판다스 입문

시리즈는 데이터가 순차적으로 배열된 1차원 배열의 형태를 갖는다.

이처럼 시리즈의 인덱스는 데이터 값의 위치를 나타내는 이름표 역할을 한다.

시리즈

- **시리즈 만들기**

판다스 내장 함수인 Series()를 이용하고 딕셔너리를 함수의 인자로 전달한다

```
# pandas 불러오기
import pandas as pd

# key:value 쌍으로 딕셔너리를 만들고, 변수 dict_data에 저장
dict_data = {'a' : 1, 'b' : 2, 'c' : 3}

# 판다스 Series() 함수로 dictionary를 Series로 변환. 변수 sr에 저장
sr = pd.Series(dict_data)

# sr의 자료형 출력
print(type(sr))
print('\n')
# 변수 sr에 저장되어 있는 시리즈 객체를 출력
print(sr)
```

- **인덱스 구조**

인덱스는 자기와 짝을 이루는 데이터 값의 순서와 주소를 저장한다.

인덱스에는 크게 두 가지 종류가 있다. **정수형 위치 인덱스**와 **인덱스 이름**이 그것이다.

시리즈 클래스의 index 속성을 이용하여 인덱스 배열을 따로 선택할 수 있다.

```
# 해당 시리즈의 인덱스가 모두 나열됨
인덱스 배열: Series객체.index
```

또한 데이터 값 배열만을 따로 선택할 수도 있다. 시리즈 클래스의 values 속성을 이용한다.

```
# 해당 시리즈의 values가 모두 나열됨
데이터 값 배열: Series객체.values
```

판다스 Series() 함수를 사용하여 파이썬 리스트를 시리즈로 변환한다. 단, 리스트를 시리즈로 변환할 때는 딕셔너리의 키처럼 인덱스로 변환될 값이 없다. 따라서 인덱스를 별도로 정의하지 않으면, 디폴트로 정수형 위치 인덱스가 자동으로 지정된다.

```
import pandas as pd

# 리스트를 시리즈로 변환하여 변수 sr에 저장
list_data = ['2019-01-02', 3.14, 'ABC', 100, True]
sr = pd.Series(list_data)
print(sr)

# 인덱스 배열은 변수 idx에 저장. 데이터 값 배열은 변수 val에 저장

idx = sr.index
val = sr.values
print(idx)
print('\n')
print(val)
```

- **원소 선택**

원소의 위치를 나타내는 주소 역할을 하는 인덱스를 이용하여 시리즈의 원소를 선택한다. 하나의 원소를 선택할 수도 있고, 여러 원소를 한꺼번에 선택할 수도 있다.

정수형 위치 인덱스는 대괄호[] 안에 위치를 나타내는 숫자를 입력하는데 반해, 인덱스 이름(라벨)을 사용할 때는 대괄호[] 안에 이름과 함께 따옴표를 입력한다.

Series() 함수를 사용하여 파이썬 tuple을 시리즈로 변환한다. tuple도 리스트처럼 딕셔너리의 키에 해당하는 값이 없어서 시리즈로 변환할 때 정수형 위치 인덱스가 자동 지정된다.

리스트 또는 tuple을 시리즈로 만들 때 정수형 위치 인덱스 대신 인덱스 이름을 따로 지정할 수 있다. Series() 함수의 index 옵션에 인덱스 이름을 직접 전달하는 방식이다.

```
import pandas as pd

# 튜플을 시리즈로 변환(인덱스 옵션 지정)
tup_data = ('영인', '2010-05-01', '여', True)
sr = pd.Series(tup_data, index=['이름', '생년월일', '성별', '학생여부']) # 인덱스 옵션은 대괄호에 집어넣기
print(sr)

# 원소를 1개 선택
print(sr[0])
print(sr['이름'])

# 여러 개의 원소를 선택(인덱스 리스트 활용)
print(sr[[1,2]])
print('\n')
print(sr[['생년월일', '성별']])

# 여러 개의 원소를 선택(인덱스 범위 지정)
print(sr[1:2])
print('\n')
print(sr['생년월일':'성별'])
```

앞에서 설명한 대로 인덱스를 이용하여 원소를 선택할 때는 대괄호[] 안에 인덱스를 입력한다. 예제에서는 시리즈의 첫 번째 데이터를 선택하기 위해 첫 번째를 뜻하는 정수형 위치 인덱스(0)와 첫 번째 위치에 있는 인덱스 라벨('이름')을 입력한다. 모두 같은 원소(영인)를 반환한다.

여러 개의 인덱스를 리스트 형태로 대괄호[] 안에 입력하면 짝을 이루는 원소 데이터를 모두 반환한다.

