Internal property, property synonym for as_blocks()
T	행과 열을 변환

Series 구조 변환: blocks

Series 인스턴스를 dict 타입으로 변환처리



Series 구조 변환: blocks 예시

Series이 인스턴스에 대해 dict 타입으로 데이터를 표시

```
import pandas as pd
```

```
obj1 = pd.Series([1,2,3,4],name="test")
```

```
print(obj1.blocks)
```

```
print(obj1.blocks['int64'].index)
```

```
print(obj1.blocks['int64'].values)
```

```
print(obj1.blocks['int64'].name)
```

```
print(obj1.blocks['int64'].dtype)
```

```
{'int64': 0 1
```

```
1 2
```

```
2 3
```

```
3 4
```

```
Name: test, dtype: int64}
```

```
Int64Index([0, 1, 2, 3], dtype='int64')
```

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

```
test
```

```
int64
```

Series 구조 변환: T

주요 Series 속성에 index, values를 교체해서 보여줌

```
import pandas as pd

d = {'a':1,'b':2,'c':3}
s3 = pd.Series(d,name='something')

print(s3.T)
```

```
a    1
b    2
c    3
Name: something, dtype: int64
```

Series 접근 속성

변수	설명
at	Fast label-based scalar accessor
iat	Fast integer location scalar accessor.
ix	A primarily label-location based indexer, with integer position fallback.
loc	Purely label-location based indexer for selection by label.
iloc	Purely integer-location based indexing for selection by position.

Series 접근 속성: at, iat, loc, iloc, ix

주요 Series 인스턴스의 값을 접근하기 위해 at은 원소의 값, loc는 슬라이싱 처리를 포함해서 검색, ix는 값을 검색

```
import pandas as pd

obj1 = pd.Series([1,2,3,4],name="test")

obj1.index = ['a','b','c','d']
print(obj1.at['a'])
print(obj1.iat[0])

print(obj1.iloc[0:3],obj1.loc['a':'c'])

print(type(obj1.ix['a']),obj1.ix['a'] == 1)
```

```
1
1
(a  100
b   -7
c    5
dtype: int64, a   100
b   -7
c    5
dtype: int64)
(<type 'numpy.int64'>, True)
```



Series 생성

Series 생성 : list-like

List를 받아서 Series 인스턴스를 생성

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

```
print(np.random.randn(5))
s = pd.Series(np.random.randn(5))
```

```
s1 = pd.Series(np.random.randn(5), index=['a',
'b', 'c', 'd', 'e'])
```

```
print(s, type(s))
print(s1, type(s1))
```

```
[-0.15090796  0.16353861 -0.28383656
 1.48342456  1.26292765]
```

```
(0  -0.708895
 1  -0.267631
 2   0.587089
 3   1.051497
 4  -1.143584
dtype: float64, <class 'pandas.core.series.Series'>)
```

```
(a   1.125429
 b   1.063341
 c   0.637474
 d   0.966092
 e   0.796640
dtype: float64, <class 'pandas.core.series.Series'>)
```

Series 생성 : dict-like

dict를 받아서 Series 인스턴스를 생성

```
import numpy as np
import pandas as pd

d = {'a':1,'b':2,'c':3}
s2 = pd.Series(d)
print(s2)
```

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

Series 생성 : MultiIndex

MultiIndex를 받아 Series 인스턴스를 생성

```
import pandas as pd
import numpy as np

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'])

s = pd.Series(np.random.randn(8), index=index)
print(s)
```

```
first second
bar one   -0.483801
    two    0.727163
baz one    0.363992
    two    1.078241
foo one    0.314946
    two   -1.205744
qux one   -0.387636
    two   -1.015069
dtype: float64
```



Series 접근

Series 조회 : index

List처럼 인덱스를 조회해서 원소를 검색

```
import numpy as np
import pandas as pd

s = pd.Series([0,1,2,3,4])

s1 = pd.Series([0,1,2,3,4], index=['a', 'b', 'c', 'd',
'e'])

print(s.index, s[0])
print(s1.index,s1['a'])
```

```
(Int64Index([0, 1, 2, 3, 4], dtype='int64'), 0)
(Index([u'a', u'b', u'c', u'd', u'e'], dtype='object'), 0)
```

Series 조회 : slice

Series의 slice는 순서를 표시하므로 index가 문자여도 가능

```
import numpy as np
import pandas as pd
s = pd.Series([0,1,2,3,4])

s1 = pd.Series([0,1,2,3,4], index=['a', 'b', 'c', 'd', 'e'])

print(s[0:3])
print(s1['a':'c'])
```

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

Series 조회 : index 직접 대응

Series의 ndarray 처럼 조회조건에 index를 리스트 넣어 검색 가능

```
import numpy as np
import pandas as pd
s = pd.Series([0,1,2,3,4])

s1 = pd.Series([0,1,2,3,4], index=['a', 'b', 'c', 'd', 'e'])

print(s[[4,3,1]])
print(s1[['d','a','c']])
```

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

Series 조회 : 논리식

Series의 ndarray 처럼 조회조건도 논리식으로 처리가 가능

```
import numpy as np
import pandas as pd
s = pd.Series([0,1,2,3,4])

s1 = pd.Series([0,1,2,3,4], index=['a', 'b', 'c', 'd', 'e'])

print(s[s<3])
print(s1[s1>3])
```

```
0  0
1  1
2  2
dtype: int64
e  4
dtype: int64
```


Series 조회 : No Index

Series이 index 범위가 벗어나면 KeyError 발생

```
import pandas as pd  
  
d = {'a':1,'b':2,'c':3}  
s2 = pd.Series(d)  
print(s2)  
  
print(s2['d'])
```

KeyError: 'd'

Series 조회 : get() 메소드

Series의 index 범위가 벗어나도 KeyError 발생하지 않으려면 get() 메소드를 사용

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

```
d = {'a':1,'b':2,'c':3}
s2 = pd.Series(d)
```

```
print(s2.get('d'))
print(s2.get('d',np.nan))
```

None
nan



Series 변경

Series 동일 값 전체변경 : replace()

인스턴스 원소의 값을 변경

```
import pandas as pd
import numpy as np

obj = pd.Series([1,2,3,4])

print(obj.replace(4,99))
print(obj)
print(obj.replace(4,99, inplace=True))
print(obj)
```

dtype: int64

```
0    1
1    2
2    3
3   99
```

dtype: int64

```
0    1
1    2
2    3
3    4
```

dtype: int64

```
None
0    1
1    2
2    3
3   99
```

dtype: int64

Series 특정 원소 변경 : replace()

Replace 메소드는 값 전체를 바꾸므로 특정부분을 추출하여 적용할 경우에만 특정 값이 변경

```
import pandas as pd
import numpy as np

obj6 = pd.Series([4,4,4,4])

print(obj6.replace(4,99, inplace=True))
print(obj6[:1].replace(99,4,inplace=True))
print(obj6)
```

```
None
0    4
1   99
2   99
3   99
dtype: int64
```

Series sort : sort_values()

set_value메소드도 inplace=True로 객체를 변경함
Index가 변경되지 않으므로 변경이 필요
Sort메소드 대신 sort_values메소드를 사용

```
import pandas as pd
import numpy as np

print(obj7)
print(obj7.sort_values(inplace=True))
print(obj7)

print(obj7.reindex([0,2,3,1]))
obj7.index = [0,1,2,3]
print(obj7)
```

```
0    4
1   999
2    4
3    4
dtype: int64
None
0    4
2    4
3    4
1   999
dtype: int64
0    4
2    4
3    4
1   999
dtype: int64
0    4
1    4
2    4
3   999
dtype: int64
```

Series multi index 접근

Series 조회 : Multi index

첫번째 인덱스만 넣을 경우는 해당 하위 index와 값이 출력되고, 인덱스를 모두 넣을 경우는 값만 출력

```
import pandas as pd
import numpy as np

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'])

s = pd.Series(np.random.randn(8), index=index)
print(s)

print(s['bar'])
print(s['bar','one']) # 첫번째와 두번째 인덱스
```

```
first second
bar one    0.447746
    two   -0.564082
baz one   -0.951146
    two    0.966715
foo one    0.994714
    two   -0.501969
qux one    0.208027
    two    0.496674
dtype: float64
second
one    0.447746
two   -0.564082
dtype: float64
0.447745697452
```




Series 산술연산

Series 연산: scala

Series와 scala 값과 계산시 전체를 vector 값으로
전환해서 계산

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

```
d = {'a':1,'b':2,'c':3}
s2 = pd.Series(d)
print(s2 + 1)
print(s2 - 1)
print(s2 * 3)
print(s2 / 2)
print(s2 // 2)
print(s2 % 2))
```

```
a    2
b    3
c    4
dtype: int64
a    0
b    1
c    2
dtype: int64
a    3
b    6
c    9
dtype: int64
a    0.5
b    1.0
c    1.5
dtype: float64
a    0
b    1
c    1
dtype: int64
a    1
b    0
c    1
dtype: int64
```

