
▣ Ch04 데이터 프레임(Data Frame) 자료 다루기

(Ch16과 Ch04 내용을 함께 구성한다.)

- > 외장 **Pandas** 자료형
- > **pandas**라는 패키지를 **import** 해서 사용 가능한 추가 자료 구조
- > Data Type : **series, dataframe**
- > 관련 함수 : **Series(), DataFrame()**

○ 시리즈(Series) 형

- > **1차원 배열**로 여러 값을 나열한 자료 구조
- > 각 데이터 항목은 **index**로 식별된다.
- > 데이터 프레임의 데이터를 구성하는 열 값들을 시리즈로 추출하여 조작할 수 있다.
- > **pandas** 패키지를 통해 사용할 수 있다.

```
In [1]: ## 시리즈(Series) 형 ##
## 생성
import pandas as pd
a = pd.Series([11, 22, 33, 44], index = [1, 2, 3, 4]) #index 강제 부여
a
```

```
Out[1]: 1    11
2    22
3    33
4    44
dtype: int64
```

○ 데이터 프레임(Data Frame) 형

- > 행과 열로 나열되는 **2차원적** 자료 구조
- > 행과 열 값들은 딕셔너리 구조를 활용하여 초기화 한다.
- > 하나의 열 값들은 시리즈형으로 추출하여 사용 가능하다.
- > 복수 개의 열 값들은 데이터 프레임형으로 추출하여 사용 가능하다.
- > **pandas** 패키지를 통해 사용할 수 있다.

[] 데이터 프레임 생성

```
In [84]: ## [Data Frame] 생성
import pandas as pd
df = pd.DataFrame({'Class' : [1, 1, 2],
                   'Eng' : [87, 79, 80],
                   'Mat' : [80, 90, 80]}) #index 자동 부여
df
```

```
Out[84]:   Class  Eng  Mat
0       1    87    80
1       1    79    90
2       2    80    80
```

```
In [1]: ## [Data Frame] 생성
import pandas as pd
df = pd.DataFrame({'Class' : [1, 1, 2],
                   'Eng' : [87, 79, 80],
                   'Mat' : [80, 90, 80]},
                   index = ['Kims', 'Lees', 'Parks']) #index 강제 부여
df
```

```
Out[1]:   Class  Eng  Mat
Kims     1    87    80
Lees     1    79    90
Parks    2    80    80
```

[] 데이터 프레임 데이터 추출

> 열 변수를 사용한 열의 행 추출

```
In [35]: ## [Data Frame] 데이터 추출 : 열 변수로 열 추출
df['Eng']
```

```
Out[35]: Kims    87
Lees    79
Parks   80
Name: Eng, dtype: int64
```

```
In [36]: ## [Data Frame] 데이터 추출 : 열 변수로 열 추출
df[['Eng', 'Mat']]
```

```
Out[36]:   Eng  Mat
Kims   87   80
Lees   79   90
Parks  80   80
```

```
In [37]: ## [Data Frame] 데이터 추출 : 열 변수로 열 추출
x = df['Eng'];           # 'Eng' 열 값을 시리즈(Series)로 추출
```

```
print(type(x))
x

<class 'pandas.core.series.Series'>
Kims      87
Lees      79
Parks     80
Name: Eng, dtype: int64
```

```
In [47]: ## [Data Frame] 데이터 추출 : 열 변수로 열 추출
x = df[['Eng']]           # 'Eng' 열 값들을 시리즈(Series)로 추출
print(type(x))
x
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Out[47]:    Eng
Kims    87
Lees    79
Parks   80
```

```
In [38]: ## [Data Frame] 데이터 추출 : 열 변수로 열 추출
x = df[['Eng', 'Mat']];      # 'Eng'와 'Mat' 열 값들을 Data Frame으로 추출
print(type(x))
x
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Out[38]:    Eng  Mat
Kims    87    80
Lees    79    90
Parks   80    80
```

```
In [39]: ## [Data Frame] 데이터 추출 : 원하는 열의 행 추출
collist = ['Eng', 'Mat']
x = df[collist];          # 'Eng'와 'Mat'는 List 형태로 입력
x
```

```
Out[39]:    Eng  Mat
Kims    87    80
Lees    79    90
Parks   80    80
```

> 열을 추출하여 조건식을 사용한 행 추출

```
In [2]: ## [Data Frame] 데이터 추출 : 조건부 행 검사
df['Class'] == 1 #비교 연산이기 때문에 결과는 True / False
```

```
Out[2]: Kims      True
Lees      True
Parks    False
Name: Class, dtype: bool
```

```
In [3]: ## [Data Frame] 데이터 추출 : 조건부 행 추출
df[df['Class'] == 1] #df에서 (df['Class'] == 1)가 True인 것만 추출
```

Out[3]:

| | Class | Eng | Mat |
|------|-------|-----|-----|
| Kims | 1 | 87 | 80 |
| Lees | 1 | 79 | 90 |

In [43]: ## [Data Frame] 데이터 추출 : 조건부 행 추출
x = df[df['Eng'] >= 80]; # 'Eng' 열 값들을 시리즈(Series)로 추출해서 비교
x

Out[43]:

| | Class | Eng | Mat |
|-------|-------|-----|-----|
| Kims | 1 | 87 | 80 |
| Parks | 2 | 80 | 80 |

In [44]: ## [Data Frame] 데이터 추출 : 조건부 행 추출
x = df[(df['Eng'] >= 80) & (df['Mat'] >= 80)];
x