마지막으로 인덱스 범위를 지정하여 선택하는 방법이다. 정수형 위치 인덱스를 사용할 때는 범위의 끝이 포함되지 않는다. 그러나 인덱스 이름을 사용하면 범위의 끝이 포함된다.

데이터프레임

시리즈를 열벡터라고 하면, 데이터프레임은 여러 개의 열벡터들이 같은 행 인덱스를 기준으로 줄지어 결합된 2차원 벡터 또는 행렬이다.

- **데이터프레임 만들기**

데이터프레임은 여러 개의 시리즈를 모아 놓은 집합으로 이해하면 된다. 딕셔너리의 값에 해당하는 각 리스트가 시리즈 배열로 변환되어 데이터프레임의 열이 된다. 딕셔너리의 키는 각 시리즈의 이름으로 변환되어 최종적으로 데이터프레임의 열 이름이 된다.

데이터프레임을 만들 때는 판다스 DataFrame() 함수를 사용한다. 여러 개의 리스트를 원소로 갖는 딕셔너리를 함수의 인자로 전달하는 방식이 주로 활용된다.

```
딕셔너리 -> 데이터프레임 변환: pandas.DataFrame(딕셔너리 객체)
```

리스트 5개를 원소로 갖는 딕셔너리를 정의하고, 판다스 DataFrame() 함수의 인자로 전달하여 모두 5개의 열을 갖는 데이터프레임을 만든다. 이때 딕셔너리의 키가 열 이름이 되고 값에 해당하는 리스트가 데이터프레임의 열이 된다.

```
import pandas as pd

# 열이름을 key로 하고, 리스트를 value로 갖는 딕셔너리 정의(2차원 배열)
dict_data = {'a' : [1,2,3], 'b' : [4,5,6], 'c' : [7,8,9], 'd' : [10,11,12], 'e' : [13,14,15]}

# 판다스 DataFrame() 함수로 딕셔너리를 데이터프레임으로 변환. 변수 df에 저장
df = pd.DataFrame(dict_data)

# df의 자료형 출력
print(type(df))
```

```
print('\n')

# 변수 df에 저장되어 있는 데이터프레임 객체를 출력
print(df)
```

- **행 인덱스/열 이름 설정**

2차원 배열을 DataFrame() 함수 인자로 전달하여 데이터프레임으로 변환할 때 행 인덱스와 열 이름 속성을 사용자가 직접 지정할 수도 있다.

```
행 인덱스/열 이름 설정: pandas.DataFrame(2차원 배열,
                                        index=행 인덱스 배열,
                                        columns=열 이름 배열)
```

```
import pandas as pd

# 행 인덱스/열 이름 지정하여 데이터프레임 만들기
df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']], # DataFrame에 리스트를 원소로 집어넣으면, index-wise로 들어감!
                  index= ['준서', '예은'], # 추가하고자 하는 리스트의 개수 # 시리즈에는 index이름만 지정가능하고 데이터프레임에서는 index & columns 지정 가능!
                  columns= ['나이', '성별', '학교']) # 추가되는 리스트 내부의 원소 개수

# 행 인덱스, 열 이름 확인하기
print(df)               # 데이터프레임
print('\n')
print(df.index)         # 행 인덱스
print('\n')
print(df.columns)       # 열 이름
```

실행 결과에서 리스트가 행으로 변환되는 점에 유의한다. 앞에서 리스트를 원소로 갖는 딕셔너리를 이용했을 때는 리스트가 열이 된 것과 차이가 있다.

데이터프레임 df의 행 인덱스와 열 이름 객체를 나타내는 df.index와 df.columns의 속성에 새로운 배열을 할당하는 방식으로 행 인덱스와 열 이름을 변경할 수 있다.

```
행 인덱스 변경: DataFrame객체.index = 새로운 행 인덱스 배열
열 이름 변경: DataFrame객체.columns = 새로운 열 이름 배열
```

```
# 행 인덱스, 열 이름 변경하기
df.index=['학생1', '학생2']
df.columns=['연령', '남녀', '소속']

print(df)               # 데이터프레임
print('\n')
print(df.index)         # 행 인덱스
print('\n')
print(df.columns)       # 열 이름
```

데이터프레임에 rename() 메소드를 적용하면 행 인덱스 또는 열 이름의 일부를 선택하여 변경할 수 있다. **단, 원본 객체를 직접 수정하는 것이 아니라 새로운 데이터프레임 객체를 반환하는 점에 유의한다. 원본 객체를 변경하려면 inplae=True 옵션을 사용한다.**

```
행 인덱스 변경: DataFrame 객체.rename(index={기존 인덱스:새 인덱스,...})
열 이름 변경: DataFrame 객체.rename(columns=({기존 이름:새 이름,...})
```

```
import pandas as pd

# 행 인덱스/열 이름 지정하여 데이터프레임 만들기
df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
                  index= ['준서', '예은'], # 추가하고자 하는 리스트의 개수
                  columns= ['나이', '성별', '학교']) # 추가되는 리스트 내부의 원소 개수

# 데이터프레임 df 출력
print(df)               # 데이터프레임
print('\n')

# 열 이름 중, '나이'를  '연령'으로,  '성별'을 '남녀'로 '학교'를 '소속'으로 바꾸기
```

```
df.rename(columns={'나이':'연령', '성별':'남녀', '학교':'소속'}, inplace=True)

# df의 행 인덱스 중에서, '준서'를 '학생1'로, '예은'을 '학생2'로 바꾸기
df.rename(index={'준서':'학생1', '예은':'학생2'}, inplace=True)

# df 출력(변경 후)
print(df)
```

- **행/열 삭제**

행을 삭제할 때는 축 옵션으로 axis=0을 입력하거나, 별도로 입력하지 않는다. 반면, 축 옵션으로 axis=1을 입력하면 열을 삭제한다. 동시에 여러 개의 행 또는 열을 삭제하려면 리스트 형태로 입력한다.

한편 drop() 메소드는 기존 객체를 변경하지 않고 새로운 객체를 반환하는 점에 유의한다. 따라서 원본 객체를 직접 변경하기 위해서는 inplace=True 옵션을 추가한다.

```
행 삭제: DataFrame 객체.drop(행 인덱스 또는 배열, axis=0)
열 삭제: DataFrame 객체.drop(열 이름 또는 배열, axis=1)
```

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'수학' : [90, 80, 70], '영어' : [98, 89, 95], '음악' : [85, 95, 100], '체육' : [100, 90, 90]}

df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
print(df)
print('\n')

# 데이터프레임 df를 복제하여 변수 df2에 저장. df2의 1개 행 삭제
df2 = df[:]
df2.drop('우현', axis=0, inplace=True)
print(df2)
print('\n')

# 데이터프레임 df를 복제하여 변수 df3에 저장. df3의 2개 행 삭제
df3 = df[:]
df3.drop(['우현', '인아'], axis=0, inplace=True)
print(df3)
```

inplace=True 옵션을 사용하는 대신, `df3=df3.drop(['우현', '인아'])` 라고 입력해도 결과는 같다.

다음은 drop() 메소드를 이용하여 열을 삭제하는 방법이다. 반드시 축 옵션을 axis=1로 설정한다. 축 옵션을 누락하거나 잘못 지정하면 오류 메시지가 출력된다.

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'수학' : [90, 80, 70], '영어' : [98, 89, 95], '음악' : [85, 95, 100], '체육' : [100, 90, 90]}

df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
print(df)
print('\n')

# 데이터프레임 df를 복제하여 변수 df4에 저장. df4의 1개 열 삭제
df4 = df[:]
df4.drop('수학', axis=1, inplace=True)
print(df4)
print('\n')

# 데이터프레임 df를 복제하여 변수 df5에 저장. df5의 2개 열 삭제
df5 = df[:]
df5.drop(['영어', '음악'], axis=1, inplace=True)
print(df5)
```

- **행 선택**

데이터프레임의 행 데이터를 선택하기 위해서는 loc과 iloc 인덱서를 사용한다. 인덱스 이름을 기준으로 행을 선택할 때는 loc을 이용하고, 정수형 위치 인덱스를 사용할 때는 iloc을 이용한다.

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'수학' : [90, 80, 70], '영어' : [98, 89, 95], '음악' : [85, 95, 100], '체육' : [100, 90, 90]}

df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
print(df)
print('\n')

# 행 인덱스를 사용하여 행 1개 선택
label1 = df.loc['서준']
position1 = df.iloc[0]
print(label1)
print('\n')
print(position1)
```

2개 이상의 행 인덱스를 리스트 형태로 입력하면 매칭되는 모든 행 데이터를 동시에 추출한다.

```
# 행 인덱스를 사용하여 2개 이상의 행 선택
label2 = df.loc[['서준', '우현']]
position2 = df.iloc[[0, 1]]
print(label2)
print('\n')
print(position2)
```

마지막으로 행 인덱스의 범위를 지정하여 여러 개의 행을 동시에 선택하는 슬라이싱 기법을 살펴보자. 이번에는 label3와 position3의 결과값에 차이가 있음에 유의한다. label3의 경우에는 '우현' 학생의 점수가 포함되지만, 정수형 위치 인덱스를 사용한 position3의 마지막 값인 '우현' 학생의 점수는 제외된다.

```
# 행 인덱스의 범위를 지정하여 행 선택
label3 = df.loc['서준':'우현']
position3 = df.iloc[0:1]
print(label3)
print('\n')
print(position3)
```

- **열 선택**

데이터프레임의 열 데이터를 1개만 선택할 때는, 대괄호[] 안에 열 이름을 따옴표와 함께 입력하거나 도트(.)다음에 열 이름을 입력하는 두 가지 방식을 사용한다.

```
열 1개 선택(시리즈 생성): DataFrame 객체['열 이름'] 또는 DataFrame 객체.열 이름
```

대괄호 안에 열 이름의 리스트를 입력하면 리스트의 원소인 열을 모두 선택하여 데이터프레임으로 반환한다. 또한 리스트의 원소로 열 이름 한 개만 있는 경우에도, 2중 대괄호([[]])를 사용하는 것이 되어 반환되는 객체는 **시리즈가 아니라 데이터프레임이 된다.**

```
열 n개 선택(데이터프레임 생성): DataFrame 객체[[열1, 열2, 열3,..., 열n]]
```

먼저 데이터프레임에서 열 1개를 선택하는 방법을 살펴보자.

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'이름' : ['서준', '우현', '인아'],'수학' : [90, 80, 70], '영어' : [98, 89, 95], '음악' : [85, 95, 100], '체육' : [100, 90, 90]}
df = pd.DataFrame(exam_data)
print(df)
print(type(df))
print('\n')

# '수학' 점수 데이터만 선택. 변수 math1에 저장
math1 = df['수학']
print(math1)
print(type(math1))
print('\n')

# '영어' 점수 데이터만 선택. 변수 english에 저장
english = df['영어']
```

```
print(english)
print(type(english))
```

데이터프레임에서 2개 이상의 열을 추출하는 방법이다. 대괄호 안에 ['음악', '체육'] 점수를 열 이름의 리스트로 입력하면 데이터프레임이 반환된다. **또한 df[['수학']]과 같이 2중 대괄호를 사용하면 열 이름 1개를 원소로 갖는 리스트를 사용하는 경우에도 시리즈가 아닌 데이터프레임을 반환한다.**

```
# '음악', '체육' 점수 데이터를 선택. 변수 music_gym에 저장
music_gym = df[['음악', '체육']]
print(music_gym)
print(type(music_gym))
print('\n')

# '수학' 점수 데이터만 선택. 변수 math2에 저장
math2 = df[['수학']]
print(math2)
print(type(math2))
```

💡 **범위 슬라이싱의 고급활용**
판다스 데이터프레임의 원소 데이터를 선택할 때, 범위를 지정하여 슬라이싱하는 방법을 여러가지로 응용할 수 있다. iloc인덱서로 예를 들어보자

```
범위 슬라이싱: DataFrame 객체.iloc[시작 인덱스 : 끝 인덱스 : 슬라이싱 간격]
```

데이터프레임 df의 모든 행에 대하여 0행부터 2행 간격으로 선택하려면 `df.iloc[: : 2]`라고 입력한다.

- **원소 선택**

원소가 위치하는 행과 열의 좌표를 입력하면 해당 위치의 원소가 반환된다. **1개의 행과 2개 이상의 열을 선택하거나 반대로 2개 이상의 행과 1개의 열을 선택하는 경우 시리즈 객체가 반환된다. 2개 이상의 행과 2개 이상의 열을 선택하면, 데이터프레임 객체를 반환한다.**

```
인덱스 이름: DataFrame 객체.loc[행 인덱스, 열 이름]
정수 위치 인덱스: DataFrame 객체.iloc[행 번호, 열 번호]
```

딕셔너리를 데이터프레임 df로 변환하고, set_index() 메소드를 적용하여 '이름' 열을 새로운 행 인덱스로 지정한다.

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'이름' : ['서준', '우현', '인아'],'수학' : [90, 80, 70], '영어' : [98, 89, 95], '음악' : [85, 95, 100], '체육' : [100, 90, 90]}
df = pd.DataFrame(exam_data)

# '이름' 열을 새로운 인덱스로 지정하고, df 객체에 변경 사항 반영
df.set_index('이름', inplace=True)
print(df)
```

데이터프레임 df의 원소 1개를 선택하여 출력하는 방법이다. `loc인덱서나 iloc인덱서 사용하지 않으면 오류 발생`

```
# 데이터프레임 df의 특정 원소 1개 선택('서준'의 '음악' 점수)
a = df.loc['서준', '음악']
print(a)
b = df.iloc[0, 2]
print(b)
```

데이터프레임 df의 원소 2개 이상을 선택하는 것도 가능하다. **2개 이상의 인덱스 배열을 리스트로 입력할 수도 있고, 슬라이싱 기법을 사용할 수도 있다**

```
# 데이터프레임 df의 특정 원소 2개 이상 선택('서준'의 '음악', '체육' 점수)
c = df.loc['서준', ['음악', '체육']]
print(c)
d = df.iloc[0, [2,3]]
print(d)
e = df.loc['서준', '음악':'체육']
print(e)
f = df.iloc[0, 2:]
print(f)
```

데이터프레임의 원소를 선택할 때, 행 인덱스와 열 이름을 각각 2개 이상 선택하여 데이터프레임을 얻는 방법이다.

```
# df 2개 이상의 행과 열에 속하는 원소들 선택('서준', '우현'의 '음악', '체육' 점수)
g = df.loc[['서준', '우현'], ['음악', '체육']]
print(g)
h = df.iloc[[0, 1], [2, 3]]
print(h)
i = df.loc['서준':'우현', '음악':'체육']
print(i)
j = df.iloc[0:2, 2:]
print(j)
```

- **열 추가**

데이터프레임의 마지막 열에 덧붙이듯 새로운 열을 추가한다.

```
열 추가: DataFrame 객체['추가하려는 열 이름'] = 데이터 값
```

이 때 모든 행에 동일한 값이 입력되는 점에 유의한다.

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환, 변수 df에 저장
exam_data = {'이름' : ['서준', '우현', '인아'], '수학' : [90, 80, 70], '영어' : [98, 89, 95], '음악' : [85, 95, 100], '체육' : [100, 90, 90]}
df = pd.DataFrame(exam_data)
print(df)
print('\n')

# 데이터프레임 df에 '국어' 점수 열 추가. 데이터 값은 80 지정
df['국어'] = 80
print(df)
```

- **행 추가**

데이터프레임에 행을 추가하는 방법이다. 추가하려는 행 이름과 데이터 값을 loc 인덱서를 사용하여 입력한다. 하나의 데이터 값을 입력하거나, 열의 개수에 맞게 배열 형태로 여러 개의 값을 입력할 수 있다.

```
행 추가: DataFrame.loc['새로운 행 이름'] = 데이터 값(또는 배열)
```

데이터프레임에 새로운 행을 추가할 때는 기존 행 인덱스와 겹치지 않는 새로운 인덱스를 사용한다. **기존 인덱스와 중복되는 경우, 새로운 행을 추가하지 않고 기존 행의 원소값을 변경한다.**

한편 행 인덱스를 지정할 때 기존 인덱스의 순서를 따르지 않아도 된다.

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환, 변수 df에 저장
exam_data = {'이름' : ['서준', '우현', '인아'],
                '수학' : [90, 80, 70],
                '영어' : [98, 89, 95],
                '음악' : [85, 95, 100],
                '체육' : [100, 90, 90]}
df = pd.DataFrame(exam_data)
print(df)
print('\n')

# 새로운 행(row) 추가 - 같은 원소 값 입력
```

```
df.loc[3] = 0
print(df)
print('\n')

# 새로운 행(row) 추가 - 원소 값 여러 개의 배열 입력
df.loc[4] = ['동규', 90, 80, 70, 60]
print(df)
print('\n')

# 새로운 행 추가(row) - 기존 행 복사
df.loc['행5'] = df.loc[3]
print(df)
```

- **원소 값 변경**

데이터프레임의 특정 원소를 선택하고 새로운 데이터 값을 지정해주면 원소 값이 변경된다. 원소 1개를 선택하여 변경할 수도 있고, 여러 개의 원소를 선택하여 한꺼번에 값을 바꿀 수도 있다.

```
원소 값 변경: DataFrame 객체의 일부분 또는 원소를 선택 = 새로운 값
```

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환, 변수 df에 저장
exam_data = {'이름' : ['서준', '우현', '인아'],
                '수학' : [90, 80, 70],
                '영어' : [98, 89, 95],
                '음악' : [85, 95, 100],
                '체육' : [100, 90, 90]}
df = pd.DataFrame(exam_data)

# '이름' 열을 새로운 인덱스로 지정하고, df 객체에 변경사항 반영
df.set_index('이름', inplace=True)
print(df)
print('\n')

# 데이터프레임 df의 특정 원소를 변경하는 방법: '서준'의 '체육' 점수
df.iloc[0][3] = 80
print(df)
print('\n')

df.loc['서준']['체육'] = 90
print(df)
print('\n')

df.loc['서준', '체육'] = 100
print(df)
```

여러 개의 원소를 선택하여 새로운 값을 할당하면, 모든 원소를 한꺼번에 같은 값으로 변경할 수 있다.

```
# 데이터프레임 df의 원소 여러 개를 변경하는 방법: '서준'의 '음악', '체육' 점수
df.loc['서준',['음악', '체육']] = 50
print(df)
print('\n')

df.loc['서준',['음악', '체육']] = 100, 50
print(df)
```

- **행, 열의 위치 바꾸기**

데이터프레임의 행과 열을 서로 맞바꾸는 방법이다. 선형대수학의 전치행렬과 같은 개념이다.

```
행, 열 바꾸기: DataFrame 객체.transpose() 또는 DataFrame 객체.T
```

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환, 변수 df에 저장
exam_data = {'이름' : ['서준', '우현', '인아'],
                '수학' : [90, 80, 70],
                '영어' : [98, 89, 95],
                '음악' : [85, 95, 100],
                '체육' : [100, 90, 90]}
df = pd.DataFrame(exam_data)
```

```
print(df)
print('\n')

# 데이터프레임 df를 전치하기(메소드 활용)
df = df.transpose()
print(df)
print('\n')

# 데이터프레임 df를 다시 전치하기(클래스 속성 활용)
df = df.T
print(df)
```

인덱스 활용

- **특정 열을 행 인덱스로 설정**

set_index() 메소드를 사용하여 데이터프레임의 특정 열을 행 인덱스로 설정한다. **단, 원본 데이터프레임을 바꾸지 않고 새로운 데이터프레임 객체를 반환하는 점에 유의한다.**

```
특정 열을 행 인덱스로 설정 : DataFrame 객체.set_index(['열 이름'] 또는 '열 이름')
```

원본 데이터프레임에 변경사항을 적용하려면, df = df.set_index('이름')와 같이 원래 변수에 할당하거나, inplace 옵션을 추가해야한다.

```
import pandas as pd

# DataFrame() 함수로 데이터프레임 변환, 변수 df에 저장
exam_data = {'이름' : ['서준', '우현', '인아'],
                '수학' : [90, 80, 70],
                '영어' : [98, 89, 95],
                '음악' : [85, 95, 100],
                '체육' : [100, 90, 90]}
df = pd.DataFrame(exam_data)
print(df)
print('\n')

# 특정 열(column)을 데이터프레임의 행 인덱스(index)로 설정
ndf = df.set_index(['이름'])
print(ndf)
print('\n')

ndf2 = ndf.set_index('음악')
print(ndf2)
print('\n')

ndf3 = ndf.set_index(['수학', '음악'])
print(ndf3)
```

- **행 인덱스 재배열**

reindex() 메소드를 사용하면 데이터프레임의 행 인덱스를 새로운 배열로 재지정할 수 있다. 기존 객체를 변경하지 않고 새로운 데이터프레임 객체를 반환한다.

```
새로운 배열로 행 인덱스를 재지정: DataFrame 객체.reindex(새로운 인덱스 배열)
```

기존 데이터프레임에 존재하지 않는 행 인덱스가 새롭게 추가되는 경우 그 행의 데이터 값은 NaN값이 입력된다.

```
import pandas as pd

# 딕셔너리 정의
dict_data = {'c0' : [1, 2, 3],
            'c1' : [4, 5, 6],
            'c2' : [7, 8, 9],
            'c3' : [10, 11, 12],
            'c4' : [13, 14, 15]}

# 딕셔너리를 데이터프레임으로 변환. 인덱스를 [r0, r1, r2]로 지정
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df)
print('\n')

# 인덱스를 [r0, r1, r2, r3, r4]로 재지정
```

```
new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
ndf = df.reindex(new_index)
print(ndf)
print('\n')

# reindex로 발생한 NaN 값을 숫자 0으로 채우기
new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
ndf2 = df.reindex(new_index, fill_value=0)
print(ndf2)
```

- **행 인덱스 초기화**

reset_index() 메소드를 활용하여 행 인덱스를 정수형 위치 인덱스로 초기화한다. 이때 기존 행 인덱스는 열로 이동한다. 다른 경우와 마찬가지로 새로운 데이터프레임 객체를 반환한다.

```
정수형 위치 인덱스로 초기화: DataFrame 객체.reset_index()
```

```
import pandas as pd

# 딕셔너리 정의
dict_data = {'c0' : [1, 2, 3],
             'c1' : [4, 5, 6],
             'c2' : [7, 8, 9],
             'c3' : [10, 11, 12],
             'c4' : [13, 14, 15]}

# 딕셔너리를 데이터프레임으로 변환. 인덱스를 [r0, r1, r2]로 지정
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df)
print('\n')

# 행 인덱스를 정수형으로 초기화
ndf = df.reset_index()
print(ndf)
```

- **행 인덱스를 기준으로 데이터프레임 정렬**

sort_index() 메소드를 활용하여 행 인덱스를 기준으로 데이터프레임의 값을 정렬한다. ascending 옵션을 사용하여 오름차순 또는 내림차순을 설정한다. 새롭게 정렬된 데이터프레임을 반환한다.

```
행 인덱스 기준 정렬: DataFrame 객체.sort_index()
```

ascending=False 옵션은 내림차순을 반환하고, ascending=True옵션은 오름차순을 반환한다.

```
import pandas as pd

# 딕셔너리 정의
dict_data = {'c0' : [1, 2, 3],
             'c1' : [4, 5, 6],
             'c2' : [7, 8, 9],
             'c3' : [10, 11, 12],
             'c4' : [13, 14, 15]}

# 딕셔너리를 데이터프레임으로 변환. 인덱스를  [r0, r1, r2]로 지정
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df)
print('\n')

# 내림차순으로 행 인덱스 정렬
ndf = df.sort_index(ascending=False)
print(ndf)
```

- **특정 열의 데이터 값을 기준으로 데이터프레임 정렬**

sort_values() 메소드를 활용하며, 새롭게 정렬된 데이터프레임 객체를 반환한다.

```
열 기준 정렬: DataFrame 객체.sort_values()
```

다음의 예제에서는 'c1'열을 기준으로 내림차순 정렬한다.

```
import pandas as pd

# 딕셔너리 정의
dict_data = {'c0' : [1, 2, 3],
             'c1' : [4, 5, 6],
             'c2' : [7, 8, 9],
             'c3' : [10, 11, 12],
             'c4' : [13, 14, 15]}

# 딕셔너리를 데이터프레임으로 변환. 인덱스를 [r0, r1, r2]로 지정
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df)
print('\n')

# c1열을 기준으로 내림차순 정렬
ndf = df.sort_values(by='c1', ascending=False)
print(ndf)
```

산술연산

판다스 객체의 산술연산은 내부적으로 3단계 프로세스를 거친다.

1. **행/열 인덱스를 기준으로 모든 원소를 정렬한다.**
2. **동일한 위치에 있는 원소끼리 일대일로 대응시킨다.**
3. **일대일 대응이 되는 원소끼리 연산을 처리한다.**

시리즈 연산

- **시리즈 vs 숫자**

시리즈 객체에 어떤 숫자를 더하면 시리즈의 개별 원소에 각각 숫자를 더하고 계산한 결과를 시리즈 객체로 반환한다. 덧셈, 뺄셈, 곱셈, 나눗셈 모두 가능하다.

```
시리즈와 숫자 연산: Series객체 + 연산자(+, -, *, /) + 숫자
```

```
import pandas as pd

# 딕셔너리 데이터로 판다스 시리즈 만들기
student1 = pd.Series({'국어' : 100, '영어' : 80, '수학' : 90})
print(student1)
print('\n')

# 학생의 과목별 점수를 200으로 나누기
percentage = student1/100

print(percentage)
print('\n')
print(type(percentage))
```

- **시리즈 vs 시리즈**

시리즈와 시리즈 사이에 사칙연산을 처리하는 방법이다. 시리즈의 모든 인덱스에 대하여 **같은 인덱스를 가진 원소끼리** 계산한다. 인덱스에 연산 결과를 매칭하여 새 시리즈를 반환한다.

```
시리즈와 시리즈 연산: Series1 + 연산자(+, -, *, /) + Series2
```

연산의 결과로 반환된 4개의 시리즈 객체를 DataFrame() 메소드를 이용하여 하나의 데이터프레임으로 합치는 과정을 보여준다.

```
import pandas as pd

# 딕셔너리 데이터로 판다스 시리즈 만들기
student1 = pd.Series({'국어':100, '영어':80, '수학':90})
```

```
student2 = pd.Series({'수학':80, '국어':90, '영어':80})

print(student1)
print('\n')
print(student2)
print('\n')

# 두 학생의 과목별 점수로 사칙연산 수행
addition = student1 + student2
subtraction = student1 - student2
multiplication = student1 * student2
division = student1 / student2
print(type(division))
print('\n')

# 사칙연산 결과를 데이터프레임으로 합치기(시리즈 -> 데이터프레임)
result = pd.DataFrame([addition, subtraction, multiplication, division],
                      index=['덧셈', '뺄셈', '곱셈', '나눗셈'])
print(result)
```

인덱스로 주어진 과목명의 순서가 다르지만, **판다스는 같은 과목명(인덱스)를 찾아 정렬한 후 같은 과목명(인덱스)의 점수(데이터값)끼리 덧셈을 한다.** 덧셈의 결과를 과목명(인덱스)에 매칭시키고 새로운 시리즈 객체를 반환한다.

어느 한쪽에만 인덱스가 존재하고 다른 쪽에는 짝을 지을 수 있는 동일한 인덱스가 없는 경우 정상적으로 연산을 처리할 수 없다. 판다스는 유효한 값이 존재하지 않는다는 의미를 갖는 `nan`으로 처리한다.

```
import pandas as pd
import numpy as np

# 딕셔너리 데이터로 판다스 시리즈 만들기
student1 = pd.Series({'국어':np.nan, '영어':80, '수학':90})
student2 = pd.Series({'수학':80, '국어':90})

print(student1)
print('\n')
print(student2)
print('\n')

# 두 학생의 과목별 점수로 사칙연산 수행(시리즈 vs 시리즈)
addition = student1 + student2
subtraction = student1 - student2
multiplication = student1 * student2
division = student1 / student2
print(type(division))
print('\n')

# 사칙연산 결과를 데이터프레임으로 합치기(시리즈 -> 데이터프레임)
result = pd.DataFrame([addition, subtraction, multiplication, division],
                      index=['덧셈', '뺄셈', '곱셈', '나눗셈'])
print(result)
```

student1의 데이터 값은 np.nan이므로 국어 점수가 존재하지 않는다. **따라서 '국어'열에 있는 원소들은 모두 NaN이 된다.**

student2에 '영어' 인덱스가 없기 때문에 **'영어' 열에 반환되는 원소값은 모두 NaN이다.**

- **연산 메소드**

연산 결과가 NaN으로 반환되는 상황을 피하려면, 연산 메소드에 fill_value 옵션을 설정하여 적용한다.

```
연산 메소드 사용(시리즈와 시리즈 덧셈): Series1.add(Series2, fill_value = 0)
```

```
import pandas as pd
import numpy as np

# 딕셔너리 데이터로 판다스 시리즈 만들기
student1 = pd.Series({'국어':np.nan, '영어':80, '수학':90})
student2 = pd.Series({'수학':80, '국어':90})

print(student1)
print('\n')
print(student2)
print('\n')

# 두 학생의 과목별 점수로 사칙연산 수행(연산 메소드 사용)
sr_add = student1.add(student2, fill_value=0)
sr_sub = student1.sub(student2, fill_value=0)
sr_mul = student1.mul(student2, fill_value=0)
sr_div = student1.div(student2, fill_value=0)
```

```
# 사칙연산 결과를 데이터프레임으로 합치기(시리즈 -> 데이터프레임)
result = pd.DataFrame([sr_add, sr_sub, sr_mul, sr_div],
                      index['덧셈', '뺄셈', '곱셈', '나눗셈'])
print(result)
```

연산 메소드에 fill_value=0 옵션을 설정하여 student1의 국어 점수와 student2의 영어 점수는 NaN대신 0으로 입력된다. 한편 result의 '영어' 열의 나눗셈 값은 무한대이다. student1의 영어 점수 80을 student2의 영어 점수 0으로 나눈 값이기 때문이다.

데이터프레임 연산

- **데이터프레임 vs 숫자**

데이터프레임에 어떤 숫자를 더하면 모든 원소에 숫자를 더한다. 기존 데이터프레임의 형태를 그대로 유지한 채 원소 값만 새로운 계산값으로 바뀐다.

```
데이터프레임과 숫자 연산: DataFrame 객체 + 연산자(+, -, *, /) + 숫자
```

```
import pandas as pd
import seaborn as sns

# titanic 데이터셋에서 age, fare 2개 열을 선택하여 데이터프레임 만들기
titanic = sns.load_dataset('titanic')
df = titanic.loc[:,['age', 'fare']]
print(df.head())                # 첫 5행만 표시
print('\n')
print(type(df))
print('\n')

# 데이터프레임에 숫자 10 더하기
addition = df + 10
print(addition.head())
print('\n')
print(type(addition))
```

- **데이터프레임 vs 데이터프레임**

각 데이터프레임의 같은 행, 같은 열 위치에 있는 원소끼리 계산한다. 이처럼 동일한 위치의 원소끼리 계산한 결과값을 원래 위치에 다시 입력하여 데이터프레임을 만든다.

```
데이터프레임의 연산자 활용: DataFrame1 + 연산자(+, -, *, /) + DataFrame2
```

```
import pandas as pd
import seaborn as sns

# titanic 데이터셋에서 age, fare 2개 열을 선택하여 데이터프레임 만들기
titanic = sns.load_dataset('titanic')
df = titanic.loc[:,['age', 'fare']]
print(df.tail())                # 마지막 5행만 표시
print('\n')
print(type(df))
print('\n')

# 데이터프레임에 숫자 10 더하기
addition = df + 10
print(addition.tail())
print('\n')
print(type(addition))
print('\n')

# 데이터프레임끼리 연산하기 (addition - df)
subtraction = addition - df
print(subtraction.tail())
print('\n')
print(type(subtraction))
```