Series 연산: vector

Series는 크기에 맞춰 계산 하므로 index가 매칭되지 않을 경우는 NaN 처리

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

```
d = {'a':1,'b':2,'c':3}
s2 = pd.Series(d)
```

```
print(s2 + s2)
print(s2 + s2[0:2])
```

```
a    2
b    4
c    6
dtype: int64
a    2
b    4
c   NaN
dtype: float64
```

Series 간 산술연산

Series 인스턴스에 대한 산술연산(+,-,*,/,//,%)

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

```
obj2 = pd.Series([1,2,3,4])
obj3 = pd.Series([5,6,7,8])
print(obj2+obj3)
print(obj2-obj3)
print(obj2*obj3)
print(obj2/obj3)
print(obj2//obj3)
print(obj2%obj3)
```

```
0    6
1    8
2   10
3   12
dtype: int64
0   -4
1   -4
2   -4
3   -4
dtype: int64
0    5
1   12
2   21
3   32
dtype: int64
0    0.200000
1    0.333333
2    0.428571
3    0.500000
dtype: float64
0    0
1    0
2    0
3    0
dtype: int64
0    1
1    2
2    3
3    4
dtype: int64
```

Series 절대값 처리

Series 인스턴스의 값들이 음수일 경우 절대값(abs) 처리

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

```
obj = pd.Series([1,2,3,4])
obj5 = obj * -1
print(obj5)
print(obj5.abs())
```

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



Series 산술 메소드

Series 연산: add

add/radd 메소드 사용

```
import pandas as pd
import numpy as np

obj = pd.Series([1,2,3,4])
obj1 = pd.Series([5,6,7,8])
obj2 = pd.Series([5,6,7,8,9])

obj3 = obj.add(obj1)
print("add",obj3)
print("radd",obj.radd(obj1))
print(obj.add(obj2))
```

```
('add', 0    6
      1    8
      2   10
      3   12
      dtype: int64)
('radd', 0    6
        1    8
        2   10
        3   12
        dtype: int64)
0    6.0
1    8.0
2   10.0
3   12.0
4    NaN
dtype: float64
```

Series 연산: sub

sub/rsub 메소드 사용

```
import pandas as pd
import numpy as np

obj = pd.Series([1,2,3,4])
obj1 = pd.Series([5,6,7,8])
obj2 = pd.Series([5,6,7,8,9])

print("sub",obj.sub(obj1))
print("rsub",obj.rsub(obj1))
```

```
('sub', 0  -4
1  -4
2  -4
3  -4
dtype: int64)
('rsub', 0   4
1   4
2   4
3   4
dtype: int64)
```


Series 연산: mul

mul/rmul 메소드 사용

```
import pandas as pd
import numpy as np

obj = pd.Series([1,2,3,4])
obj1 = pd.Series([5,6,7,8])
obj2 = pd.Series([5,6,7,8,9])

print("mul",obj.mul(obj1))
print("rmul",obj.rmul(obj1))
```

```
('mul', 0    5
      1   12
      2   21
      3   32
      dtype: int64)
('rmul', 0    5
         1   12
         2   21
         3   32
         dtype: int64)
```

Series 연산: div

div/rdiv/floordiv/rfloordiv/truediv/rtruediv/divide 메소드 사용

```
import pandas as pd
import numpy as np

obj = pd.Series([1,2,3,4])
obj1 = pd.Series([5,6,7,8])
obj2 = pd.Series([5,6,7,8,9])

print("div",obj.div(obj1))
print("rdiv",obj.rdiv(obj1))
print("floordiv", obj.floordiv(obj1))
print("rfloordiv", obj.rfloordiv(obj1))
print("truediv",obj.truediv(obj1))
print("rtruediv",obj.rtruediv(obj1))
print("divide",obj.divide(obj2, fill_value=0.0))
```

```
('div', 0    0.200000
1    0.333333
2    0.428571
3    0.500000
dtype: float64)
```

```
('rdiv', 0    5.000000
1    3.000000
2    2.333333
3    2.000000
dtype: float64)
```

```
('floordiv', 0    0
1    0
2    0
3    0
dtype: int64)
```

```
('rfloordiv', 0    5
1    3
2    2
3    2)
```

```
('truediv', 0    0.200000
1    0.333333
2    0.428571
3    0.500000
dtype: float64)
```

```
('rtruediv', 0    5.000000
1    3.000000
2    2.333333
3    2.000000
dtype: float64)
```

```
('divide', 0    0.200000
1    0.333333
2    0.428571
3    0.500000
4    0.000000
dtype: float64)
```

Series 연산: mod

mod/rmod 메소드 사용

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

```
obj = pd.Series([1,2,3,4])
obj1 = pd.Series([5,6,7,8])
obj2 = pd.Series([5,6,7,8,9])
```

```
print("mod",obj.mod(obj1, fill_value=0.0))
print("rmod",obj.rmod(obj1, fill_value=0.0))
```

```
('mod', 0 1
1 2
2 3
3 4
dtype: int64)
('rmod', 0 0
1 0
2 1
3 0
dtype: int64)
```



Series NaN 연산

Series 생성시 값이 없을 경우

Series 인스턴스의 원소가 NaN 타입일 경우 dtype이 float64로 세팅되고 add 처리시 결과가 NaN 으로 처리됨
add메소드에서 NaN 값이 있을 경우 fill_value 인자에 초기 값을 부여해야 함

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

```
obj2 = pd.Series([1,2,3,4])
obj4 = pd.Series(index=[0,1,2,3])
print(obj4)
print(obj2.add(obj4,fill_value=0))
obj2.astype(dtype=np.int64)
print(obj2)
print(obj2.add(obj4))
```

```
0 NaN
1 NaN
2 NaN
3 NaN
dtype: float64
0 1.0
1 2.0
2 3.0
3 4.0
dtype: float64
0 1
1 2
2 3
3 4
dtype: int64
0 NaN
1 NaN
2 NaN
3 NaN
dtype: float64
```

Series들을 연결:append

Series 들을 연결하기 위해 append 메소드를 사용함.
Verify_integrity=True 줄 경우 index 중복 시 오류
(ValueError: Indexes have overlapping values:)

```
import pandas as pd
import numpy as np

obj4 = pd.Series(index=[0,1,2,3])
data = pd.Series([0.25, 0.5, 0.75, 1.0])
print(data.append(obj4))
obj5 = pd.Series([5,6,7,8],index=[5,6,7,8])
print(data.append(obj5,verify_integrity=True))
```

```
0    0.25
1    0.5
2    0.75
3    1.0
0     na
1    NaN
2    NaN
3    NaN
dtype: object
0    0.25
1    0.50
2    0.75
3    1.00
5    5.00
6    6.00
7    7.00
8    8.00
dtype: float64
```



Count

Series 원소의 갯수:count

Series count 메소드를 이용해서 null이 아닌 갯수를 처리

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

```
obj = pd.Series([1,2,3,4])  
print(obj.count())
```

```
obj4 = pd.Series(index=[0,1,2,3])  
print(obj4.count())
```

4
0

Series 원소의 갯수: value count

Series `value_counts` 메소드를 사용해서 원소들이 구성을 확인

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

```
data = np.random.randint(0, 7, size=50)
sv = pd.Series(data)
```

```
print(sv.value_counts())
```

```
0    11
6    10
1     9
2     7
5     6
4     5
3     2
dtype: int64
```

Series : mode

Series 내의 최고 발생한 것을 확인하는 메소드

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

```
data = np.random.randint(0, 7, size=50)
sv = pd.Series(data)
```

```
print(sv.value_counts())
print(sv.mode())
```

```
3    14
0    11
2     7
1     6
6     5
4     4
5     3
dtype: int64
0     3
dtype: int32
```



Series key/value 확인

Series 내의 key/value 확인

Key/iteritems 메소드를 가지고 key/value를 확인
Values 속성으로 원소들의 값을 확인

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

```
data = pd.Series([0.25, 0.5, 0.75, 1.0])  
print(data.keys())  
print(data.values)  
for k,v in data.iteritems() :  
    print (k,v),
```

```
RangeIndex(start=0, stop=4, step=1)  
[ 0.25  0.5   0.75  1. ]  
(0, 0.25) (1, 0.5) (2, 0.75) (3, 1.0)
```

Series 평균 / 표준편차 / 분산

Series 합, 평균, 표준편차, 분산

평균(mean), 표준편차(std), 분산(var)에 대해 구하기

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

```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
print(data.sum(), data.sum()/len(data))
print(" average ",data.mean())
print(" median ",data.median())
print(" var ",data.var())
```

```
print(" standard deviation ",data.std())
print("      ",math.sqrt(data.var()))
```

```
(2.5, 0.625)
(' average ', 0.625)
(' median ', 0.625)
(' var ', 0.10416666666666667)
(' standard deviation ', 0.3227486121839514)
('      ', 0.3227486121839514)
```