Out[44]:

| | Class | Eng | Mat |
|-------|-------|-----|-----|
| Kims | 1 | 87 | 80 |
| Parks | 2 | 80 | 80 |

> 행 추출하여 열 선택 추출

In [53]: ## [Data Frame] 데이터 추출: 조건부 행, 열 추출
df[df['Class'] == 1]['Mat']

Out[53]: Kims ... 80
Lees ... 90
Name: Mat, dtype: int64

In [55]: ## [Data Frame] 데이터 추출: 조건부 행, 열 추출
df[df['Class'] == 1][['Mat', 'Eng']]

Out[55]:

| | Mat | Eng |
|------|-----|-----|
| Kims | 80 | 87 |
| Lees | 90 | 79 |

In [46]: ## [Data Frame] 데이터 추출: 조건부 행, 열 추출
x = df[(df['Eng'] >= 80) & (df['Mat'] >= 80)][['Eng', 'Mat']]; ### 'Eng', 'Mat' 열
x

Out[46]:

| | Eng | Mat |
|-------|-----|-----|
| Kims | 87 | 80 |
| Parks | 80 | 80 |

(알아 두면 좋아요) .을 이용해 변수를 간단하게 추출하기

> DataFrame.Col_variable로 열 추출 가능

```
In [51]: ## [Data Frame] 데이터 추출 :  
df.Mat
```

```
Out[51]: Kims     80  
Lees      90  
Parks     80  
Name: Mat, dtype: int64
```

```
In [ ]:
```

[] 데이터 추출 : loc[] 이용

> **loc[]**는 데이터프레임에서 특정 행과 열을 선택하는 인덱싱(indexing) 방법 중 하나

> **Index**는 행을 식별하는 **Key** 역할을 함.

> **Indexing은 Index를 이용해 데이터를 추출하는 방법 > 결과적으로 행을 추출**

> **loc[row_index, col_index]**로 추출

> **loc[x:y]** 숫자 **index**가 x 이상 ~ y 이하 행을 추출

```
In [4]: ## [Data Frame] 생성  
import pandas as pd  
df = pd.DataFrame({'Class' : [1, 1, 2],  
                   'Eng' : [87, 79, 80],  
                   'Mat' : [80, 90, 80]},  
                   index = ['Kims', 'Lees', 'Parks']) #index 강제 부여  
df
```

```
Out[4]:
```

| | Class | Eng | Mat |
|--------------|-------|-----|-----|
| Kims | 1 | 87 | 80 |
| Lees | 1 | 79 | 90 |
| Parks | 2 | 80 | 80 |

```
In [154...]: ## [Data Frame] 데이터 추출: loc[] > 행 (Series 자료형) 추출  
# loc[row_index, col_index]  
result = df.loc['Kims'] #결과는 Series 자료형  
print(type(result))  
result
```

```
<class 'pandas.core.series.Series'>  
Out[154]: Class     1  
Eng      87  
Mat      80  
Name: Kims, dtype: int64
```

```
In [78]: ## [Data Frame] 데이터 추출: loc[] > 행 (Data Frame 자료형) 추출  
result = df.loc[['Kims']] #결과는 Data Frame 자료형  
print(type(result))  
result
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[78]:

| | Class | Eng | Mat |
|------|-------|-----|-----|
| Kims | 1 | 87 | 80 |

In [5]:

```
## [Data Frame] 데이터 추출: loc[] > 복수 행 추출
result = df.loc[:] #전체 행 선택, 결과는 Data Frame 자료형
result
```

Out[5]:

| | Class | Eng | Mat |
|-------|-------|-----|-----|
| Kims | 1 | 87 | 80 |
| Lees | 1 | 79 | 90 |
| Parks | 2 | 80 | 80 |

In [7]:

```
## [Data Frame] 데이터 추출: loc[] > 복수 행 추출
result = df.loc[['Kims', 'Lees']] #행 선택, 결과는 Data Frame 자료형
result
```

Out[7]:

| | Class | Eng | Mat |
|------|-------|-----|-----|
| Kims | 1 | 87 | 80 |
| Lees | 1 | 79 | 90 |

In [80]:

```
## [Data Frame] 데이터 추출: loc[] > 행 (Data Frame 자료형) 추출
result = df.loc['Kims':'Lees'] #결과는 Data Frame 자료형
print(type(result))
result
```

<class 'pandas.core.frame.DataFrame'>

Out[80]:

| | Class | Eng | Mat |
|------|-------|-----|-----|
| Kims | 1 | 87 | 80 |
| Lees | 1 | 79 | 90 |

> 행 추출하여 열 선택 추출

In [81]:

```
## [Data Frame] 데이터 추출: loc[] > 복수 행, 선택 열 추출
# loc[row_index, col_index]
result = df.loc[['Kims', 'Lees'], 'Mat'] #결과는 Series 자료형
result = df.loc[['Kims', 'Lees'], ['Mat']] #결과는 Data Frame 자료형
result
```

Out[81]:

| | Mat |
|------|-----|
| Kims | 80 |
| Lees | 90 |

In [82]:

