

# 빅 데이터 분석

## Pandas

August 2020

Prepared by Prof. Youn-Sik Hong

본 강의 자료는 "Python for Data Analysis"(2<sup>nd</sup> Ed.)  
by William McKinney 내용을 참고하여 작성하였음

# 목차

**01 Pandas 소개**

02 Series

03 DataFrame

04 pandas 주요 기능: reindex, ffill, drop

05 pandas 주요 기능: selection, filtering

06 산술 연산, 함수 적용 및 정렬

# What is pandas ?

---

## ■ *Pandas* is Python package for data analysis.

- It provides built-in data structures which simplify the manipulation and analysis of data sets.
- Pandas is easy to use and powerful, but “**with great power comes great responsibility**”.

## ■ Reference site

- <http://pandas.pydata.org/pandas-docs/stable/>

# Pandas: Essential Concepts

---

- A **Series** is similar to a **dictionary**.

{ 'grades' : [50,90,100,45] }

index

values (*list*)

- A **DataFrame** is a **dictionary** of Series:

```
{  { 'names' : ['bob', 'ken', 'art', 'joe']}
  { 'grades' : [50,90,100,45] }
}
```

# 목차

01 Pandas 소개

**02 Series**

03 DataFrame

04 pandas 주요 기능: reindex, ffill, drop

05 pandas 주요 기능: selection, filtering

06 산술 연산, 함수 적용 및 정렬

# Series (1/5) – pandas-01. ipynb

## ■ Series는 **index-value** 쌍

- **value**는 1차원 배열(리스트)

## ■ It has a name called **index** associated with the array.

- index 를 지정하지 않으면
  - 0 ~ (N-1) 의 정수가 할당됨
  - N은 데이터 길이

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

```
0    4  
1    7  
2   -5  
3    3  
dtype: int64
```

## ■ Series 속성

- **index**
- **values**

```
obj.values
```

```
array([ 4,  7, -5,  3], dtype=int64)
```

```
obj.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

## Series (2/5) : index 속성

```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
```

Index 속성 설정

```
d      4
b      7
a     -5
c      3
dtype: int64
```

```
obj2.index
```

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

```
obj2['a']
```

index로 값 (values) 참조

```
-5
```

```
obj2[['c', 'a', 'd']]
```

obj2[ *[list of indexes]* ]

Inner bracket 은  
Python 리스트

```
c      3
a     -5
d      4
```

Series 객체

## Series (3/5) : 연산

### ■ 연산 결과 역시 Series 객체(index-value 구조)

```
obj2
d      6
b      7
a     -5
c      3
dtype: int64
```



```
obj2[obj2>0]
```

```
d      6
b      7
c      3
dtype: int64
```

```
obj2*2
```

```
d      12
b      14
a     -10
c       6
dtype: int64
```

```
np.exp(obj2)
```

```
d      403.428793
b     1096.633158
a       0.006738
c     20.085537
dtype: float64
```

**Quiz : 실행 결과는?**

```
'b' in obj2
'b' in obj2.index
'e' in obj2
6 in obj2.values
```



## Series (4/5) : Dictionary 객체를 Series 객체로 생성

### ■ Pandas의 Series 객체는 python의 dictionary 타입과 유사

- python의 dictionary 객체를 사용하여 Pandas의 Series 객체를 생성
  - dictionary 객체의 **key** → Series 객체의 **index**
- Dictionary 객체는 keys(), values()와 같이 메소드를 호출하지만,
  - Series 객체는 **index, values 속성**을 사용

```
sdata = {'Ohio':35000, 'Texas':71000, 'Oregon':16000, 'Utah':5000}
```

```
obj3 = pd.Series(sdata)  
obj3
```

```
Ohio      35000  
Texas     71000  
Oregon    16000  
Utah       5000  
dtype: int64
```

**Dictionary 객체  
(key-value)**

**Quiz : 실행 결과는?**

```
obj3.index  
obj3.values
```

## Series (5/5) : NaN – missing value (결측치)

sdata

Ohio	35000
Texas	71000
Oregon	16000
Utah	5000

```
states = ['California', 'Ohio', 'Oregon', 'Texas']  
obj4 = pd.Series(sdata, index=states)  
obj4
```

California	NaN
Ohio	35000.0
Oregon	16000.0
Texas	71000.0

dtype: float64

결측치를 갖고  
있는 원소는?

isnull, notnull

```
pd.isnull(obj4)
```

California	True
Ohio	False
Oregon	False
Texas	False

dtype: bool

```
pd.notnull(obj4)
```

California	False
Ohio	True
Oregon	True
Texas	True

dtype: bool

```
obj4[pd.isnull(obj4)]
```

California	NaN
------------	-----

dtype: float64

# Practice #1

## ■ Self learning : Do it yourself !

```
sdata = {'Ohio': 35000, 'Texas': 71000,  
         'Oregon': 16000, 'Utah': 5000}  
obj3 = pd.Series(sdata)  
states = ['California', 'Ohio', 'Oregon', 'Texas']  
obj4 = pd.Series(sdata, index=states)
```

```
obj3  
obj4  
obj3 + obj4  
obj4.name = 'population'  
obj4.index.name = 'state'  
obj4
```

It automatically sort the results with index

The Series object and its index have the 'name' attributes.

# 목차

01 Pandas 소개

02 Series

**03 DataFrame**

04 pandas 주요 기능: reindex, ffill, drop

05 pandas 주요 기능: selection, filtering

06 산술 연산, 함수 적용 및 정렬

# DataFrame(1/6) – pandas-02. ipynb

## ■ DataFrame은 2차원 구조 : 행(row) + 열(column)

- 행과 열에 대한 index를 각각 갖고 있음

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],  
        'year': [2000, 2001, 2002, 2001, 2002, 2003],  
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}  
frame = pd.DataFrame(data)  
frame
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

열에 대한 index

```
frame.columns
```

```
Index(['state', 'year', 'pop'], dtype='object')
```

행에 대한 index (자동 생성)

```
frame.index
```

```
RangeIndex(start=0, stop=6, step=1)
```

Quiz : 실행 결과는?

```
frame.values
```

## DataFrame(2/6) : columns, index 속성 지정

```
frame = pd.DataFrame(data,
                      columns=['year', 'state', 'pop'],
                      index=['one', 'two', 'three', 'four', 'five', 'six'])
frame
```

열에 대한 index  
순서를 바꿈

행에 대한 index를  
새롭게 설정

	year	state	pop
one	2000	Ohio	1.5
two	2001	Ohio	1.7
three	2002	Ohio	3.6
four	2001	Nevada	2.4
five	2002	Nevada	2.9
six	2003	Nevada	3.2



	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

## DataFrame(3/6) : column 및 row 추출

	year	state	pop
one	2000	Ohio	1.5
two	2001	Ohio	1.7
three	2002	Ohio	3.6
four	2001	Nevada	2.4
five	2002	Nevada	2.9
six	2003	Nevada	3.2

```
frame['state']
#frame.state
```

**frame2['name of column']**

**frame2.name of column**

column 추출

```
one      Ohio
two      Ohio
three    Ohio
four     Nevada
five     Nevada
six      Nevada
Name: state, dtype: object
```

```
frame.loc['one']
```

**frame2.loc['name of row']**

row 추출

```
year      2000
state     Ohio
pop        1.5
Name: one, dtype: object
```

```
frame.iloc[2]
```

**frame2.iloc[index of row]**

row 추출

```
year      2002
state     Ohio
pop        3.6
Name: three, dtype: object
```

## DataFrame(4/6) : column 값 변경

```
new_cols = ['year', 'state', 'pop', 'debt']
old_index = ['one', 'two', 'three', 'four', 'five', 'six']
frame2 = pd.DataFrame(data, columns=new_cols, index=old_index)
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN



```
frame2['debt'] = 16.5
```

```
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

```
frame2['debt'] = np.arange(6.)
```

```
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	0.0
two	2001	Ohio	1.7	1.0
three	2002	Ohio	3.6	2.0
four	2001	Nevada	2.4	3.0
five	2002	Nevada	2.9	4.0
six	2003	Nevada	3.2	5.0



## DataFrame(5/6) : Series 객체를 사용한 column 값 변경

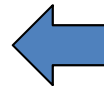
리스트(또는 배열)을 column에 할당할 경우 리스트 크기가 **DataFrame** 크기와 같아야 함.  
**Series** 객체를 **DataFrame**에 지정할 때, **index**에 해당하는 **column** 값을 지정하지 않은 경우 **NaN**을 할당.

```
val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
```

```
frame2['debt'] = val
```

```
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN



frame2				
	year	state	pop	debt
one	2000	Ohio	1.5	0
two	2001	Ohio	1.7	1
three	2002	Ohio	3.6	2
four	2001	Nevada	2.4	3
five	2002	Nevada	2.9	4
six	2003	Nevada	3.2	5

## Practice #2

### ■ Self learning : Do it yourself !

```
frame2['eastern'] = frame2.state == 'Ohio'
frame2
del frame2['eastern']
frame2.columns
frame2.values
```

어떤 컬럼이 새롭게 만들어질까요?

delete the column



frame2				
	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

Quiz :

DataFrame의 행을 없애려면?

## DataFrame (6/6) : Nested dictionaries

### ■ nested dictionary {...{...}}의 경우에도 DataFrame을 생성할 수 있다.

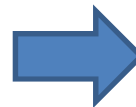
- 바깥쪽 dictionary의 key는 column, 안쪽 dictionary의 key는 row가 된다.

```
pop = {'Nevada': {2001: 2.4, 2002: 2.9},  
       'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

```
frame3 = pd.DataFrame(pop)  
frame3
```

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

index를 기준으로  
정렬되지 않았음



```
frame3 = pd.DataFrame(pop, index=[2000, 2001, 2002])
```

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

index를 기준으로 정렬

# 목차

01 Pandas 소개

02 Series

03 DataFrame

**04 pandas 주요 기능: reindex, ffill, drop**

05 pandas 주요 기능: selection, filtering

06 산술 연산, 함수 적용 및 정렬

# Reindex (1/4) : Series 객체 index 재설정 – pandas-03. ipynb

## ■ reindex : index를 재설정

- index 설정에 맞게 pandas 객체의 값(values)을 재배치.
- index에 해당되는 값이 없을 경우 **NaN** 할당

```
obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])  
obj
```

```
d    4.5  
b    7.2  
a   -5.3  
c    3.6  
dtype: float64
```

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

**index** 순서 변경  
새 **index**도 추가

```
a   -5.3  
b    7.2  
c    3.6  
d    4.5  
e   NaN  
dtype: float64
```

obj2.fill()



```
a   -5.3  
b    7.2  
c    3.6  
d    4.5  
e    4.5
```

## Reindex (2/4) : **reindex**와 함께 결측치를 채움

### ■ **ffill** : 결측치를 채워 넣을 때

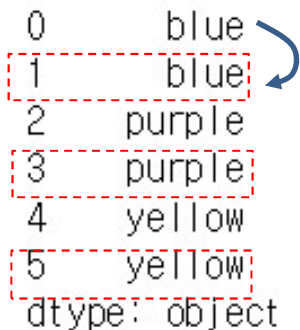
- 누락된 값(missing value)을 이전 항목의 값으로 채움.

```
obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
obj3
```

```
0      blue
2    purple
4    yellow
dtype: object
```

```
obj3.reindex(range(6), method='ffill')
```

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



**reindex**와 **ffill** 메소드를 따로 실행시킬 수도 있음.

```
obj4 = obj3.reindex(range(6))
#obj3.reindex(np.arange(6)) - 갯수가 많으면 numpy 사용!
obj4.ffill()
```

## Reindex (3/4) : DataFrame index 재설정(row-wise)

### ■ DataFrame의 reindex : 행과 열 모두 기존 index를 새롭게 변경 가능

- 행은 리스트 형태로 전달

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

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



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

```
frame2.ffill(axis=0)
```

	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



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

**ffill이 적용될 axis를 설정  
axis=0, column-wise**

## Reindex (4/4) : DataFrame index 재설정(column-wise)

### ■ DataFrame의 reindex : 행과 열 모두 기존 index를 새롭게 변경 가능

- 열은 columns 속성에 지정

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

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



```
states = ['Texas', 'Utah', 'California']  
frame3 = frame.reindex(columns=states)
```

```
frame3.ffill(axis=1)
```

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



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

**ffill이 적용될 axis를 설정  
axis=1, row-wise**



# Dropping Entries from an Axis : **drop**

## ■ Series 객체에서 항목 제거

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

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

```
new_obj = obj.drop('c')  
new_obj
```

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

```
obj.drop('c', inplace=True)  
obj
```

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

The **inplace = True** option removes all discarded values from the actual object.

## Practice #3

### ■ Self learning : Do it yourself !

- DataFrame 객체에서 항목 제거: **column** 또는 **row** 제거

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),  
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],  
                    columns=['one', 'two', 'three', 'four'])  
  
data  
data.drop(['Colorado', 'Ohio'])  
data.drop(['Colorado', 'Ohio'], axis=0)  
data.drop('two', axis=1)  
data.drop(['two', 'four'], axis='columns')  
data.drop('Ohio', inplace=True)
```

Delete all values in the row.

Delete all values  
in the column.

# 목차

01 Pandas 소개

02 Series

03 DataFrame

04 pandas 주요 기능: reindex, ffill, drop

**05 pandas 주요 기능: selection, filtering**

06 산술 연산, 함수 적용 및 정렬

# Indexing, Selection, and Filtering (1/3) – pandas-04. ipynb

## ■ Series 객체에서 index를 사용한 항목 선택

```
obj = pd.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
```

Index('b') 또는  
정수 index(1)를 사용하여  
항목 선택

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

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



```
obj[[1, 0, 3]]
```

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

## Indexing, Selection, and Filtering (2/3) – pandas-04. ipynb

### ■ Series 객체에서 slicing을 통한 항목 선택

- 레이블을 사용한 index slicing의 경우 endpoint를 포함.
- 숫자 index slicing의 경우 endpoint 제외.

obj['b':'c']	obj[1:2]	obj[1:3]
b 1.0 c 2.0	b 5.0	b 5.0 c 5.0

dtype: float64

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

a 0.0 b 5.0 c 5.0 d 3.0
----------------------------------

dtype: float64

## Practice #4

### ■ Self learning : Do it yourself !

- DataFrame에서 행(row)은 slicing을 통해 범위를 지정할 수 있음.
- DataFrame에서 열(column)은 직접 column 이름으로 선택.

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

```
data
```

```
data[:2]
```

Select rows by slicing

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

```
data[['three', 'one']]
```

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

# Indexing, Selection, and Filtering (3/3)

## ■ DataFrame에서 조건식을 사용하여 항목 선택

```
data = pd.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 < 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

## Selection with loc and iloc (1/2)

- loc은 이름으로 선택할 때, iloc은 정수 index로 선택할 때 사용

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

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

```
two    5  
three  _6
```



```
data.iloc[1, [1,2]]
```

```
two    5  
three  _6
```



## Selection with loc and iloc (2/2)

- loc은 이름으로 선택할 때, iloc은 정수 index로 선택할 때 사용

	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.iloc[2, [3, 0, 1]]
```

```
four    11  
one      8  
two      9  
Name: Utah, dtype: int32
```

```
data.iloc[2]
```

```
one      8  
two      9  
three    10  
four     11  
Name: Utah, dtype: int32
```

```
data.iloc[[1, 2], [3, 0, 1]]
```

	four	one	two
Colorado	7	4	5
Utah	11	8	9

## Practice #5

---

### ■ Self learning : Do it yourself !

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),  
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],  
                    columns=['one', 'two', 'three', 'four'])  
  
data  
data.loc[:, 'Utah', 'two']  
data.iloc[:, :3][data.three > 5]  
data.iloc[:, :3]
```

# 목차

01 Pandas 소개

02 Series

03 DataFrame

04 pandas 주요 기능: reindex, ffill, drop

05 pandas 주요 기능: selection, filtering

**06 산술 연산, 함수 적용 및 정렬**

# Arithmetic methods with fill values (1/5) – pandas-05. ipynb

## ■ Pandas 객체간 산술 연산을 할 때,

- 일치하지 않는 index가 있으면, 이 index는 연산 결과에 합쳐진다.

```
s1 = pd.Series([7.3, -2.5, 3.4, 1.5],  
               index=['a', 'c', 'd', 'e'])  
s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],  
               index=['a', 'c', 'e', 'f', 'g'])
```

s1		s2	
a	7.3	a	-2.1
c	-2.5	c	3.6
d	3.4	e	-1.5
e	1.5	f	4.0
		g	3.1



s1 + s2	
a	5.2
c	1.1
d	NaN
e	0.0
f	NaN
g	NaN

- a, c, e는 2개 객체에 공통
- d, f, g는 각 객체 고유 index

공통 index는 정상적인 연산 실행  
독립 index는 NaN 값 할당

## Arithmetic methods with fill values (2/5)

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

df3 = df1 + df2

	b	c	d	e
Colorado	NaN	NaN	NaN	NaN
Ohio	3.0	NaN	6.0	NaN
Oregon	NaN	NaN	NaN	NaN
Texas	9.0	NaN	12.0	NaN
Utah	NaN	NaN	NaN	NaN

df3.ffill(axis=1)

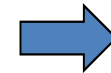
	b	c	d	e
Colorado	NaN	NaN	NaN	NaN
Ohio	3.0	3.0	6.0	6.0
Oregon	NaN	NaN	NaN	NaN
Texas	9.0	9.0	12.0	12.0
Utah	NaN	NaN	NaN	NaN

연산 후가 아니라 연산하기 전에  
초기값 설정이 필요!!

## Arithmetic methods with fill values (3/5)

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	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



```
df3 = df1.add(df2, fill_value=0)
```



	b	c	d	e
Colorado	6.0	7.0	8.0	0
Ohio	3.0	1.0	6.0	0
Oregon	0	0	0	0
Texas	9.0	4.0	12.0	0
Utah	0	0	0	0

	b	c	d	e
Colorado	0	0	0	NaN
Ohio	3.0	0	6.0	5.0
Oregon	9.0	NaN	10.0	11.0
Texas	9.0	4.0	12.0	8.0
Utah	0.0	NaN	1.0	2.0

	b	c	d	e
Colorado	6.0	7.0	8.0	NaN
Ohio	3.0	1.0	6.0	5.0
Oregon	9.0	NaN	10.0	11.0
Texas	9.0	4.0	12.0	8.0
Utah	0.0	NaN	1.0	2.0



```
df3.ffill(axis=1)
```

# Arithmetic methods with fill values (4/5)

```
df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),  
                    columns=list('abcd'))  
df2 = pd.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.loc[1, 'b'] = np.nan
```

**df2**

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	NaN	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	NaN	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	5.0	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

# Arithmetic methods with fill values (5/5)

- 산술 연산 메소드는 피 연산자(operand)의 순서를 바꾸는 역 연산(reverse) 존재

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

```
1 / df1 # df1.rdiv(1)
```

	a	b	c	d
0	inf	<u>1.000000</u>	0.500000	<u>0.333333</u>
1	<u>0.250000</u>	<u>0.200000</u>	<u>0.166667</u>	0.142857
2	0.125000	<u>0.111111</u>	<u>0.100000</u>	0.090909

```
df1.div(1)
```

	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

add → radd  
sub → rsub  
div → rdiv  
floordiv → rfloordiv  
mul → rmul  
pow → rpow

r means 'reverse'

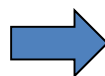


## 함수 적용과 mapping (1/2)

- Pandas 객체에도 numpy의 universal function을 적용할 수 있다.

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

	b	d	e
Utah	-1.170508	1.336384	1.570980
Ohio	0.949823	-0.735398	-2.103433
Texas	-0.130548	0.188836	0.254877
Oregon	0.671731	0.209810	0.491482



```
np.abs(frame)
```

	b	d	e
Utah	1.170508	1.336384	1.570980
Ohio	0.949823	0.735398	2.103433
Texas	0.130548	0.188836	0.254877
Oregon	0.671731	0.209810	0.491482

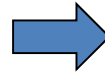
```
np.round(np.abs(frame), 3)
```

	b	d	e
Utah	1.171	1.336	1.571
Ohio	0.950	0.735	2.103
Texas	0.131	0.189	0.255
Oregon	0.672	0.210	0.491

## 함수 적용과 mapping (2/2)

### ■ 간단한 기능은 lambda 함수를 사용해 행 또는 열에 적용

	b	d	e
Utah	-1.170508	1.336384	1.570980
Ohio	0.949823	-0.735398	-2.103433
Texas	-0.130548	0.188836	0.254877
Oregon	0.671731	0.209810	0.491482



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

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

Column-wise  
operation(default)

```
b    0.723065  
d    1.459948  
e    3.428888  
dtype: float64
```

```
frame.apply(f, axis='columns')
```

```
Utah    0.773148  
Ohio    2.106767  
Texas   1.073389  
Oregon  2.501444  
dtype: float64
```

Row-wise operation

## Practice #6

### ■ Self learning : Do it yourself !

- **apply** 메소드는 Series 객체를 return 할 수 있음
- **applymap** 메소드는 서식을 적용할 때 사용

```
frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),
                      index=['Utah', 'Ohio', 'Texas', 'Oregon'])
frame
def f(x):
    return pd.Series([x.min(), x.max()], index=['min', 'max'])
frame.apply(f)
format = lambda x: '%.2f' %x
frame.applymap(format)
frame['e'].map(format)
```

# Sorting (1/2)

## ■ `sort_index` (axis)

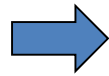
- `index(axis=0, default)` 또는 `column(axis=1)`을 기준으로 행 또는 열을 정렬.

## ■ `sort_values` ( )

- 값을 기준으로 정렬. 2차 행렬은 정렬 기준이 되는 column을 지정할 수 있음.

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

d	0
a	1
b	2
c	3



```
obj.sort_index()
```

a	1
b	2
c	3
d	0

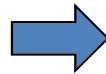
```
obj.sort_values()
```

d	0
a	1
b	2
c	3

## Sorting (2/2)

```
frame = pd.DataFrame(np.arange(8).reshape((2,4)),  
                      index=['three', 'one'],  
                      columns=['d', 'a', 'b', 'c'])
```

	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

row-wise

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

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

column-wise

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

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

내림차순

## Practice #7

---

### ■ Self learning : Do it yourself !

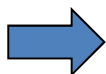
```
obj = pd.Series([4, 7, -3, 2])
obj.sort_values()
obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])
obj.sort_values()
frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})
frame
frame.sort_values(by='b')
frame.sort_values(by=['a', 'b'])
```

## 통계 및 요약 (1/2)

### ■ numpy와 달리 Pandas의 통계 관련 메소드는 NaN을 제외하고 연산

```
df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],  
                  [np.nan, np.nan], [0.75, -1.3]],  
                  index=['a', 'b', 'c', 'd'],  
                  columns=['one', 'two'])
```

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3



```
df.sum()
```

column-wise

```
one    9.25  
two   -5.80  
dtype: float64
```

```
df.sum(axis='rows') #df.sum(axis=0)
```

```
df.sum(axis='columns')
```

row-wise

```
a    1.40  
b    2.60  
c    0.00  
d   -0.55  
dtype: float64
```

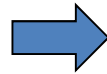
```
df.mean(axis='columns', skipna=False)
```

Do not exclude NaN

```
a    NaN  
b    1.300  
c    NaN  
d   -0.275  
dtype: float64
```

## 통계 및 요약 (2/2)

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3



```
df.idxmax()
```

```
one    b  
two    d  
dtype: object
```

```
df.cumsum()
```

	one	two
a	1.40	NaN
b	8.50	-4.5
c	NaN	NaN
d	9.25	-5.8

```
df.describe()
```

	one	two
count	3.000000	<a href="#">2.000000</a>
mean	3.083333	<a href="#">-2.900000</a>
std	3.493685	<a href="#">2.262742</a>
min	0.750000	-4.500000
25%	<a href="#">1.075000</a>	-3.700000
50%	1.400000	<a href="#">-2.900000</a>
75%	<a href="#">4.250000</a>	<a href="#">-2.100000</a>
max	7.100000	-1.300000



# Unique Values, Value Counts, and Membership (1/2)

## ■ **unique**: 중복되지 않은 값만을 numpy 배열 객체(ndarray) 객체로 반환

- 발생 순서대로 찾은 결과(정렬되지 않았음).

```
obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

```
uniques = obj.unique()
```

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

```
set(obj.values)
```

```
{'a', 'b', 'c', 'd'}
```

## ■ **value\_counts** : 값이 발생한 빈도

```
obj.value_counts()
```

a	3
c	3
b	2
d	1

```
pd.value_counts(obj.values, sort=False)
```

c	3
d	1
b	2
a	3

- 배열이나 리스트 등 다른 자료구조에서도 사용 가능

## Unique Values, Value Counts, and Membership (2/2)

```
data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],  
                    'Qu2': [2, 3, 1, 2, 3],  
                    'Qu3': [1, 5, 2, 4, 4]})  
data
```

	Qu1	Qu2	Qu3
0	1	2	1
1	3	3	5
2	4	1	2
3	3	2	4
4	4	3	4



DataFrame은 모두  
5개의 값을 갖고 있음  
(1, 2, 3, 4, 5)

```
result = data.apply(pd.value_counts).fillna(0)
```

result

	Qu1	Qu2	Qu3
1	1.0	1.0	1.0
2	0.0	2.0	1.0
3	2.0	2.0	0.0
4	2.0	0.0	2.0
5	0.0	0.0	1.0

Each value indicates  
how many times  
the value occurred  
in each column.