Series describe

Series 숫자 데이터 통합 조회

평균(mean), 표준편차(std), 분산(var) 등을 한번에
구하기(describe)

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

series = pd.Series(np.random.randn(500))

series[20:500] = np.nan
series[10:20] = 5

print(series.describe())

print(series.describe(percentiles=[.05, .25, .75, .95]))
```

```
count    20.000000
mean      2.497204
std       2.686809
min      -1.437236
25%      -0.091405
50%       3.664195
75%       5.000000
max       5.000000
dtype: float64
count    20.000000
mean      2.748251
std       2.410239
min      -0.626104
5%       -0.499558
25%       0.159144
50%       3.585367
75%       5.000000
95%       5.000000
max       5.000000
dtype: float64
```


Series 문자 데이터 통합 조회

문자들에 대한 구성에 대해 확인

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

```
s = pd.Series(['a', 'a', 'b', 'b', 'a', 'a', np.nan, 'c', 'd', 'a'])
```

```
print(s.describe())
print(s.unique())
```

```
count    9
unique    4
top      a
freq     5
dtype: object
['a' 'b' nan 'c' 'd']
```



unique

Series 동일한 숫자 원소 확인

Series 인스턴스내의 동일한 숫자 원소가 몇개인지
를 확인 (nunique)

```
import pandas as pd
import numpy as np

series = pd.Series(np.random.randn(500))

series[20:500] = np.nan

series[10:20] = 5
print(series)

print(series.nunique())
```

```
0    -0.378757
1    -0.295574
...
10     5.000000
11     5.000000
12     5.000000
13     5.000000
14     5.000000
15     5.000000
16     5.000000
17     5.000000
18     5.000000
19     5.000000
20         NaN
21         NaN
22         NaN
23         NaN
...
497        NaN
498        NaN
499        NaN
dtype: float64
11
```

Series 동일한 문자 원소 확인

문자 원소에 대해 대표적인 원소들을 확인(unique)

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

s = pd.Series(['a', 'a', 'b', 'b', 'a', 'a', np.nan, 'c', 'd', 'a'])

print(s.describe())
print(s.unique())
```

```
count    9
unique    4
top      a
freq     5
dtype: object
['a' 'b' nan 'c' 'd']
```



min/max

Series min/max

Series 인스턴스 내의 원소들에 대한 min/max 구하기

```
import pandas as pd  
  
obj = pd.Series([1,2,3,4])  
print(obj.min())  
print(obj.max())
```

1
4

Series idxmin/idxmax

Series 인스턴스 내의 원소들에 대한 min/max 에 대한 index 구하기

```
import pandas as pd

obj = pd.Series([1,2,3,4])
print(obj.min())
print(obj.max())
print(obj.idxmin())
print(obj.idxmax())
```

1
4
0
3

Series cummin/cummax

Series 인스턴스 내의 원소들에 대한
cummin/cummax 구하기

```
import pandas as pd
```

```
obj = pd.Series([1,2,3,4,1,1,1])  
print(obj.cummax(axis=0))  
print(obj.cummin(axis=0))
```

```
0  1  
1  2  
2  2  
3  3  
4  3  
5  4  
6  4  
dtype: int64  
0  1  
1  1  
2  1  
3  1  
4  1  
5  1  
6  1  
dtype: int64
```




Series head/tail 조회

Series head/tail 조회

Head/tail 조회 default가 5건이며, n= 숫자를 인자로 전달해서 더 많은 건을 조회할 수 있음

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

long_series = pd.Series(np.random.randn(15))
print(long_series.head())
print(long_series.tail())
print(long_series.head(n=7))
```

```
0    0.932664
1   -0.025537
2   -0.157819
3   -0.814285
4   -0.600578
dtype: float64
10   -0.053254
11   -0.383670
12   -0.371867
13   -0.856907
14    0.763877
dtype: float64
0    0.932664
1   -0.025537
2   -0.157819
3   -0.814285
4   -0.600578
5   -0.680245
6    1.303397
dtype: float64
```



Boolean Reductions

Boolean Reductions

비교나 논리 연산을 사용할 경우에도 Series 인스턴스 전체가 처리가 되므로 이를 축소해서 boolean 처리하기 위한 메소드

```
import pandas as pd  
  
data = pd.Series([0.25, 0.5, 0.75, 1.0])  
  
print((data+data) == (data*2))
```

```
0    True  
1    True  
2    True  
3    True  
dtype: bool
```

empty

원소가 존재하지 않은 Series 인스턴스를 평가할 때
사용

```
import pandas as pd  
  
obj_emp = pd.Series()  
print(obj_emp.empty)
```

True

any(), all() : 비교

원소의 값이 논리식에 위한 전부 True 경우만 all()에서 True, any() 메소드는 하나의 True만 존재해도 True로 처리

```
import pandas as pd
```

```
data = pd.Series([0.25, 0.5, 0.75, 1.0])  
print((data>0.5).all())  
print((data>0.5).any())
```

```
False  
True
```

any(), all() : 사칙 / 비교 연산

원소의 값이 논리식에 위한 전부 True 경우만 all()에서 True, any() 메소드는 하나의 True만 존재해도 True로 처리

```
import pandas as pd
```

```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
```

```
print((data == data).all())
```

```
print((data+data != data).all())
```

```
True  
True
```

bool()

Bool메소드는 하나의 원소의 값이 True/False 여부 체크

```
import pandas as pd

print(pd.Series([True]).bool())
print(pd.Series([False]).bool())
```

True
False

equals()

계산된 결과가 동등한지 처리하는 메소드

```
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1.0])

print((data+data).equals(data*2))
print((data+data) == (data*2))
print(((data+data) == (data*2)).all())
```

```
True
0    True
1    True
2    True
3    True
dtype: bool
True
```

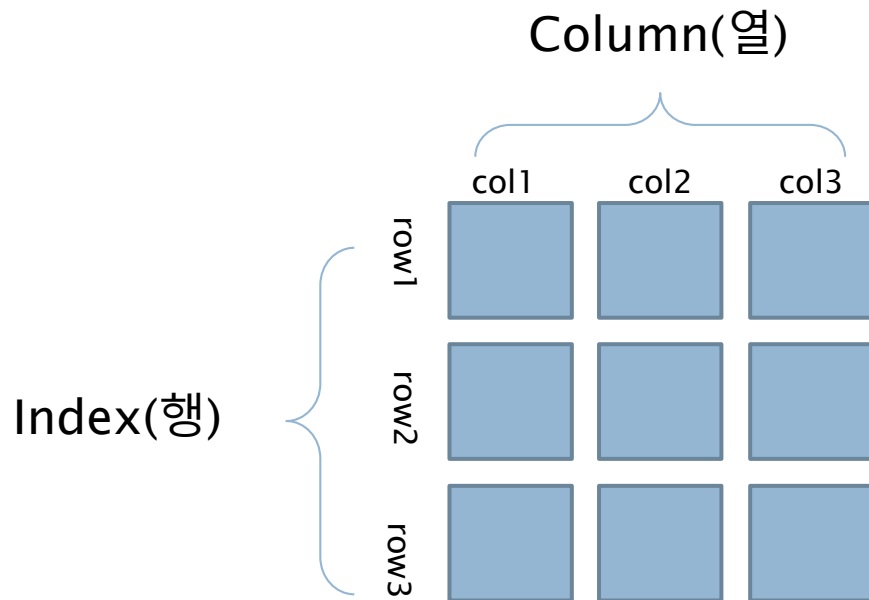
2차원 데이터 관리 (DATAFRAME)



DataFrame class 구조

DataFrame 구조

$n*m$ 행열구조를 가지는 데이터 구조이고 index와 column이 별도의 명을 가지고, column별로 다른 데이터 타입을 가질 수 있음



DataFrame 생성

$n*m$ 행열구조를 가지는 데이터 구조 생성

```
class DataFrame(pandas.core.generic.NDFrame)
| 2차원 행렬
| Parameters
| -----
| data : numpy ndarray ,dict, or DataFrame
|       dict can contain Series, arrays, constants, or list-like objects
| index : Index or array-like
|       행에 대한 정보 기본은 np.arange(n), 명칭도 부여 가능
| columns : Index or array-like
|          행에 대한 정보 기본은 np.arange(n), 명칭도 부여 가능
| dtype : dtype, default None
|       Data type to force, otherwise infer
| copy : boolean, default False
|       Copy data from inputs. Only affects DataFrame / 2d ndarray input
```



DataFrame 이해하기

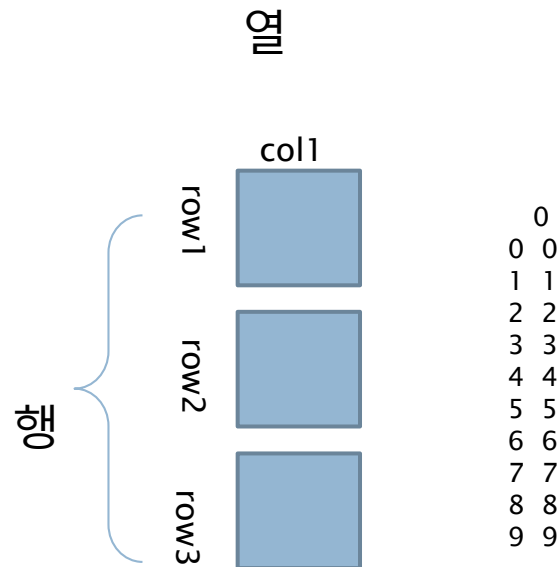
DataFrame 생성

DataFrame은 기본적으로 column 단위로 데이터를 관리함