```
## [Data Frame] 데이터 추출: loc[] > 복수 행, 선택 열 추출
result = df.loc[['Kims', 'Lees'], :] #전체 행 선택, 결과는 Data Frame 자료형
result = df.loc[['Kims', 'Lees'], ['Mat', 'Eng']] #결과는 Data Frame 자료형
result
```

Out[82]:

| | Mat | Eng |
|------|-----|-----|
| Kims | 80 | 87 |
| Lees | 90 | 79 |

In [8]: ## [Data Frame] 데이터 추출: loc[] > 조건부 추출
result = df.loc[df['Eng'] >= 80] #[비교 연산식]이 True인 행 선택
result

Out[8]:

| | Class | Eng | Mat |
|-------|-------|-----|-----|
| Kims | 1 | 87 | 80 |
| Parks | 2 | 80 | 80 |

> Index를 이용한 추출

In [9]: ## [Data Frame] 생성
import pandas as pd
df = pd.DataFrame({'Name' : ['Kims', 'Lees', 'Parks'],
 'Class' : [1, 1, 2],
 'Eng' : [87, 79, 80],
 'Mat' : [80, 90, 80]}) #index 자동 부여
df

Out[9]:

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 1 | Lees | 1 | 79 | 90 |
| 2 | Parks | 2 | 80 | 80 |

In [10]: ## [Data Frame] 데이터 추출: index로 추출
df.loc[0] #0번 index 행 추출

Out[10]: Name Kims
Class 1
Eng 87
Mat 80
Name: 0, dtype: object

In [11]: ## [Data Frame] 데이터 추출: index로 추출
df.loc[[0, 2]] #0, 2번 index 행 추출

Out[11]:

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 2 | Parks | 2 | 80 | 80 |

In [12]: ## [Data Frame] 데이터 추출: index로 추출
df.loc[0:2] #0~2번 index 행 추출

Out[12]:

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 1 | Lees | 1 | 79 | 90 |
| 2 | Parks | 2 | 80 | 80 |

In [13]:

```
## [Data Frame] 데이터 추출: index로 추출  
df.loc[1:] #1에서 끝까지 index 행 추출
```

Out[13]:

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 1 | Lees | 1 | 79 | 90 |
| 2 | Parks | 2 | 80 | 80 |

> 행 추출하여 열 선택 추출

In [14]:

```
## [Data Frame] 데이터 추출: index로 추출  
df.loc[:, ['Eng', 'Mat']] #1에서 끝까지 index 행 추출
```

Out[14]:

| | Eng | Mat |
|---|-----|-----|
| 0 | 87 | 80 |
| 1 | 79 | 90 |

In [19]:

```
## [Data Frame] 생성  
import pandas as pd  
df = pd.DataFrame({'Class' : [1, 1, 2],  
                   'Eng' : [87, 79, 80],  
                   'Mat' : [80, 90, 80]},  
                   index = ['Kims', 'Lees', 'Parks']) #index 강제 부여  
#index = [1, 2, 3]) #index 강제 부여  
df
```

Out[19]:

| | Class | Eng | Mat |
|---|-------|-----|-----|
| 1 | 1 | 87 | 80 |
| 2 | 1 | 79 | 90 |
| 3 | 2 | 80 | 80 |

> 숫자 index 번호가 없을 때, index 번호로 .loc[:] 사용 불가능

In [20]:

```
## [Data Frame] 데이터 추출:  
## index 번호가 없을 때, index 번호로 .loc[:] 사용 불가능  
df.loc[0:1]
```

Out[20]:

| | Class | Eng | Mat |
|---|-------|-----|-----|
| 1 | 1 | 87 | 80 |

> 숫자 index 번호가 없을 때, index 번호로 .iloc[:] 사용 가능

In [18]:

```
## [Data Frame] 데이터 추출:  
## index 번호가 없을 때, index 번호로 .iloc[:] 사용 가능
```

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

Out[18]:

| | Class | Eng | Mat |
|------|-------|-----|-----|
| Kims | 1 | 87 | 80 |

In []:

[] 데이터 추출 : iloc[] 이용

> `iloc[]`는 데이터 추출 전에 행과 열에 임시로 숫자 `index`를 지정하여 사용하는 인덱싱 방법

> `loc[]`와 달리 조전부 추출이 불가능

> `iloc[x:y]` x 이상 ~ y 미만 행을 추출 >> `loc[x:y]` 숫자 `index`가 x 이상 ~ y 이하 행을 추출

```
# # [Data Frame] 생성
import pandas as pd
df = pd.DataFrame({ 'Name' : ['Kims', 'Lees', 'Parks'],
                     'Class' : [1, 1, 2],
                     'Eng' : [87, 79, 80],
                     'Mat' : [80, 90, 80]}) # index 자동 부여
df
```

Out[21]:

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 1 | Lees | 1 | 79 | 90 |
| 2 | Parks | 2 | 80 | 80 |

```
# # [Data Frame] 데이터 추출: iloc[] > 행 추출
result = df.iloc[0] # 1개 데이터이므로 Series 형으로 추출
result = df.iloc[[0]] # Data Frame 형으로 추출
result = df.iloc[:] # 복수개 데이터이므로 Data Frame 형으로 추출
result = df.iloc[1:3] # 복수개 데이터이므로 Data Frame 형으로 추출
result
```

Out[22]:

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 1 | Lees | 1 | 79 | 90 |
| 2 | Parks | 2 | 80 | 80 |

```
# # [Data Frame] 데이터 추출: iloc[] > 행 추출
result = df.iloc[[0, 2]] # 복수개 데이터이므로 Data Frame 형으로 추출
result
```

Out[125]:

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 2 | Parks | 2 | 80 | 80 |

Index 번호로 열 추출