```
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)
print(df)
```



DataFrame 컬럼명 추가

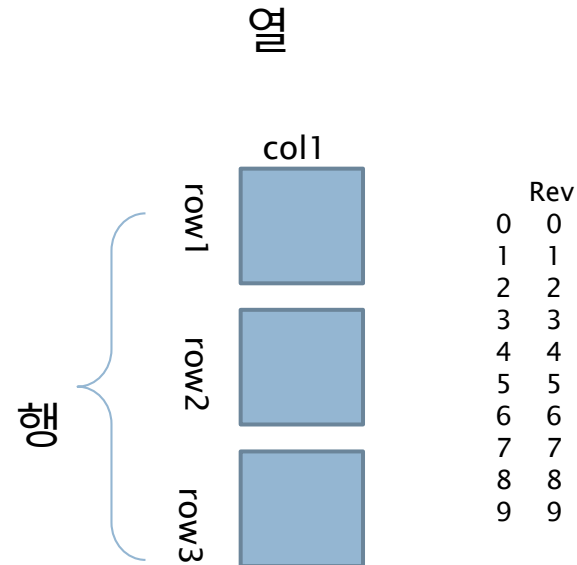
DataFrame은 기본적으로 column 명을 추가

```
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)
print(df)

df.columns = ['Rev']
print(df)
```



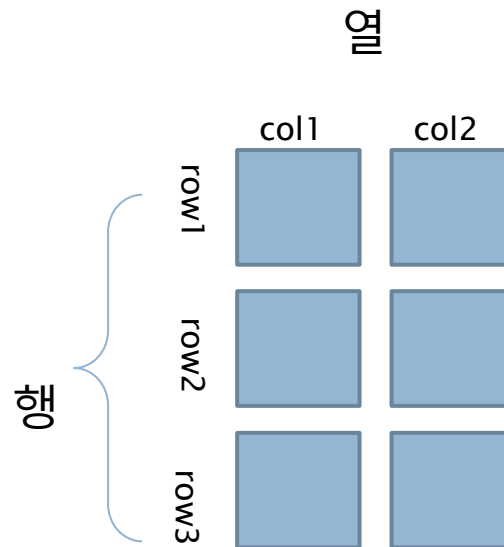
DataFrame 칼럼 추가:칼럼복사

DataFrame은 기존에 없는 column에 칼럼을 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']
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

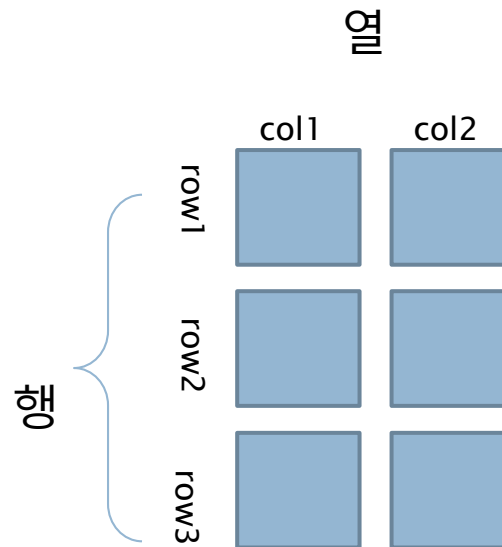
DataFrame 칼럼 추가: 값

DataFrame은 기존에 없는 column에 값을 할당시
행에 맞춰 칼럼을 추가

```
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

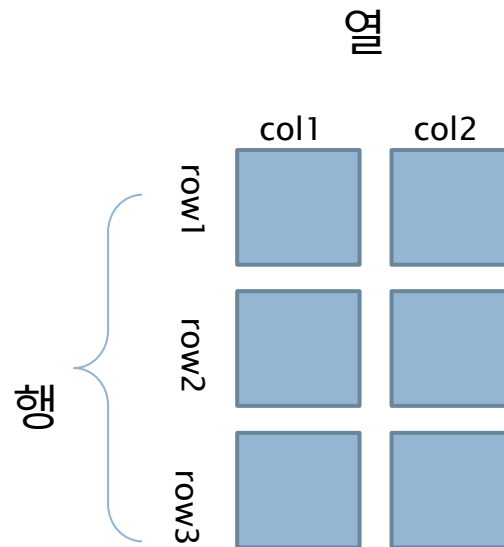
DataFrame 칼럼값 변경

DataFrame은 기존에 존재한 column에 값을 추가할 경우 행에 맞춰 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

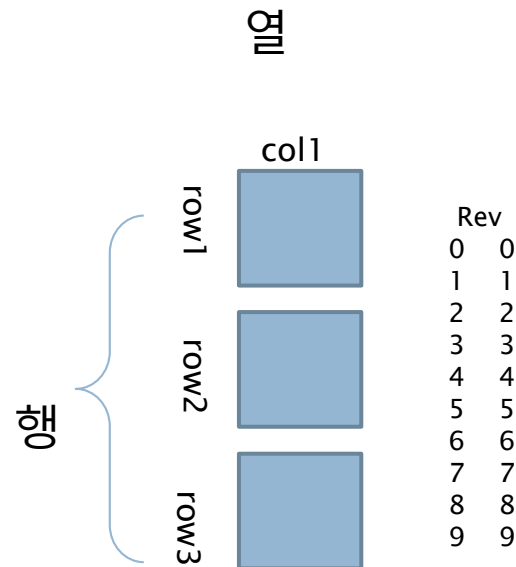
DataFrame 칼럼 삭제

DataFrame은 기존에 존재한 column을 del로 삭제

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



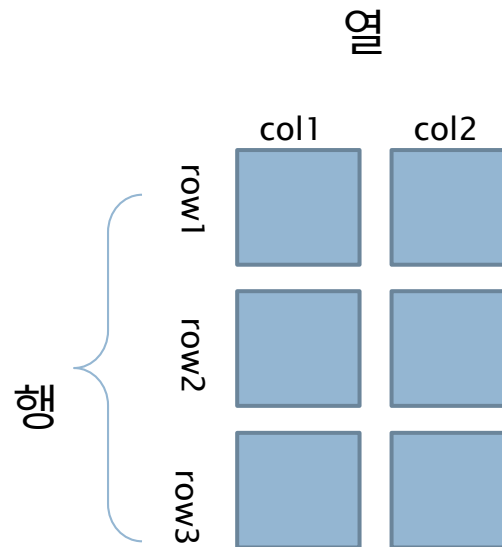
DataFrame 행 이름 부여

DataFrame은 기존에 행에 이름을 부여(index 속성)

```
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

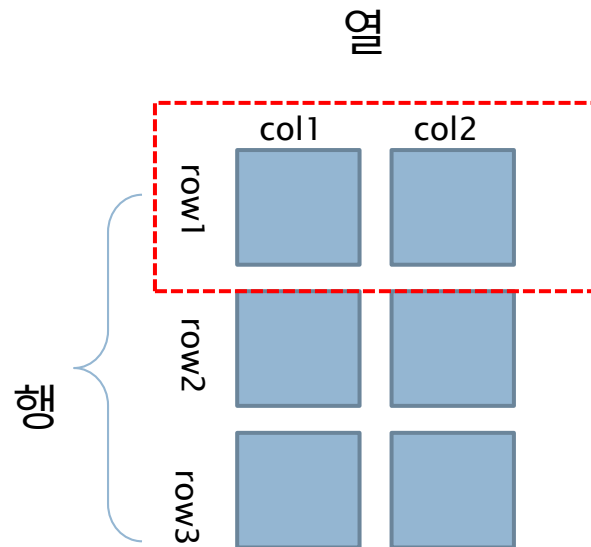
DataFrame 단일 행 검색

DataFrame은 단일 행을 인덱스 방식([])