```
In [126]: ## [Data Frame] 데이터 추출: iloc[] > 행, 열 추출  
result = df.iloc[[0, 2], [1]]          #열 선택도 열 순서 번호(index)로 명시  
result = df.iloc[[0, 2], [1, 0]]      #열 선택도 열 순서 번호(index)로 명시  
result
```

Out[126]: **Class Name**

| | Class | Name |
|---|-------|-------|
| 0 | 1 | Kims |
| 2 | 2 | Parks |

> iloc[]는 조건 추출이 불가능

```
In [131]: df.iloc[:, [1]]
```

Out[131]: **Class**

| | Class |
|---|-------|
| 0 | 1 |
| 1 | 1 |
| 2 | 2 |

```
In [33]: df.loc[df['Class'] == 1]
```

Out[33]: **Name Class Eng Mat**

| | Name | Class | Eng | Mat |
|---|------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 1 | Lees | 1 | 79 | 90 |

```
In [34]: df.iloc[df['Class'] == 1]
```

```

-----
--> NotImplementedError ..... Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18520\1661649294.py in <module>
----> 1 df.iloc[df['Class'] == 1]

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
    1151
    1152     .... maybe_callable = com.apply_if_callable(key, self.obj)
-> 1153         return self._getitem_axis(maybe_callable, axis=axis)
    1154
    1155     ... def _is_scalar_access(self, key: tuple):

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self, key, axis)
    1698
    1699     .... if com.is_bool_indexer(key):
-> 1700         self._validate_key(key, axis)
    1701     .... return self._getbool_axis(key, axis=axis)
    1702

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_key(self, key, axis)
    1538     .... if hasattr(key, "index") and isinstance(key.index, Index):
    1539         .... if key.index.inferred_type == "integer":
-> 1540             raise NotImplementedError(
    1541                 "iLocation based boolean "
    1542                 "indexing on an integer type"

NotImplementedError: iLocation based boolean indexing on an integer type is not available

```

In [35]: `df[df['Class'] == 1]` #df['Class'] == 1인 행 선택하여 데이터 프레임 구성

Out[35]:

| | Name | Class | Eng | Mat |
|----------|------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 1 | Lees | 1 | 79 | 90 |

In [37]: `df[df['Class'] == 1].iloc[:, [0, 2, 3]]`

Out[37]:

| | Name | Eng | Mat |
|----------|------|-----|-----|
| 0 | Kims | 87 | 80 |
| 1 | Lees | 79 | 90 |

<> loc[x:y]가 x 이상 ~ y 이하 행을 추출하는 이유

> index가 사용자가 직접 부여하므로 0부터 시작한다는 보장이 없음

> 따라서 index 범위는 쪽 짹어서 loc[부터:까지]가 된다.

In [114...]:

```

## [Data Frame] 생성
import pandas as pd
df = pd.DataFrame({'Class' : [1, 1, 2],
                   'Eng' : [87, 79, 80],

```

```
'Mat' : [80, 90, 80]},  
index = [1, 2, 3]) #index 강제 부여  
df
```

Out[114]:

| | Class | Eng | Mat |
|---|-------|-----|-----|
| 1 | 1 | 87 | 80 |
| 2 | 1 | 79 | 90 |
| 3 | 2 | 80 | 80 |

In [115...]: df.loc[1:2]

Out[115]:

| | Class | Eng | Mat |
|---|-------|-----|-----|
| 1 | 1 | 87 | 80 |
| 2 | 1 | 79 | 90 |

<> iloc[x:y]가 x 이상 ~ y 미만 행을 추출하는 이유

- > index가 0부터 자동 부여되므로 0부터 시작한다는 보장이 있음
- > 따라서 index 범위는 range() 처럼 iloc[부터:전까지]가 된다.

In []:

```
## [Data Frame] 생성  
import pandas as pd  
df = pd.DataFrame({ 'Name' : ['Kims', 'Lees', 'Parks'],  
                    'Class' : [1, 1, 2],  
                    'Eng' : [87, 79, 80],  
                    'Mat' : [80, 90, 80]}) #index 자동 부여  
df
```

In [116...]: df.iloc[1:2]

Out[116]:

| | Class | Eng | Mat |
|---|-------|-----|-----|
| 2 | 1 | 79 | 90 |

In []:

(알아 두면 좋아요) 열 변수를 index로 전환하기

- > index를 열 변수로 전환도 가능(단, index에 변수명이 있을 때만 가능)
- > **.set_index()** 메서드 사용
 - >> key 인덱스로 사용하려는 열
 - >> drop 인덱스 제거 (default는 True)
 - >> append 인덱스 추가 (default는 False)
 - >> inplace False이면 자신은 안 바꾸고 변경 결과만 반환 (default는 False)

```
In [61]: ## [Data Frame] 생성
import pandas as pd
df = pd.DataFrame({ 'Name' : ['Kims', 'Lees', 'Parks'],
                     'Class' : [1, 1, 2],
                     'Eng' : [87, 79, 80],
                     'Mat' : [80, 90, 80]} ,) #index 강제 부여
df
```

Out[61]:

| | Name | Class | Eng | Mat |
|----------|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 1 | Lees | 1 | 79 | 90 |
| 2 | Parks | 2 | 80 | 80 |

```
In [62]: # 'Name' 열을 인덱스로 설정
df.set_index('Name', inplace=True) #'Name'으로 인덱스 대체
df
```

Out[62]:

| | Class | Eng | Mat |
|--------------|-------|-----|-----|
| Name | | | |
| Kims | 1 | 87 | 80 |
| Lees | 1 | 79 | 90 |
| Parks | 2 | 80 | 80 |

.reset_index()

> drop True이면 index 제거 (default는 True)

> inplace False이면 자신은 안 바꾸고 변경 결과만 반환 (default는 False)

```
In [66]: # 현재 인덱스를 일반 열로 변경
df.reset_index(inplace=True) #inplace=True는 자신을 바꿈 (default는 False)
df
```

Out[66]:

| | Name | Class | Eng | Mat |
|----------|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 1 | Lees | 1 | 79 | 90 |
| 2 | Parks | 2 | 80 | 80 |

```
In [67]: # 현재 인덱스를 일반 열로 변경
ddf = df.reset_index(inplace=False, drop=False) #inplace=False는 자신은 안 바꾸고
ddf
```

Out[67]:

| | index | Name | Class | Eng | Mat |
|----------|-------|-------|-------|-----|-----|
| 0 | 0 | Kims | 1 | 87 | 80 |
| 1 | 1 | Lees | 1 | 79 | 90 |
| 2 | 2 | Parks | 2 | 80 | 80 |

```
In [70]: ## inplace=False는 자신은 안 바꾸고 변경 결과만 반환
df
```

```
Out[70]:
```

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 1 | Lees | 1 | 79 | 90 |
| 2 | Parks | 2 | 80 | 80 |

(알아두면 좋아요) 리스트, 시리즈, 데이터 프레임 복사

- > 얕은 복사(shallow copy)와 깊은 복사(deep copy)가 있다.
- > 얕은 복사는 복사 대상을 공유하는 형태로 참조 방식
- > 깊은 복사는 복사 대상과 똑같은 데이터를 생성하여 항목을 복사하는 방식

```
In [73]:
```

```
## [Data Frame] 생성
import pandas as pd
df = pd.DataFrame({'Name' : ['Kims', 'Lees', 'Parks'],
                   'Class' : [1, 1, 2],
                   'Eng' : [87, 79, 80],
                   'Mat' : [80, 90, 80]}) #index 자동 부여
df
```

```
Out[73]:
```

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 80 |
| 1 | Lees | 1 | 79 | 90 |
| 2 | Parks | 2 | 80 | 80 |

```
In [74]:
```

```
## 얕은 복사(shallow copy)
df_new = df          # df_new가 df를 참조하는 형태로 복사
df['Mat'] = 100
df_new
```

```
Out[74]:
```

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 100 |
| 1 | Lees | 1 | 79 | 100 |
| 2 | Parks | 2 | 80 | 100 |

```
In [75]:
```

```
## 깊은 복사(deep copy)
df_new = df[:]        # df를 슬라이싱 해서 df_new로 복사하므로 두 프레임은 별개
df['Mat'] = 0
df_new
```

```
Out[75]:
```

| | Name | Class | Eng | Mat |
|---|-------|-------|-----|-----|
| 0 | Kims | 1 | 87 | 100 |
| 1 | Lees | 1 | 79 | 100 |
| 2 | Parks | 2 | 80 | 100 |

```
In [76]: ## 깊은 복사(deep copy)
df_new = df.copy()
df_new['Mat'] = 90
df_new
```

```
Out[76]:   Name  Class  Eng  Mat
0    Kims      1    87    90
1    Lees      1    79    90
2   Parks      2    80    90
```

In []:

[] 데이터 프레임 데이터 조작

```
In [77]: ## [Data Frame] 생성
import pandas as pd
df = pd.DataFrame({'Name' : ['Kims', 'Lees', 'Parks'],
                   'Class' : [1, 1, 2],
                   'Eng' : [87, 79, 80],
                   'Mat' : [80, 90, 80]},) #index 강제 부여
df
```

```
Out[77]:   Name  Class  Eng  Mat
0    Kims      1    87    80
1    Lees      1    79    90
2   Parks      2    80    80
```

```
In [78]: ## [Data Frame] 조작 : 열 추가
df['avg'] = (df['Eng'] + df['Mat']) / 2 # 'avg' 열을 만들어서 평균 값 추가
df
```

```
Out[78]:   Name  Class  Eng  Mat  avg
0    Kims      1    87    80  83.5
1    Lees      1    79    90  84.5
2   Parks      2    80    80  80.0
```

```
In [79]: ## [Data Frame] 조작 : 열 순서 변경
df = df[['Class', 'Name', 'Eng', 'Mat', 'avg']] # 원하는 컬럼 순으로 재정렬
df
```

```
Out[79]:   Class  Name  Eng  Mat  avg
0      1   Kims   87   80  83.5
1      1   Lees   79   90  84.5
2      2   Parks  80   80  80.0
```

```
In [80]: ## [Data Frame] 조작 : 열 순서 변경
col_seq = ['Class', 'Name', 'Eng', 'Mat', 'avg']
```

```
df = df[col_seq] # 원하는 컬럼 순으로 재정렬  
df
```

Out[80]:

| | Class | Name | Eng | Mat | avg |
|---|-------|-------|-----|-----|------|
| 0 | 1 | Kims | 87 | 80 | 83.5 |
| 1 | 1 | Lees | 79 | 90 | 84.5 |
| 2 | 2 | Parks | 80 | 80 | 80.0 |

In []:

[] 데이터프레임 개별 항목 값 조작

> 반복문을 사용하여 개별 조작

In [81]:

```
## [Data Frame] 조작 : 반복문 사용 접근  
for index, row in df.iterrows():  
    print(index, df.at[index, 'Mat']) # .at[]은 행과 열로 특정값 검색
```