```
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'])
```



```
Rev    0
col    0
Name: a, dtype:
int64
```

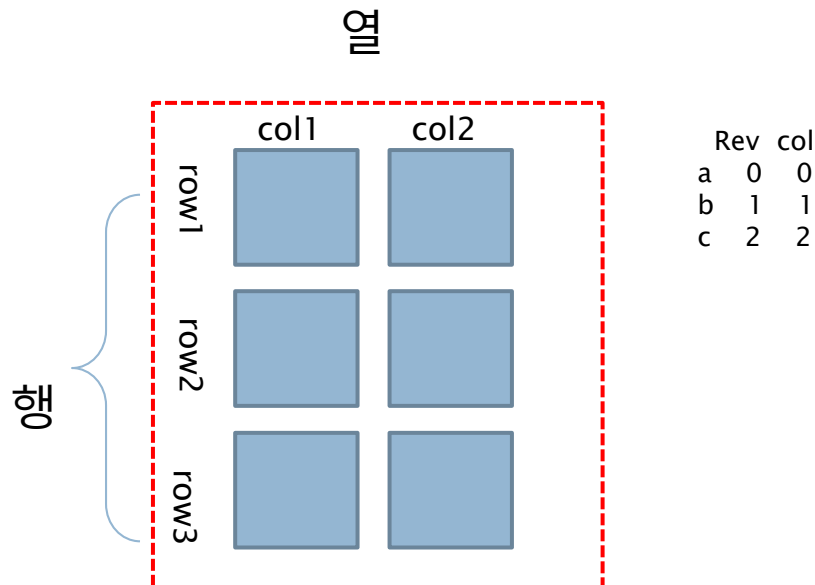
DataFrame 멀티 행 검색

DataFrame은 멀티행을 슬라이싱 방식([:])으로 검색

```
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'])
```



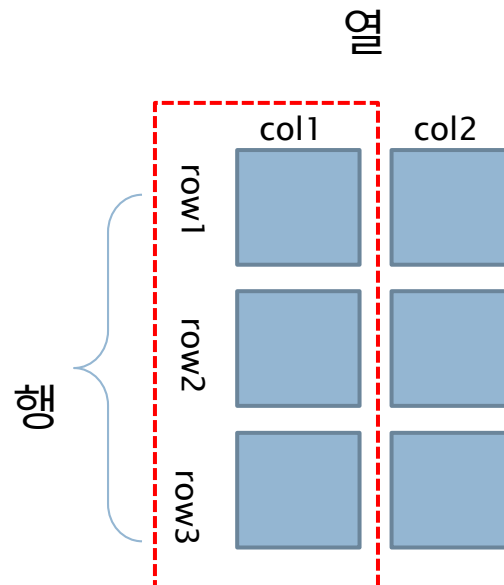
DataFrame 단일 열 검색

DataFrame은 단일 열을 인덱스 방식([])

```
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'])
```



```
a 0
b 1
c 2
d 3
e 4
f 5
g 6
h 7
i 8
j 9
Name: Rev, dtype:
int64
```

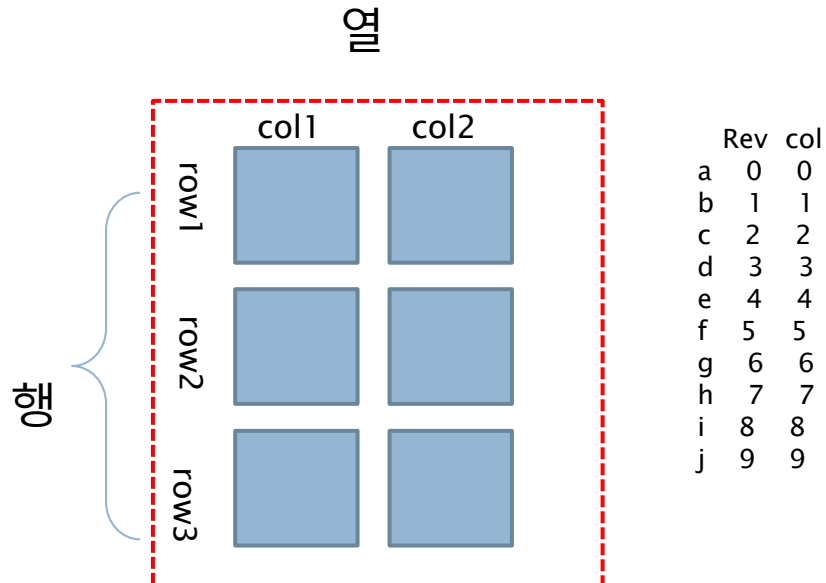

DataFrame 멀티 열 검색

DataFrame은 멀티 열은 리스트 방식([[,]])으로 검색

```
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']])
```



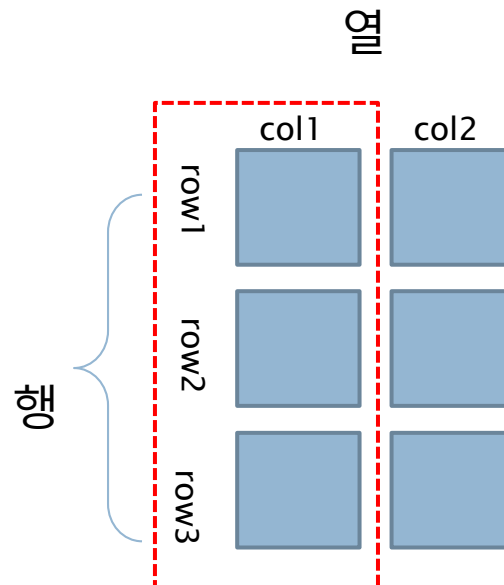
DataFrame 행과열 검색 1

DataFrame은 ix 속성을 이용해서 행과 열을 동시에 검색 ([행(슬라이싱 :), 칼럼(명)])

```
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.ix[0:3,'Rev'])
```



```
a 0
b 1
c 2
Name: Rev, dtype:
int64
```

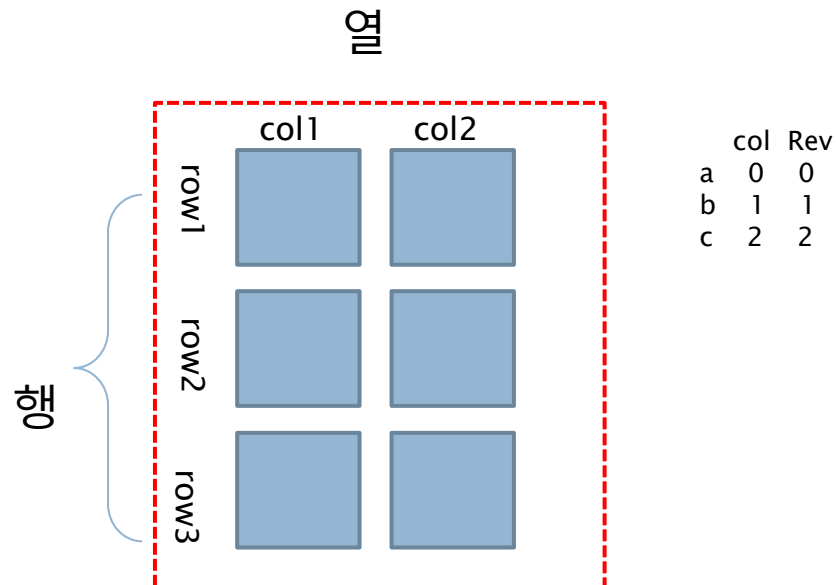
DataFrame 행과열 검색 2

DataFrame은 ix 속성을 이용해서 행과 복수의 열을 동시에 검색 ([행(슬라이싱 :), [칼럼명,칼럼명])

```
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.ix[:3,['col', 'Rev']])
```



DataFrame head 검색

DataFrame은 head()메소드를 이용해서 default=5까지 검색

```
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())
```

예

	col1	col2
row1		
row2		
row3		

행

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

DataFrame tail 검색

DataFrame은 tail()메소드를 이용해서 default=5
까지 검색

```
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())
```

예

	col1	col2
row1		
row2		
row3		

행

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



DataFrame 생성

DataFrame 생성: list

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: dict

column단위로 리스트를 만들어서 dict에 대입해서 데이터를 생성

```
import pandas as pd

names = ['Bob','Jessica','Mary','John','Mel']
births = [968, 155, 77, 578, 973]

dd = {'names': names, 'births':births}
df1 = pd.DataFrame(dd)

print(df1)
```

	births	names
0	968	Bob
1	155	Jessica
2	77	Mary
3	578	John
4	973	Mel

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)
```

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

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



DataFrame 접근

DataFrame 접근: column

칼럼을 기준으로 접근해서 데이터 검색

```
import numpy as np
import pandas as pd

A = np.array(['one', 'one', 'two', 'two', 'three',
              'three'])
B = np.array(['start', 'end']*3)
C = [np.random.randint(10, 99, 6)]*6
df = pd.DataFrame(zip(A, B, C), columns=['A',
                                         'B', 'C'])

print(df)
print(df.index)
print(df.columns)
print(df["A"])
```

```
      A      B      C
0  one  start  [71, 58, 23, 79, 19, 93]
1  one   end  [71, 58, 23, 79, 19, 93]
2  two  start  [71, 58, 23, 79, 19, 93]
3  two   end  [71, 58, 23, 79, 19, 93]
4 three  start  [71, 58, 23, 79, 19, 93]
5 three   end  [71, 58, 23, 79, 19, 93]
Int64Index([0, 1, 2, 3, 4, 5], dtype='int64')
Index([u'A', u'B', u'C'], dtype='object')
0    one
1    one
2    two
3    two
4   three
5   three
Name: A, dtype: object
```