```
0 80  
1 90  
2 80
```

In [82]:

```
## [Data Frame] 조작 : 반복문 사용 조작  
for index, row in df.iterrows():  
    if df.at[index, 'Mat'] <= 90:  
        df.at[index, 'Mat'] = df.at[index, 'Mat'] + 10  
df
```

Out[82]:

| | Class | Name | Eng | Mat | avg |
|---|-------|-------|-----|-----|------|
| 0 | 1 | Kims | 87 | 90 | 83.5 |
| 1 | 1 | Lees | 79 | 100 | 84.5 |
| 2 | 2 | Parks | 80 | 90 | 80.0 |

In [83]:

```
## [Data Frame] 조작 : 열 값 일괄 적용 조작  
df['Mat'] = df['Mat'] - 10  
df
```

Out[83]:

| | Class | Name | Eng | Mat | avg |
|---|-------|-------|-----|-----|------|
| 0 | 1 | Kims | 87 | 80 | 83.5 |
| 1 | 1 | Lees | 79 | 90 | 84.5 |
| 2 | 2 | Parks | 80 | 80 | 80.0 |

In [84]:

```
alist = [1, 2, 3, 4, 5]  
sum(alist)
```

Out[84]:

```
15
```

[] 데이터 프레임 관련 함수 사용

```
In [85]: ## [Data Frame] 관련 함수 : 통계 보기  
sum(df['Eng'])
```

```
Out[85]: 246
```

```
In [87]: ## [Data Frame] 관련 함수 : 통계 보기  
len(df) #행의 개수
```

```
Out[87]: 3
```

```
In [88]: ## [Data Frame] 관련 함수 : 통계 보기  
avg_mat = sum(df['Mat']) / len(df)  
print('Average of Math : ', avg_mat)
```

```
Average of Math : 83.33333333333333
```

```
In [89]: ## [Data Frame] 관련 함수 사용  
x = df['Mat']; # 'Mat' 열 값들을 시리즈로 추출  
x.mean() # 'Mat' 열 값들의 평균 구하기
```

```
Out[89]: 83.33333333333333
```

```
In [90]: ## [Data Frame] 관련 함수 사용  
x = df['Mat']; # 'Mat' 열 값들을 시리즈로 추출  
x.value_counts() # 'Mat' 열 값들의 개수 구하기
```

```
Out[90]:  
Mat  
80    2  
90    1  
Name: count, dtype: int64
```

데이터 정렬

> `.sort_values(by = 'avg')`는 자신은 그대로 두고 정렬 결과를 반환

```
In [91]: ## [Data Frame] 관련 함수 : sorting된 Data Frame 생성  
sdf = df.sort_values(by = 'avg') # 'avg' 컬럼 값 순으로 정렬  
sdf
```

```
Out[91]:  


|   | Class | Name  | Eng | Mat | avg  |
|---|-------|-------|-----|-----|------|
| 2 | 2     | Parks | 80  | 80  | 80.0 |
| 0 | 1     | Kims  | 87  | 80  | 83.5 |
| 1 | 1     | Lees  | 79  | 90  | 84.5 |


```

```
In [93]: ## .sort_values(by = 'avg')는 자신은 그대로 두고 정렬 결과를 반환  
df
```

```
Out[93]:  


|   | Class | Name  | Eng | Mat | avg  |
|---|-------|-------|-----|-----|------|
| 0 | 1     | Kims  | 87  | 80  | 83.5 |
| 1 | 1     | Lees  | 79  | 90  | 84.5 |
| 2 | 2     | Parks | 80  | 80  | 80.0 |


```

```
In [94]: ## [Data Frame] 관련 함수 : sorting된 Data Frame 생성  
sdf = df.sort_values(by = 'avg', ascending=False)  
sdf
```

Out[94]:

| | Class | Name | Eng | Mat | avg |
|---|-------|-------|-----|-----|------|
| 1 | 1 | Lees | 79 | 90 | 84.5 |
| 0 | 1 | Kims | 87 | 80 | 83.5 |
| 2 | 2 | Parks | 80 | 80 | 80.0 |

컬럼명 변경

> .rename()

In [95]: ## [Data Frame] 관련 함수 : 컬럼명 변경

```
df = df.rename(columns = {'Mat' : 'Math'}) # 컬럼명 변경
```

Out[95]:

| | Class | Name | Eng | Math | avg |
|---|-------|-------|-----|------|------|
| 0 | 1 | Kims | 87 | 80 | 83.5 |
| 1 | 1 | Lees | 79 | 90 | 84.5 |
| 2 | 2 | Parks | 80 | 80 | 80.0 |

행, 열 제거

> .drop()

In [96]:

[Data Frame] 관련 함수 : 행, 열 제거

```
ddf = df.drop( 'avg', axis = 'columns') # 컬럼 이름으로 컬럼 제거
```

Out[96]:

| | Class | Name | Eng | Math |
|---|-------|-------|-----|------|
| 0 | 1 | Kims | 87 | 80 |
| 1 | 1 | Lees | 79 | 90 |
| 2 | 2 | Parks | 80 | 80 |

In [97]:

[Data Frame] 관련 함수 : 행, 열 제거

```
ddf = df.drop( 1, axis = 'rows') # 행 이름으로 열 제거
```

```
ddf
```

Out[97]:

| | Class | Name | Eng | Math | avg |
|---|-------|-------|-----|------|------|
| 0 | 1 | Kims | 87 | 80 | 83.5 |
| 2 | 2 | Parks | 80 | 80 | 80.0 |

행 일부만 출력

> .head()

> .tail()

In [98]:

[Data Frame] 관련 함수 : 일부만 출력