DataFrame 접근: 여러 개 column

여러 개의 칼럼을 기준으로 접근해서 데이터 검색

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

```
A = np.array(['one', 'one', 'two', 'two', 'three',
              'three'])
B = np.array(['start', 'end']*3)
C = [np.random.randint(10, 99, 6)]*6
df = pd.DataFrame(zip(A, B, C), columns=['A',
                                         'B', 'C'])
```

```
print(df)
print(df.index)
print(df.columns)
print(df[["A", "C"]])
```

```
      A      B      C
0  one  start  [71, 58, 23, 79, 19, 93]
1  one   end  [71, 58, 23, 79, 19, 93]
2  two  start  [71, 58, 23, 79, 19, 93]
3  two   end  [71, 58, 23, 79, 19, 93]
4 three  start  [71, 58, 23, 79, 19, 93]
5 three   end  [71, 58, 23, 79, 19, 93]
Int64Index([0, 1, 2, 3, 4, 5], dtype='int64')
Index([u'A', u'B', u'C'], dtype='object')
```

```
      A      C
0  one  [16, 24, 37, 65, 43, 64]
1  one  [16, 24, 37, 65, 43, 64]
2  two  [16, 24, 37, 65, 43, 64]
3  two  [16, 24, 37, 65, 43, 64]
4 three  [16, 24, 37, 65, 43, 64]
5 three  [16, 24, 37, 65, 43, 64]
```

DataFrame 접근: row

행 기준으로 데이터를 접근해서 검색

```
import numpy as np
import pandas as pd

A = np.array(['one', 'one', 'two', 'two', 'three',
              'three'])
B = np.array(['start', 'end']*3)
C = [np.random.randint(10, 99, 6)]*6
df = pd.DataFrame(zip(A, B, C), columns=['A',
                                         'B', 'C'])

print(df.ix[0])
print(df.ix[0:2])
```

```
A          one
B          start
C  [74, 50, 77, 81, 47, 58]
Name: 0, dtype: object
   A  B          C
0 one start [74, 50, 77, 81, 47, 58]
1 one  end [74, 50, 77, 81, 47, 58]
2 two start [74, 50, 77, 81, 47, 58]
```

row 접근시 슬라이싱 계산차이

행 기준으로 접근시 DataFrame으로 슬라이싱과 속성에
서 슬라이싱하는 경우 실제 출력되는 개수가 다름

```
import numpy as np
import pandas as pd

A = np.array(['one', 'one', 'two', 'two', 'three',
              'three'])
B = np.array(['start', 'end']*3)
C = [np.random.randint(10, 99, 6)]*6
df = pd.DataFrame(zip(A, B, C), columns=['A',
                                          'B', 'C'])

print(".....", df.ix[0:1])
print(".....", df[0:1])
```

```
('.....',   A   B           C
0 one start [58, 26, 58, 15, 64, 97]
1 one end  [58, 26, 58, 15, 64, 97])
('.....',   A   B           C
0 one start [58, 26, 58, 15, 64, 97])
```



DataFrame multi column 접근

DataFrame 조회 : index

DataFrame은 column 기반이 접근해서 조회함

```
import pandas as pd
import numpy as np

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)
print(df['bar'])
print(df['bar','one'])
print(df['bar']['one'])
```

```
first      bar      baz      foo      qux second
one      two      one      two      one      two      one
A   -0.109192  1.560258 -0.402283  0.565411 -0.964342
   0.603564 -1.014468
B   -0.916891 -0.818499 -0.307554 -0.317725 -0.582503
   0.823485 -0.032378
C    1.113378  0.572111  0.685743 -1.341933 -1.916343
   0.288700  0.949549
```

```
first
second      two
A      1.318373
B      0.337134
C     -2.028451
second      one      two
A   -0.109192  1.560258
B   -0.916891 -0.818499
C    1.113378  0.572111
```

```
A   -0.109192
B   -0.916891
C    1.113378
Name: (bar, one), dtype: float64
```

```
A   -0.109192
B   -0.916891
C    1.113378
Name: one, dtype: float64
```


DataFrame multi index 접근

DataFrame 조회 : loc메소드

DataFrame은 row 기반이 접근해서 조회함
[:] 슬라이싱 오류가 발생하므로iloc메소드 사용

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

```
A = np.array(['one', 'one', 'two', 'two', 'three', 'three'])
B = np.array(['start', 'end']*3)
C = [np.random.randint(10, 99, 6)]*6
df = pd.DataFrame(zip(A, B, C), columns=['A', 'B', 'C'])
#2개 열을 index로 지정
df.set_index(['A', 'B'], inplace=True)
print(df)
print("df.loc", df.loc['one'])
print("df.loc[one, start]", df.loc['one', 'start'])
```

```
(df.loc,          C
B
start [74, 15, 47, 75, 11, 49]
end   [74, 15, 47, 75, 11, 49])
(df.loc[one, start], C [74, 15, 47, 75, 11, 49]
Name: (one, start), dtype: object)
```

DataFrame 조회 : iloc메소드

DataFrame은 row 기반으로 [:] 연산

```
import numpy as np
import pandas as pd

A = np.array(['one', 'one', 'two', 'two', 'three', 'three'])
B = np.array(['start', 'end']*3)
C = [np.random.randint(10, 99, 6)]*6
df = pd.DataFrame(zip(A, B, C), columns=['A', 'B', 'C'])
#2개 열을 index로 지정
df.set_index(['A', 'B'], inplace=True)
print(df)
print("df.iloc[0:3]", df.iloc[0:3])
```

```
('df.iloc[0:3]',  
A  B  
one start [98, 69, 80, 53, 93, 14]  
end [98, 69, 80, 53, 93, 14]  
two start [98, 69, 80, 53, 93, 14])
```

C



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
ftypes	Return the ftypes (indication of sparse/dense and dtype) in this object.
index	행에 대한 접근 표시
columns	칼럼에 대한 접근 표시
axes	행과 열에 대한 축을 접근 표시
empty	DataFrame 내부가 없으면 True 원소가 있으면 False
T	행과 열을 변환

내부 값 접근 속성

변수	설명
at	Fast label-based scalar accessor
blocks	Internal property, property synonym for as_blocks()
iat	Fast integer location scalar accessor.
iloc	Purely integer-location based indexing for selection by position.
ix	A primarily label-location based indexer, with integer position fallback.
loc	Purely label-location based indexer for selection by label.
values	Numpy 로 변환

DataFrame attribute 예시:1

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)
print(df.shape)
print(df.index) # 행에 대한 부분
print(df.columns) #칼럼에 대한 부분
print(df.axes) # 행렬에 대한 축
```

```
Names Births
0    Bob    968
1  Jessica    155
2    Mary     77
3    John    578
4     Mel    973
(5, 2)
Int64Index([0, 1, 2, 3, 4], dtype='int64')
Index([u'Names', u'Births'], dtype='object')
[Int64Index([0, 1, 2, 3, 4], dtype='int64'),
Index([u'Names', u'Births'], dtype='object')]
```

DataFrame attribute 예시:2

Dtype, at, ndim에 대한 속성 값들을 확인

```
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.dtypes)
print(df.at.__dict__)
print("scalar accessor ",df.at[0,'Names'])
print(df.ndim, type(df.ndim))
```

```
Names      object
Births     int64
dtype: object
{'ndim': 2, 'obj': 0}
Names Births
0   Bob   968
1  Jessica 155
2   Mary   77
3   John  578
4    Mel  973, 'name': 'at', 'axis': None}
('scalar accessor ', 'Bob')
(2, <type 'int'>)
```


DataFrame attribute 예시:3

blocks 속성에서 column 단위로 데이터 검색

```
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.blocks, type(df.blocks))
print(df.blocks['object'])
print(df.blocks['int64'])
```

```
{{'object':  Names
0    Bob
1  Jessica
2    Mary
3    John
4    Mel, 'int64':  Births
0    968
1    155
2     77
3    578
4    973}, <type 'dict'>)}

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

DataFrame attribute 예시:4

Empty, ftypes, iat(행과열 숫자위치), iloc(행), ix(행) 등을 이용해서 검색

```
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.empty)
print(df.ftypes)
print(df.iat.__dict__)
print("scalar accessor ",df.iat[0,1])
print(df.iloc.__dict__)
print(df.iloc[0])
print(df.ix[0])
```

```
False
Names    object:dense
Births   int64:dense
dtype: object
{'ndim': 2, 'obj':  Names Births
0   Bob    968
1  Jessica  155
2   Mary    77
3   John   578
4   Mel   973, 'name': 'iat', 'axis': None}
('scalar accessor ', 968)
{'ndim': 2, 'obj':  Names Births
0   Bob    968
1  Jessica  155
2   Mary    77
3   John   578
4   Mel   973, 'name': 'iloc', 'axis': None}
Names    Bob
Births   968
Name: 0, dtype: object
Names    Bob
Births   968
Name: 0, dtype: object
```

DataFrame attribute 예시:5

T(행과 열을 전환), size(원소갯수), values(값을 numpy로 전환), loc(lable로 검색)을 이용해서 검색

```
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.T)
print(df.size)
print(df.values)
print(df.loc[0,'Names'])
```

```
0      1      2      3      4
Names  Bob  Jessica  Mary  John  Mel
Births 968    155    77   578   973
10
[['Bob' 968L]
 ['Jessica' 155L]
 ['Mary' 77L]
 ['John' 578L]
 ['Mel' 973L]]
Bob
```



DataFrame 메소드

Groupby : 1 칼럼

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)
one = df1.groupby('letter')
print(one.sum())
```

	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

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

Groupby : 여러 칼럼 1

칼럼기준을 그룹을 연계해서 서비스 진행하지만 인덱스가 multi index로 변함

```
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)
letterone =
df1.groupby(['letter','one']).sum()
print(letterone)
```

	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

		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	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

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

```
Int64Index([0, 1, 2],
            dtype='int64')
```

MULTIINDEX CLASS



MultiIndex 구조

MultIndex

Index나 column에 대한 메타데이터에 대한 객체화

```
class MultiIndex(pandas.indexes.base.Index)
| A multi-level, or hierarchical, index object for pandas objects
|
| Parameters
| -----
| levels : sequence of arrays
|     The unique labels for each level
| labels : 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 : boolean, default False
|     Copy the meta-data
| verify_integrity : boolean, default True
|     Check that the levels/labels are consistent and valid
|
```

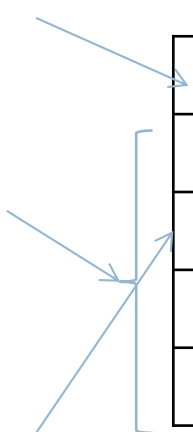
표에 대한 메타데이터 관리

실제 데이터를 접근할 때 별도의 메타데이터로 관리가 필요할 경우

names는 각 열의 명을 관리

levels는 각 열의 대표값을 list로 관리

labels는 각 열의 실제 위치를 관리



number	color
1	red
1	blue
2	red
2	blue



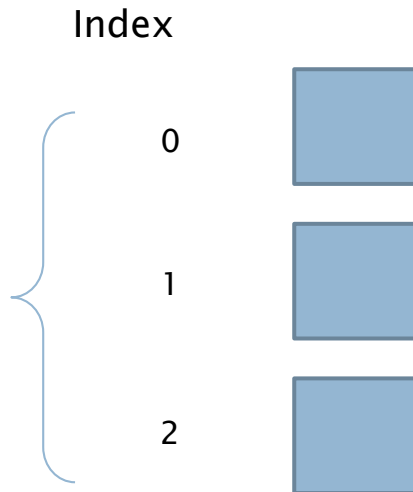
객체화

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

Index에 대한 객체화 : 1

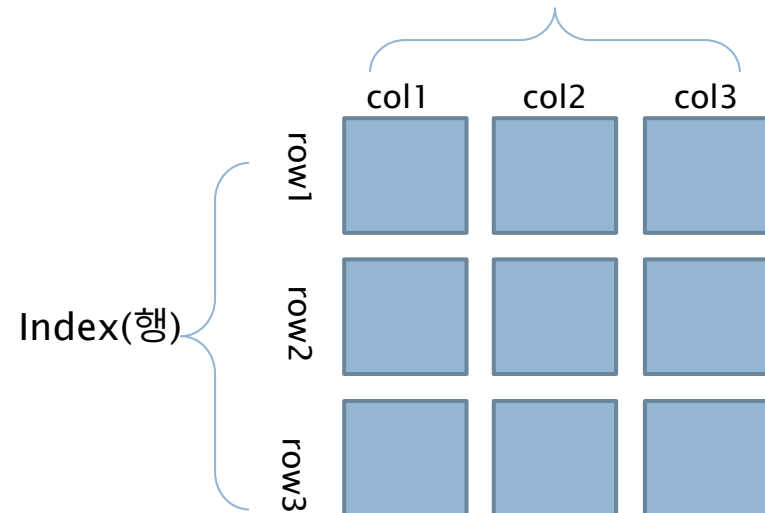
Index에 대한 정보관리를 객체화하여 데이터부분에 대한 접근을 지원

Series



DataFrame

Column(열)



Index에 대한 객체화 : 2

Levels, labels, names으로 분리해서 접근할 수 있는 정보를 관리

Index(행)

names

Levels에 대한 명

levels

Index에 대한 이름관리

labels

Index에 대한 위치관리

		Column(열)		
		col1	col2	col3
Index(행)	row1			
	row2			
	row3			

Column(열)

names

Levels에 대한 명

levels

Column에 대한 이름관리

labels

Column에 대한 위치관리



MultiIndex 생성

MultIndex 생성하기: tuple

Tuple 형태로 전달시 계층에 대한 이름(levels), 각 이름 별 계층 위치(labels) 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)
```

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

MultIndex 생성하기: array

List로 lable 형태로 전달시 계층에 대한 이름 (levels), 각 이름별 계층 위치(labels) 그리고 level 에 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(mi1)
```

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


MultIndex 생성하기: product

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

```
import pandas as pd

numbers = [0, 1, 2]
colors = [u'green', u'purple']
mi2 = pd.MultiIndex.from_product([numbers, colors], names=['number', 'color'])
print(mi2)
```

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



MultiIndex 내부 구조

MultIndex : levels

Levels 에 대한 조회

```
import pandas as pd

midx =
pd.MultiIndex.from_product([list(range(3)),['a','b','c'],pd.date_range('20130101',
periods=3)],names=['numbers','letters','dates'])
print(midx.levels[0])
print(midx.levels[1])
print(midx.levels[2])
```

```
Int64Index([0, 1, 2], dtype='int64', name=u'numbers')
Index([u'a', u'b', u'c'], dtype='object', name=u'letters')
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03'], dtype='datetime64[ns]', name=u'dates',
freq='D')
```


MultIndex : level value

Label에 level 값을 표시해서 조회

```
import pandas as pd

midx =
pd.MultiIndex.from_product([list(range(3)),['a','b','c'],pd.date_range('20130101',
periods=3)],names=['numbers','letters','dates'])

print(midx.get_level_values(0))
print(midx.get_level_values(1))
print(midx.get_level_values(2))
```

```
Int64Index([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2], dtype='int64', name='numbers')
Index([u'a', u'a', u'a', u'b', u'b', u'b', u'c', u'c', u'c', u'a', u'a', u'a', u'b', u'b', u'b', u'c', u'c', u'c', u'a', u'a', u'a', u'b', u'b', u'b',
      u'c', u'c', u'c'], dtype='object', name='letters')
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-01',
              '2013-01-02', '2013-01-03', '2013-01-01', '2013-01-02', '2013-01-03',
              '2013-01-01', '2013-01-02', '2013-01-03', '2013-01-01', '2013-01-02', '2013-01-03',
              '2013-01-01', '2013-01-02', '2013-01-03'], dtype='datetime64[ns]', name='dates', freq='D')
```

MultIndex : names

multiIndex 내의 names에 대한 세부 조회

```
import pandas as pd

midx =
pd.MultiIndex.from_product([list(range(3)),['a','b','c'],pd.date_range('20130101',
periods=3)],names=['numbers','letters','dates'])

print(midx.names, type(midx.names))
print(midx.names.index('letters'))
print(midx.names.index('dates'))
```

(FrozenList([u'numbers', u'letters', u'dates']), <class 'pandas.core.base.FrozenList'>)

1
2

3차원 데이터 관리 (PANEL)

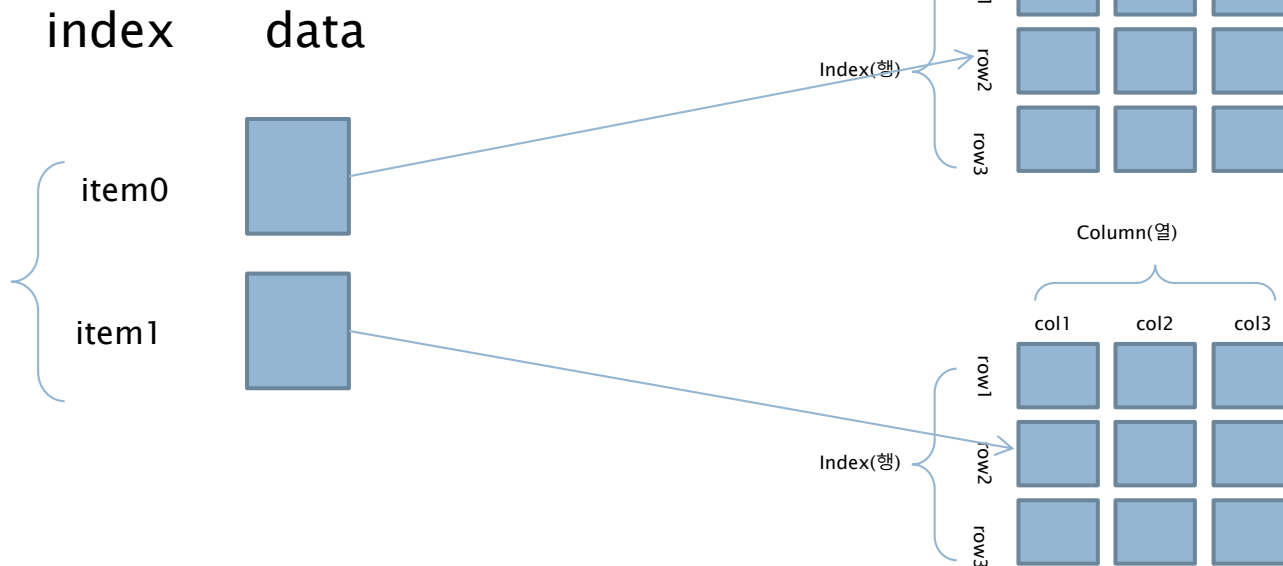


Panel 구조

Panel 구조

3차원의 데이터를 관리하는 컨테이너

```
data = {'Item1' : pd.DataFrame(np.random.randn(3, 3)),  
        'Item2' : pd.DataFrame(np.random.randn(3, 3))}  
pd.Panel(data)
```



Panel

items: axis 0, item은 dataframe과 매핑

major_axis: axis 1, 행으로 구성된 dataframe

minor_axis: axis 2, 열로 구성된 dataframe

```
class Panel(pandas.core.generic.NDFrame)
| Represents wide format panel data, stored as 3-dimensional array
|
| Parameters
| -----
| data : ndarray (items x major x minor), or dict of DataFrames
| items : Index or array-like
|         axis=0
| major_axis : Index or array-like
|         axis=1
| minor_axis : Index or array-like
|         axis=2
| dtype : dtype, default None
|         Data type to force, otherwise infer
| copy : boolean, default False
|         Copy data from inputs. Only affects DataFrame / 2d ndarray input
```

Panel 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
ftypes	Return the ftypes (indication of sparse/dense and dtype) in this object.
axes	행과 열에 대한 축을 접근 표시
empty	DataFrame 내부가 없으면 True 원소가 있으면 False

Panel 내부 값 접근 속성

변수	설명
at	Fast label-based scalar accessor
blocks	Internal property, property synonym for as_blocks()
iat	Fast integer location scalar accessor.
iloc	Purely integer-location based indexing for selection by position.
ix	A primarily label-location based indexer, with integer position fallback.
loc	Purely label-location based indexer for selection by label.
values	Numpy 로 변환



Panel 생성

Data만 넣고 생성하기

3차원 데이터를 넣고 생성

```
import pandas as pd
import numpy as np

pn = pd.Panel(np.random.rand(4,3,2))
print(pn)
```

```
<class 'pandas.core.panel.Panel'>
Dimensions: 4 (items) x 3 (major_axis) x 2 (minor_axis)
Items axis: 0 to 3
Major_axis axis: 0 to 2
Minor_axis axis: 0 to 1
```

Dict를 이용해서 생성하기

Dict 타입으로 데이터 만들어서 생성하기

```
import pandas as pd
import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}

wp1 = pd.Panel(data)
print(wp1)
```

```
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 4 (major_axis) x 3 (minor_axis)
Items axis: Item1 to Item2
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 2
```

파라미터 넣고 생성하기

3차원 데이터를 넣고 차원별 이름 붙이기

```
import pandas as pd
import numpy as np

wp = pd.Panel(np.random.randn(2, 5, 4), items=['Item1', 'Item2'],
              major_axis=pd.date_range('1/1/2000', periods=5), minor_axis=['A', 'B', 'C', 'D'])

print(wp)
```

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 5 (major_axis) x 4 (minor_axis)

Items axis: Item1 to Item2

Major_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Minor_axis axis: A to D



Panel 접근

데이터 접근 방법

Panel 클래스에서 데이터 접근법은 [] 연산자와 메소드를 이용해서 처리

Operation	Syntax	Result
item	wp[item]	DataFrame
major_axis label	wp.major_xs(val)	DataFrame
minor_axis label	wp.minor_xs(val)	DataFrame

데이터 접근: 0 차원

Panel 의 items를 기준으로 접근

```
import pandas as pd
import numpy as np

wp = pd.Panel(np.random.randn(2, 5, 4),
items=['Item1', 'Item2'],
major_axis=pd.date_range('1/1/2000',
periods=5),minor_axis=['A', 'B', 'C', 'D'])

print(wp)
print(wp['Item1'])
print(wp['Item2'])
```

```
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 5 (major_axis) x 4 (minor_axis)
Items axis: Item1 to Item2
Major_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00
Minor_axis axis: A to D
  A      B      C      D
2000-01-01 -0.755870  0.675745  0.299948 -0.364387
2000-01-02 -0.366579 -1.038088  0.626151  0.995836
2000-01-03 -3.048796 -0.356768 -1.935733 -1.011240
2000-01-04 -0.876230 -0.671853  0.707184  0.214709
2000-01-05 -1.036889  1.897133  0.542407 -2.027655
  A      B      C      D
2000-01-01  1.354674 -0.488009  0.775997  0.346407
2000-01-02 -1.793025  0.483439 -0.432747  0.234766
2000-01-03 -0.829370  0.092516 -1.332394 -0.257560
2000-01-04  0.643688  1.812432  0.684545 -0.050472
2000-01-05 -2.368791 -0.244515  0.397663 -0.054107
```

데이터 접근: 1차원

Major_axis로 접근

```
import pandas as pd
import numpy as np

wp = pd.Panel(np.random.randn(2, 5, 4),
items=['Item1', 'Item2'],
major_axis=pd.date_range('1/1/2000',
periods=5),minor_axis=['A', 'B', 'C', 'D'])

print(wp)
print(wp['Item1'])
print(wp['Item2'])
```

```
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 5 (major_axis) x 4 (minor_axis)
Items axis: Item1 to Item2
Major_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00
Minor_axis axis: A to D
  A      B      C      D
2000-01-01 -0.755870  0.675745  0.299948 -0.364387
2000-01-02 -0.366579 -1.038088  0.626151  0.995836
2000-01-03 -3.048796 -0.356768 -1.935733 -1.011240
2000-01-04 -0.876230 -0.671853  0.707184  0.214709
2000-01-05 -1.036889  1.897133  0.542407 -2.027655
  A      B      C      D
2000-01-01  1.354674 -0.488009  0.775997  0.346407
2000-01-02 -1.793025  0.483439 -0.432747  0.234766
2000-01-03 -0.829370  0.092516 -1.332394 -0.257560
2000-01-04  0.643688  1.812432  0.684545 -0.050472
2000-01-05 -2.368791 -0.244515  0.397663 -0.054107
```

데이터 접근: 2차원

Minor_axis로 접근

```
import pandas as pd
import numpy as np

wp = pd.Panel(np.random.randn(2, 5, 4),
items=['Item1', 'Item2'],
major_axis=pd.date_range('1/1/2000',
periods=5),minor_axis=['A', 'B', 'C', 'D'])

print(wp.minor_axis[1])
print(wp.minor_xs(wp.minor_axis[1]))
print(wp.minor_xs('B'))
```

```
B
      Item1  Item2
2000-01-01 -0.943572  0.540412
2000-01-02  0.009293  0.030380
2000-01-03  1.286393 -0.119081
2000-01-04 -0.726561  1.230658
2000-01-05  0.717234  0.367189
      Item1  Item2
2000-01-01 -0.943572  0.540412
2000-01-02  0.009293  0.030380
2000-01-03  1.286393 -0.119081
2000-01-04 -0.726561  1.230658
2000-01-05  0.717234  0.367189
```

INDEX CLASS



Index 구조

Index

Series 처리에 필요한 index 대한 메타데이터 객체 화

```
class Index(pandas.core.base.IndexOpsMixin, pandas.core.strings.StringAccessorMixin,
pandas.core.base.PandasObject)
| Immutable ndarray implementing an ordered, sliceable set. The basic object
| storing axis labels for all pandas objects
|
| Parameters
| -----
| data : array-like (1-dimensional)
| dtype : NumPy dtype (default: object)
| 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 생성

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.Int64Index([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([u'a', u'b', u'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')
```