```
df.head(2) # 데이터 행 상위 일부만 출력 (숫자 생략하면 알아서)
```

```
Out[98]:   Class Name Eng Math avg
          0     1   Kims   87    80  83.5
          1     1   Lees   79    90  84.5
```

```
In [99]: ## [Data Frame] 관련 함수 : 일부만 출력
           df.tail(1)           # 데이터 행 하위 일부만 출력 (숫자 생략하면 알아서)
```

```
Out[99]:   Class Name Eng Math avg
          2     2   Parks   80    80  80.0
```

행과 열의 수 출력

```
> .shape()
```

```
In [183...]: ## [Data Frame] 관련 함수 : 행, 열의 수 출력
              df.shape
```

```
Out[183]: (3, 5)
```

프레임 속성 출력

```
> .info()
```

```
In [184...]: ## [Data Frame] 관련 함수 : 데이터 프레임 속성 출력
              df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 5 columns):
 # ... Column ... Non-Null Count ... Dtype ...
--- ... --- ... ... ...
 0 ... Class ... 3 non-null ... int64 ...
 1 ... Name ... 3 non-null ... object ...
 2 ... Eng ... 3 non-null ... int64 ...
 3 ... Math ... 3 non-null ... int64 ...
 4 ... avg ... 3 non-null ... float64 ...
dtypes: float64(1), int64(3), object(1)
memory usage: 248.0+ bytes
```

프레임 통계 출력

```
> .describe()
```

```
In [185...]: ## [Data Frame] 관련 함수 : 통계 보기
              df.describe()
```

Out[185]:

| | Class | Eng | Math | avg |
|--------------|----------|-----------|-----------|-----------|
| count | 3.000000 | 3.000000 | 3.000000 | 3.000000 |
| mean | 1.333333 | 82.000000 | 83.333333 | 82.666667 |
| std | 0.577350 | 4.358899 | 5.773503 | 2.362908 |
| min | 1.000000 | 79.000000 | 80.000000 | 80.000000 |
| 25% | 1.000000 | 79.500000 | 80.000000 | 81.750000 |
| 50% | 1.000000 | 80.000000 | 80.000000 | 83.500000 |
| 75% | 1.500000 | 83.500000 | 85.000000 | 84.000000 |
| max | 2.000000 | 87.000000 | 90.000000 | 84.500000 |

In []:

□ Excel이나 csv 타입 파일로부터 데이터 프레임 구성하기

> import pandas 를 통해 read_excel() 또는 read_csv()를 사용한다.

>> 데이터에 한글이나 특수 문자가 포함되어있는경우 encoding='UTF-8' 적용

In [100...]

```
## [Data Frame] Excel 파일 >> 데이터 프레임 구축
import pandas as pd
df_exam = pd.read_excel('excel_exam.xlsx', header=None)      #header가 없을 때, 데이터
df_exam = pd.read_excel('excel_exam.xlsx')                   #header가 있을 때, 첫 행이
df_exam
```

Out[100]:

| | id | nclass | math | english | science |
|-----------|-----------|---------------|-------------|----------------|----------------|
| 0 | 1 | 1 | 50 | 98 | 50 |
| 1 | 2 | 1 | 60 | 97 | 60 |
| 2 | 3 | 1 | 45 | 86 | 78 |
| 3 | 4 | 1 | 30 | 98 | 58 |
| 4 | 5 | 2 | 25 | 80 | 65 |
| 5 | 6 | 2 | 50 | 89 | 98 |
| 6 | 7 | 2 | 80 | 90 | 45 |
| 7 | 8 | 2 | 90 | 78 | 25 |
| 8 | 9 | 3 | 20 | 98 | 15 |
| 9 | 10 | 3 | 50 | 98 | 45 |
| 10 | 11 | 3 | 65 | 65 | 65 |
| 11 | 12 | 3 | 45 | 85 | 32 |
| 12 | 13 | 4 | 46 | 98 | 65 |
| 13 | 14 | 4 | 48 | 87 | 12 |
| 14 | 15 | 4 | 75 | 56 | 78 |
| 15 | 16 | 4 | 58 | 98 | 65 |
| 16 | 17 | 5 | 65 | 68 | 98 |
| 17 | 18 | 5 | 80 | 78 | 90 |
| 18 | 19 | 5 | 89 | 68 | 87 |
| 19 | 20 | 5 | 78 | 83 | 58 |

In [187...]

```
## [Data Frame] Excel 파일 >> 데이터 프레임 구축
import pandas as pd
df_exam = pd.read_excel('excel_exam.xlsx', sheet_name='Sheet2', header=None)    # header가 없을 때, 데이터 프레임에 열 index가 자동 부여됨
df_exam
```

Out[187]:

| | 0 | 1 | 2 | 3 | 4 |
|----|----|---|----|----|----|
| 0 | 1 | 1 | 50 | 98 | 50 |
| 1 | 2 | 1 | 60 | 97 | 60 |
| 2 | 3 | 1 | 45 | 86 | 78 |
| 3 | 4 | 1 | 30 | 98 | 58 |
| 4 | 5 | 2 | 25 | 80 | 65 |
| 5 | 6 | 2 | 50 | 89 | 98 |
| 6 | 7 | 2 | 80 | 90 | 45 |
| 7 | 8 | 2 | 90 | 78 | 25 |
| 8 | 9 | 3 | 20 | 98 | 15 |
| 9 | 10 | 3 | 50 | 98 | 45 |
| 10 | 11 | 3 | 65 | 65 | 65 |
| 11 | 12 | 3 | 45 | 85 | 32 |
| 12 | 13 | 4 | 46 | 98 | 65 |
| 13 | 14 | 4 | 48 | 87 | 12 |
| 14 | 15 | 4 | 75 | 56 | 78 |
| 15 | 16 | 4 | 58 | 98 | 65 |
| 16 | 17 | 5 | 65 | 68 | 98 |
| 17 | 18 | 5 | 80 | 78 | 90 |
| 18 | 19 | 5 | 89 | 68 | 87 |
| 19 | 20 | 5 | 78 | 83 | 58 |

In [101...]

```
## [Data Frame] 데이터 프레임 >> csv 파일로 저장
import pandas as pd
df_exam = pd.read_excel('excel_exam.xlsx')      #header가 있을 때, 첫 행이 데이터 프레임의 헤더로 사용되는 경우
df_exam.to_csv("exam_csv.csv")
df_exam.to_csv("exam_csv.csv", index=False) #index는 빼고 저장

df_exam = pd.read_csv('exam_csv.csv')    #header가 있을 때, 첫 행이 데이터 프레임의 헤더로 사용되는 경우
df_exam
```

Out[101]:

| | id | nclass | math | english | science |
|-----------|-----------|---------------|-------------|----------------|----------------|
| 0 | 1 | 1 | 50 | 98 | 50 |
| 1 | 2 | 1 | 60 | 97 | 60 |
| 2 | 3 | 1 | 45 | 86 | 78 |
| 3 | 4 | 1 | 30 | 98 | 58 |
| 4 | 5 | 2 | 25 | 80 | 65 |
| 5 | 6 | 2 | 50 | 89 | 98 |
| 6 | 7 | 2 | 80 | 90 | 45 |
| 7 | 8 | 2 | 90 | 78 | 25 |
| 8 | 9 | 3 | 20 | 98 | 15 |
| 9 | 10 | 3 | 50 | 98 | 45 |
| 10 | 11 | 3 | 65 | 65 | 65 |
| 11 | 12 | 3 | 45 | 85 | 32 |
| 12 | 13 | 4 | 46 | 98 | 65 |
| 13 | 14 | 4 | 48 | 87 | 12 |
| 14 | 15 | 4 | 75 | 56 | 78 |
| 15 | 16 | 4 | 58 | 98 | 65 |
| 16 | 17 | 5 | 65 | 68 | 98 |
| 17 | 18 | 5 | 80 | 78 | 90 |
| 18 | 19 | 5 | 89 | 68 | 87 |
| 19 | 20 | 5 | 78 | 83 | 58 |

In [189...]

```
def int2han(remain):
    how = [1000, 100, 10, 1]
    how_han = ['천', '백', '십', '']
    digit_han = ['', '일', '이', '삼', '사', '오', '육', '칠', '팔', '구']
    result = ''
    for i in range(len(how)):
        out = remain // how[i]
        if out != 0:
            result += digit_han[out] + how_han[i]
        remain = remain % how[i]
    return result

inn = 123
result = int2han(inn)
print(result)
```

일백이십삼

In []:

[실습-1] 데이터 프레임 조작하기

1. Excel > Data Frame 변환

> 'excel_exam.xlsx' 파일을 읽어서 데이터 프레임 구성

2. 'total'과 'avg' 열 변수 추가

> 'total'은 'math', 'english','science'의 합

> 'avg'는 'math', 'english','science'의 평균

3. 'pass' 열 변수를 추가

> 'pass'를 'None' 값으로 초기화

4. 'pass' 값을 변경

> 'pass'는 'avg' 값이 60 이상이면 'P', 아니면 'F'로 값 변경

5. 열과 행이 선택된 'df_pass' 데이터 프레임 생성

> 'pass'가 'P' 행들만 선택

> 선택된 행 중에 'id', 'nclass', 'math', 'english', 'science', 'avg' 열만 추출해서 구성

> 열 순서를 'nclass', 'id', 'math', 'english', 'science', 'avg' 순으로 구성

6. Data Frame > csv 파일로 변환

> 파일로 변환하기 전에 'df_pass' 데이터 프레임을 'avg' 내림차 순으로 정렬

> 데이터 프레임을 'excel_exam_pass.csv' 파일로 저장 (단, index는 빼고)

7. Data Frame > csv 파일로 변환

> 'excel_exam_pass.csv' 파일을 데이터 프레임으로 읽어서, 출력 확인

In []:

정리하기

In [3]: import pandas as pd

```
# 1. 데이터 프레임 만들기
df = pd.DataFrame({'name' : ['김지훈', '이유진', '박동현', '김민지'],
                    'english' : [90, 80, 60, 70],
                    'math' : [50, 60, 100, 20]})
```

```
df_midterm = pd.DataFrame({'english' : [90, 80, 60, 70],
                            'math' : [50, 60, 100, 20],
                            'nclass' : [1, 1, 2, 2]})
```

```
# 2. 외부 데이터 이용하기
```

```
# 엑셀 파일 불러오기
```

```
df_exam = pd.read_excel('excel_exam.xlsx')

# CSV 파일 불러오기
df_csv_exam = pd.read_csv('exam.csv')

# CSV 파일로 저장하기
df_midterm.to_csv('output_newdata.csv')
```

In []: