

# Part 1. 판다스 입문

---

1. 데이터과학자가 판다스를 배우는 이유
2. 판다스 자료구조
  - 2-1. 시리즈
  - 2-2. 데이터프레임
3. 인덱스 활용
4. 산술 연산
  - 4-1. 시리즈 연산
  - 4-2. 데이터프레임 연산

# Part 1. 판다스 입문

## 1. 데이터과학자가 판다스를 배우는 이유

### 빅데이터의 시대. 데이터 과학이라는 새로운 영역의 출현.

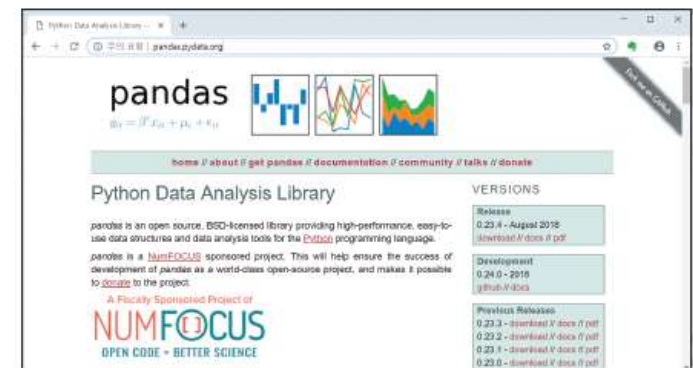
- 클라우드 컴퓨팅의 확산. 빅데이터 저장, 분석에 필요한 컴퓨팅 자원이 매우 저렴해짐.
- 컴퓨팅 파워의 대중화는 최적의 학습환경과 연구 인프라를 제공.

### 데이터과학은 데이터를 연구하는 분야이고, 데이터 자체가 가장 중요한 자원

- 데이터 분석 업무의 80~90%는 데이터를 수집하고 정리하는 일이 차지.
- 나머지 10~20%는 알고리즘을 선택하고, 모델링 결과를 분석하여 데이터로부터 유용한 정보 (information)을 뽑아내는 분석 프로세스의 몫.
- 데이터과학자가 하는 가장 중요한 일이 데이터를 수집하고 분석이 가능한 형태로 정리하는 것.

### 판다스는 데이터를 수집하고 정리하는데 최적화된 도구.

- 가장 배우기 쉬운 프로그래밍 언어, 파이썬(Python) 기반.
- 오픈소스(open source)로 무료로 이용 가능.



[그림 1-1] 판다스 공식 홈페이지(<http://pandas.pydata.org/>)

# Part 1. 판다스 입문

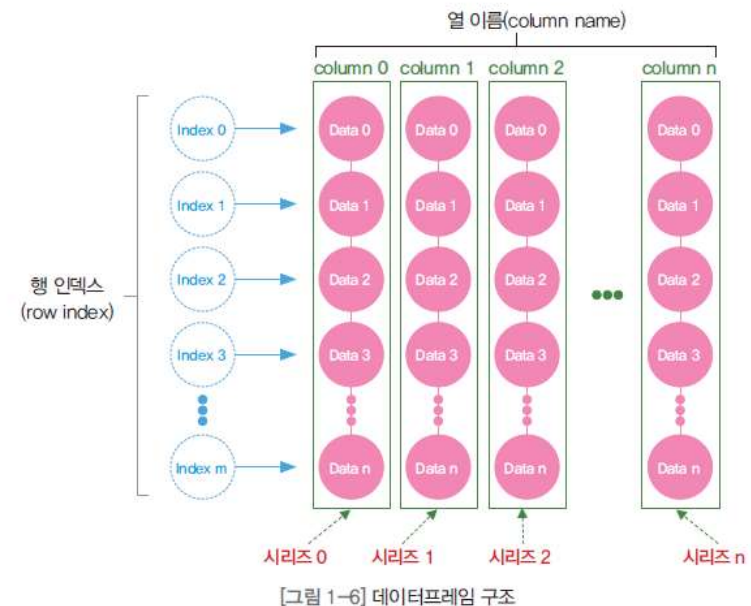
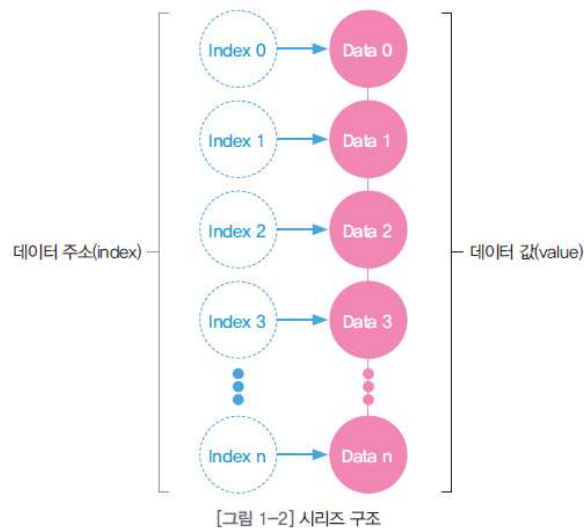
## 2. 판다스 자료구조

### • 목적

분석을 위해 다양한 소스(source)로부터 수집하는 데이터는 형태나 속성이 매우 다양하다. 특히, 서로 다른 형식을 갖는 여러 종류의 데이터를 컴퓨터가 이해할 수 있도록 동일한 형식을 갖는 구조로 통합할 필요가 있다. 판다스의 일차적인 목적은 **형식적으로 서로 다른 여러 가지 유형의 데이터를 공통의 포맷으로 정리**하는 것이다.

### • 종류

판다스는 **시리즈(Series)**와 **데이터프레임(DataFrame)**이라는 **구조화된 데이터 형식**을 제공한다. 서로 다른 종류의 데이터를 한곳에 담는 그릇(컨테이너)이 된다. 다만, 시리즈는 1차원 배열이고, 데이터프레임이 2차원 배열이라는 점에서 차이가 있다. 특히, 행과 열로 이루어진 2차원 구조의 데이터프레임은 데이터 분석 실무에서 자주 사용된다.



# Part 1. 판다스 입문

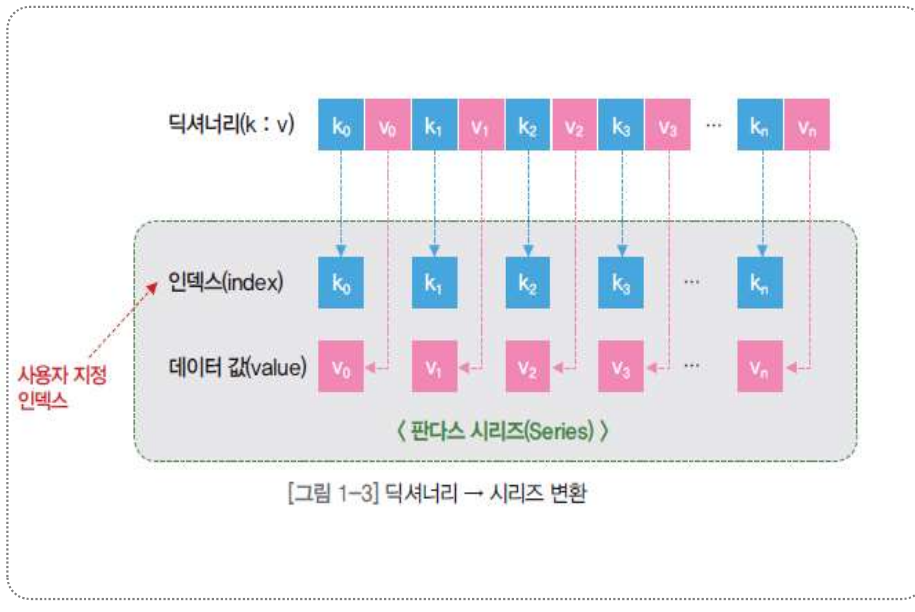
## 2-1. 시리즈

### • 시리즈 만들기

- 1) 딕셔너리와 시리즈의 구조가 비슷하기 때문에, **딕셔너리를 시리즈로 변환**하는 방법을 많이 사용.
- 2) 판다스 내장 함수인 Series()를 이용하고, 딕셔너리를 함수의 매개변수(인자)로 전달.

딕셔너리 → 시리즈 변환: `pandas.Series( 딕셔너리 )`

- 3) 딕셔너리의 키(k)는 시리즈의 인덱스에 대응하고, 딕셔너리의 각 키에 매칭되는 값(v)이 시리즈의 데이터 값(원소)로 변환.



### 예제 1-1

딕셔너리를 시리즈로 변환해 본다. {'a': 1, 'b': 2, 'c': 3}와 같이 'k:v' 구조를 갖는 딕셔너리를 정의하여 변수 dict\_data에 저장한다. 변수 dict\_data에 저장되어 있는 딕셔너리를 Series() 함수의 인자로 전달하면, 시리즈로 변환한다. Series() 함수가 반환한 시리즈 객체를 변수 sr에 저장한다.

〈예제 1-1〉 딕셔너리 → 시리즈 변환

(File: example/part1/1.1\_dict\_to\_series.py)

```
1 # -*- coding: utf-8 -*-
2
3 # pandas 불러오기
4 import pandas as pd
5
6 # key:value 쌍으로 딕셔너리를 만들고, 변수 dict_data에 저장
7 dict_data = {'a': 1, 'b': 2, 'c': 3}
8
9 # 판다스 Series() 함수로 dictionary를 Series로 변환. 변수 sr에 저장
10 sr = pd.Series(dict_data)
11
12 # sr의 자료형 출력
13 print(type(sr))
14 print('\n')
15 # 변수 sr에 저장되어 있는 시리즈 객체를 출력
16 print(sr)
```

〈실행 결과〉 코드 전부 실행

```
<class 'pandas.core.series.Series'>

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

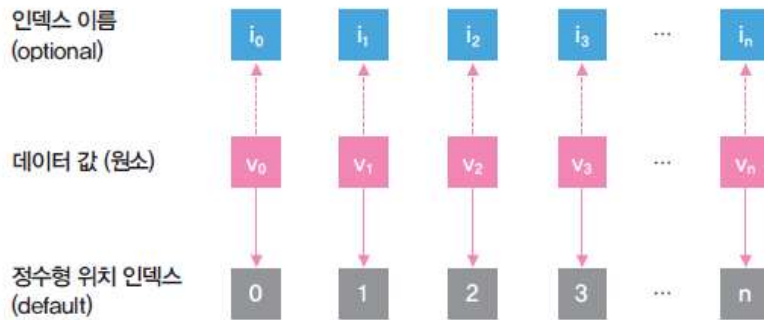
# Part 1. 판다스 입문

## 2-1. 시리즈

### • 인덱스 구조

- 1) 인덱스는 자기와 짝을 이루는 **원소의 순서와 주소를 저장**.
- 2) 인덱스를 잘 활용하면 데이터 값의 탐색, 정렬, 선택, 결합 등 데이터 조작을 쉽게 할 수 있다.
- 3) **인덱스의 종류 (2가지)**

- ① 정수형 위치 인덱스(integer position)
- ② 인덱스 이름(index name) 또는 인덱스 라벨(index label)



[그림 1-4] 시리즈 인덱스의 유형

### 예제 1-2

#### ① 시리즈 만들기

파이썬 리스트를 시리즈로 변환해 본다. 단, 딕셔너리의 키처럼 인덱스로 변환될 값이 없다. 따라서, 인덱스를 별도로 정의하지 않으면, 디폴트로 정수형 위치 인덱스(0, 1, 2, ...)가 자동 지정된다. 다음 예제에서는 0 ~ 4 범위의 정수값이 인덱스로 지정된다.

<예제 1-2> 시리즈 인덱스

(File: example/part1/1.2\_series\_index.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 리스트를 시리즈로 변환하여 변수 sr에 저장
6 list_data = ['2019-01-02', 3.14, 'ABC', 100, True]
7 sr = pd.Series(list_data)
8 print(sr)
```

<실행 결과> 코드 1~8라인을 부분 실행

```
0    2019-01-02
1         3.14
2         ABC
3         100
4         True
dtype: object
```

#### ② 인덱스 vs. 데이터 값 배열 확인하기

시리즈의 index 속성과 values 속성을 이용하면, 인덱스 배열과 데이터 값의 배열을 불러올 수 있다. .

<예제 1-2> 시리즈 인덱스

(File: example/part1/1.2\_series\_index.py(이어서 계속))

```
~ ~ ~ 생략 ~ ~ ~
11 # 인덱스 배열은 변수 idx에 저장. 데이터 값 배열은 변수 val에 저장
12 idx = sr.index
13 val = sr.values
14 print(idx)
15 print('\n')
16 print(val)
```

<실행 결과> 코드 11~16라인을 부분 실행

```
RangeIndex(start=0, stop=5, step=1)

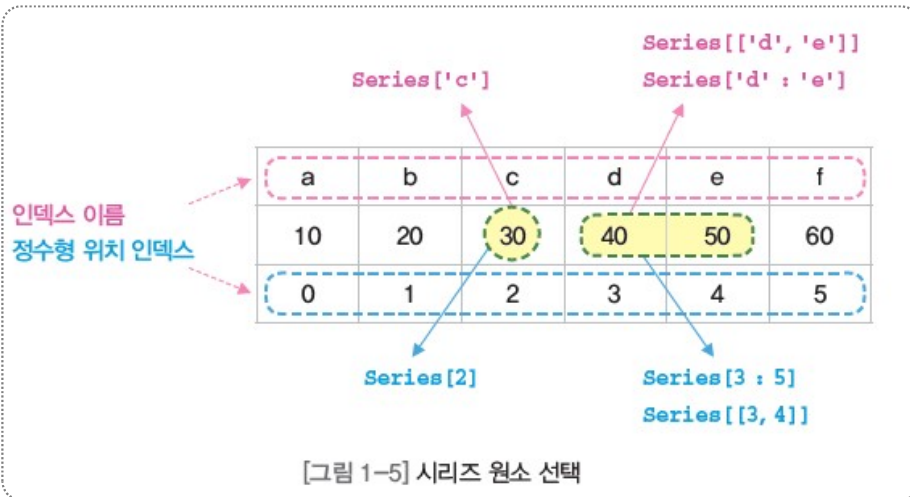
['2019-01-02' 3.14 'ABC' 100 True]
```

# Part 1. 판다스 입문

## 2-1. 시리즈

### • 원소 선택

- 1) 인덱스를 이용하여, 시리즈의 원소를 선택.
- 2) 하나의 원소를 선택하거나, 여러 원소를 한꺼번에 선택 가능.
- 3) 인덱스 범위를 지정하여 여러 개의 원소 선택 가능.
- 4) 인덱스의 유형에 따라 사용법이 조금 다르다.
  - 정수형 인덱스: 대괄호([ ]) 안에 숫자 입력. (0부터 시작)
  - 인덱스 이름(라벨): 대괄호([ ]) 안에 이름과 함께 따옴표를 입력. 큰 따옴표(" "), 작은 따옴표(' ') 모두 사용.



### 예제 1-3

#### ① 시리즈 만들기

파이썬 튜플을 시리즈로 변환한다. 정수형 위치 인덱스 대신에 인덱스 이름을 따로 지정할 수 있다. Series() 함수의 index 옵션에 인덱스 이름을 리스트 형태로 직접 전달하는 방식이다.

#### <예제 1-3> 시리즈 원소 선택

(File: example/part1/1.3\_series\_element.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 튜플을 시리즈로 변환 (인덱스 옵션 지정)
6 tup_data = ('영인', '2010-05-01', '여', True)
7 sr = pd.Series(tup_data, index=['이름', '생년월일', '성별', '학생여부'])
8 print(sr)
```

<실행 결과> 코드 1~8라인을 부분 실행

이름	영인
생년월일	2010-05-01
성별	여
학생여부	True
dtype:	object

#### ② 원소 1개 선택

인덱스를 이용하여 원소를 선택할 때는 대괄호([ ]) 안에 인덱스를 입력한다. 시리즈의 첫 번째 데이터를 선택하기 위해 정수형 위치 인덱스(0)와 첫 번째 위치에 있는 인덱스 라벨('이름')을 입력한다.

#### <예제 1-3> 시리즈 원소 선택

(File: example/part1/1.3\_series\_element.py(이어서 계속))

```
~ ~ ~ 생략 ~ ~ ~
11 # 원소를 1개 선택
12 print(sr[0])          # sr의 1번째 원소를 선택 (정수형 위치 인덱스)
13 print(sr['이름'])     # '이름' 라벨을 가진 원소를 선택 (인덱스 이름)
```

<실행 결과> 코드 11~13라인을 부분 실행

영인
영인

# Part 1. 판다스 입문

## 2-1. 시리즈

### • 원소 선택 (계속)

#### ③ 여러 개의 원소를 선택(인덱스 리스트 활용)

인덱스 리스트를 활용하는 방법이다.

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

정수형 위치 인덱스는 0부터 시작하기 때문에 2번째 인덱스 이름인 '생년월일'은 정수형 인덱스 1을 사용하고, 3번째 인덱스 이름인 '성별'은 정수형 인덱스 2를 사용한다.

〈예제 1-3〉 시리즈 원소 선택 (File: example/part1/1.3\_series\_element.py(이어서 계속))

```
~ ~ ~ 선택 ~ ~ ~  
  
16 # 여러 개의 원소를 선택 (인덱스 리스트 활용)  
17 print(sr[[1, 2]])  
18 print('\n')  
19 print(sr[['생년월일', '성별']])
```

〈실행 결과〉 코드 16~19라인을 부분 실행

```
생년월일    2010-05-01  
성별         여  
dtype: object  
  
생년월일    2010-05-01  
성별         여  
dtype: object
```

#### ④ 여러 개의 원소를 선택(인덱스 범위 지정)

인덱스 범위를 지정하여 선택하는 방법이다.

22번 라인의 sr[1 : 2] 에서 정수형 위치 인덱스를 사용할 때는, 범위의 끝(2)이 포함되지 않는다('성별' 불포함).

그러나, 21번 라인의 sr['생년월일' : '성별'] 과 같이, 인덱스 이름을 사용하면 범위의 끝('성별')이 포함된다.

〈예제 1-3〉 시리즈 원소 선택

(File: example/part1/1.3\_series\_element.py(이어서 계속))

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

〈실행 결과〉 코드 22~25라인을 부분 실행

```
생년월일    2010-05-01  
dtype: object  
  
생년월일    2010-05-01  
성별         여  
dtype: object
```

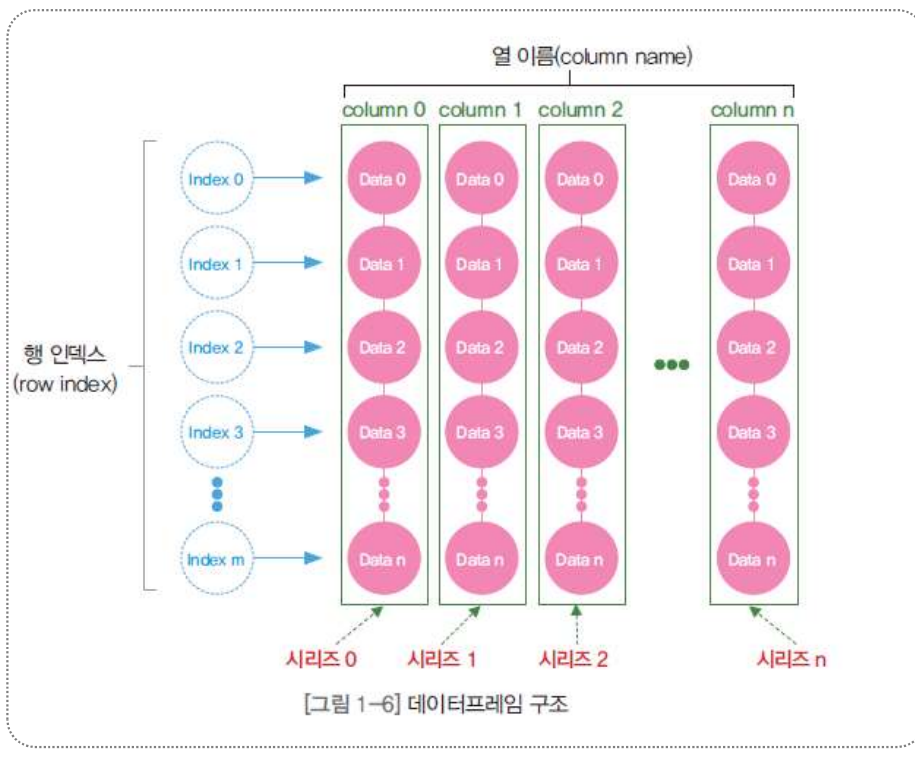


# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 개요

- 1) 데이터프레임은 2차원 배열. R의 데이터프레임에서 유래.
- 2) 데이터프레임의 열은 시리즈 객체. 시리즈를 열벡터(vector)라고 하면, 데이터프레임은 여러 개의 열벡터들이 같은 행 인덱스를 기준으로 줄지어 결합된 2차원 벡터 또는 행렬(matrix).
- 3) 데이터프레임은 행과 열을 나타내기 위해 두 가지 종류의 주소를 사용. 행 인덱스(row index)와 열 이름(column name 또는 column label)으로 구분.



- 4) 데이터프레임의 각 열은 공통의 속성을 갖는 일련의 데이터를 나타냄.
- 5) 각 행은 개별 관측대상에 대한 다양한 속성 데이터들의 모음인 레코드(record).

[예시]

다음 주식종목 리스트에서, 각 행은 하나의 주식종목에 관한 관측값(observation)을 나타낸다.

각 열은 종목코드, 회사이름, 액면가, 총주식수 등 공통의 속성이거나 범주를 나타내는데, 보통 변수(variable)로 활용된다

종목 코드	회사 이름	액면가	총 주식수
005930	삼성전자	100원	5,970백만 주
017670	SK텔레콤	500원	81백만 주
005380	현대자동차	5000원	214백만 주

[표 1-1] 주식 종목 리스트



# Part 1. 판다스 입문

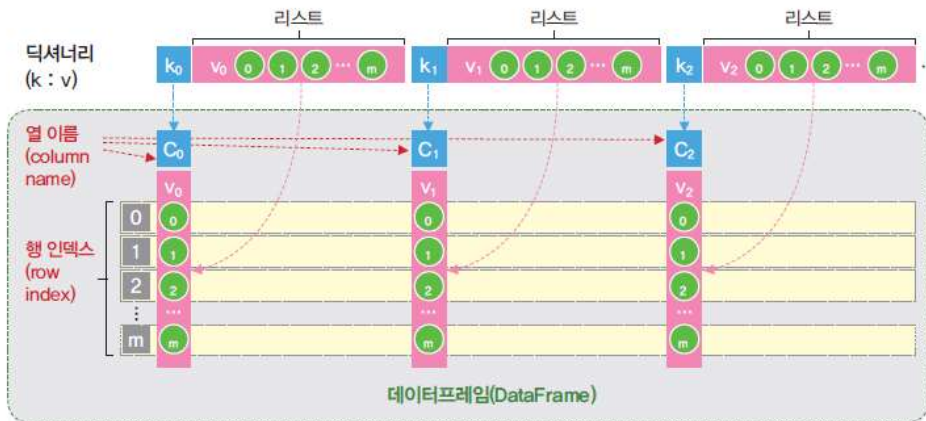
## 2-2. 데이터프레임

### • 데이터프레임 만들기

- 1) 같은 길이(원소의 개수가 동일한)의 배열 여러 개가 필요. 데이터프레임은 여러 개의 시리즈(열, column)를 모아 놓은 집합.
- 2) 판다스 DataFrame() 함수를 사용. 여러 개의 리스트를 원소로 갖는 딕셔너리를 함수에 전달하는 방식을 주로 활용.

딕셔너리 → 데이터프레임 변환: `pandas.DataFrame( 딕셔너리 객체 )`

- 3) 딕셔너리의 값(v)에 해당하는 각 리스트가 시리즈로 변환되어 데이터프레임의 각 열이 된다.
- 4) 딕셔너리의 키(k)는 각 시리즈의 이름으로 변환되어, 최종적으로 데이터프레임의 열 이름이 된다.



[그림 1-7] 딕셔너리 → 데이터프레임 변환

### 예제 1-4

원소 3개씩 담고 있는 리스트를 5개 만든다. 이들 5개의 리스트를 원소로 갖는 딕셔너리를 정의하고, 판다스 DataFrame() 함수에 전달하면 5개의 열을 갖는 데이터프레임을 만든다.

이때, 딕셔너리의 키(k)가 열 이름( $c_0 \sim c_4$ )이 되고, 값(v)에 해당하는 각 리스트가 데이터프레임의 열이 된다. 행 인덱스에는 정수형 위치 인덱스(0, 1, 2)가 자동 지정된다.

<예제 1-4> 딕셔너리 → 데이터프레임 변환

(File: example/part1/1.4\_dict\_to\_dataframe.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 열이름을 key로 하고, 리스트를 value로 갖는 딕셔너리 정의 (2차원 배열)
6 dict_data = {'c0': [1,2,3], 'c1': [4,5,6], 'c2': [7,8,9], 'c3': [10,11,12], 'c4': [13,14,15]}
7
8 # 판다스 DataFrame() 함수로 딕셔너리를 데이터프레임으로 변환. 변수 df에 저장
9 df = pd.DataFrame(dict_data)
10
11 # df의 자료형 출력
12 print(type(df))
13 print('\n')
14 # 변수 df에 저장되어 있는 데이터프레임 객체를 출력
15 print(df)
```

<실행 결과> 코드 전부 실행

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

	c0	c1	c2	c3	c4
0	1	4	7	10	13
1	2	5	8	11	14
2	3	6	9	12	15

# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 행 인덱스/열 이름 설정

: 데이터프레임의 행 인덱스와 열 이름을 사용자가 지정 가능.

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

#### 예제 1-5

#### ① 데이터프레임을 만들 때 설정

'3개의 원소를 갖는 리스트' 2개를 원소로 갖는 리스트(2차원 배열)로 데이터프레임을 만든다. 이때, 각 리스트가 행으로 변환되는 점에 유의한다.

- index 옵션: ['준서', '예은'] 배열을 지정
- columns 옵션: ['나이', '성별', '학교'] 배열을 지정

〈예제 1-5〉 행 인덱스/열 이름 설정 (File: example/part1/1.5\_change\_df\_idx\_col.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 행 인덱스/열 이름 지정하여 데이터프레임 만들기
6 df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
7                   index=['준서', '예은'],
8                   columns=['나이', '성별', '학교'])
9
10 # 행 인덱스, 열 이름 확인하기
11 print(df)          # 데이터프레임
12 print('\n')
13 print(df.index)     # 행 인덱스
14 print('\n')
15 print(df.columns)  # 열 이름
```

〈예제 1-5〉 행 인덱스/열 이름 설정

(File: example/part1/1.5\_change\_df\_idx\_col.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 행 인덱스/열 이름 지정하여 데이터프레임 만들기
6 df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
7                   index=['준서', '예은'],
8                   columns=['나이', '성별', '학교'])
9
10 # 행 인덱스, 열 이름 확인하기
11 print(df)          # 데이터프레임
12 print('\n')
13 print(df.index)     # 행 인덱스
14 print('\n')
15 print(df.columns)  # 열 이름
```

〈실행 결과〉 코드 1~15라인을 부분 실행

```
나이 성별 학교
준서  15  남  덕영중
예은  17  여  수리중

Index(['준서', '예은'], dtype='object')

Index(['나이', '성별', '학교'], dtype='object')
```

#### ② 속성을 지정하여 변경하기

데이터프레임 df의 행 인덱스 배열을 나타내는 df.index와 열 이름 배열을 나타내는 df.columns에 새로운 배열을 할당하는 방식으로, 행 인덱스와 열 이름을 변경할 수 있다.

- 행 인덱스 변경: `DataFrame 객체.index = 새로운 행 인덱스 배열`
- 열 이름 변경: `DataFrame 객체.columns = 새로운 열 이름 배열`

# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 행 인덱스/열 이름 설정 (계속)

〈예제 1-5〉 행 인덱스/열 이름 설정 (File: example/part1/1.5\_change\_df\_idx\_col.py(이어서 계속))

```
~ ~ ~ 생략 ~ ~ ~

18 # 행 인덱스, 열 이름 변경하기
19 df.index=['학생1', '학생2']
20 df.columns=['연령', '남녀', '소속']
21
22 print(df)           # 데이터프레임
23 print('\n')
24 print(df.index)     # 행 인덱스
25 print('\n')
26 print(df.columns)   # 열 이름
```

〈실행 결과〉 코드 18~26라인을 부분 실행

```
연령  남녀  소속
학생1  15  남  덕영중
학생2  17  여  수리중

Index(['학생1', '학생2'], dtype='object')

Index(['연령', '남녀', '소속'], dtype='object')
```

### ③ rename 메소드 사용

rename() 메소드를 적용하면, 행 인덱스 또는 열 이름의 일부를 선택하여 변경 가능. 단, 새로운 데이터프레임 객체를 리턴.

\* 원본 객체를 변경하려면, inplace=True 옵션을 사용.

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

### 예제 1-6

〈예제 1-6〉 행 인덱스/열 이름 변경 (File: example/part1/1.6\_change\_df\_idx\_col2.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 행 인덱스/열 이름 지정하여 데이터프레임 만들기
6 df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
7                   index=['준서', '예은'],
8                   columns=['나이', '성별', '학교'])
9
10 # 데이터프레임 df 출력
11 print(df)
12 print("\n")
13
14 # 열 이름 중, '나이'를 '연령'으로, '성별'을 '남녀'로, '학교'를 '소속'으로 바꾸기
15 df.rename(columns={'나이':'연령', '성별':'남녀', '학교':'소속'}, inplace=True)
16
17 # df의 행 인덱스 중에서, '준서'를 '학생1'로, '예은'을 '학생2'로 바꾸기
18 df.rename(index={'준서':'학생1', '예은':'학생2'}, inplace=True)
19
20 # df 출력(변경 후)
21 print(df)
```

〈실행 결과〉 코드 전부 실행

```
나이  성별  학교
준서   15  남  덕영중
예은   17  여  수리중

연령  남녀  소속
학생1  15  남  덕영중
학생2  17  여  수리중
```

# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 행/열 삭제

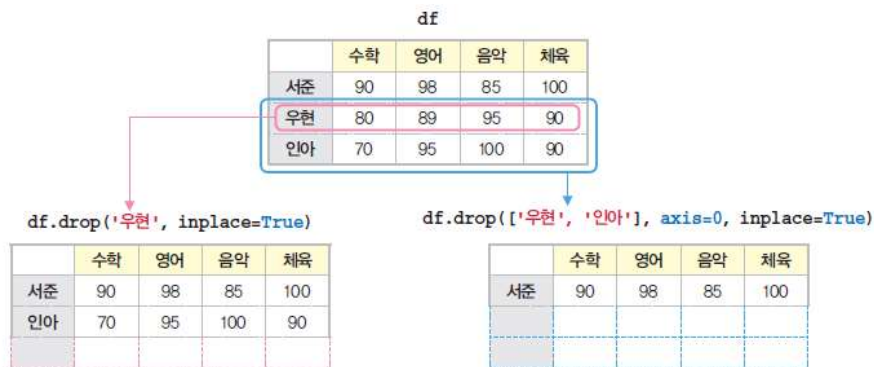
drop() 메소드에 행을 삭제할 때는 축(axis) 옵션으로 axis=0을 입력하거나, 별도로 입력하지 않는다.

반면, 축옵션으로 axis=1을 입력하면 열을 삭제한다. 동시에 여러 개의 행 또는 열을 삭제하려면, 리스트 형태로 입력한다.

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

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

#### ① 행 삭제



[그림 1-9] 행 삭제

#### <예제 1-7> 행 삭제

(File: example/part1/1.7\_remove\_row.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6 exam_data = {'수학' : [ 90, 80, 70], '영어' : [ 98, 89, 95],
7              '음악' : [ 85, 95, 100], '체육' : [ 100, 90, 90]}
8
9 df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
10 print(df)
11 print('\n')
12
13 # 데이터프레임 df를 복제하여 변수 df2에 저장. df2의 1개 행(row) 삭제
14 df2 = df[:]
15 df2.drop('우현', inplace=True)
16 print(df2)
17 print('\n')
18
19 # 데이터프레임 df를 복제하여 변수 df3에 저장. df3의 2개 행(row) 삭제
20 df3 = df[:]
21 df3.drop(['우현', '인아'], axis=0, inplace=True)
22 print(df3)
```

#### <실행 결과> 코드 전부 실행

	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

	수학	영어	음악	체육
서준	90	98	85	100
인아	70	95	100	90

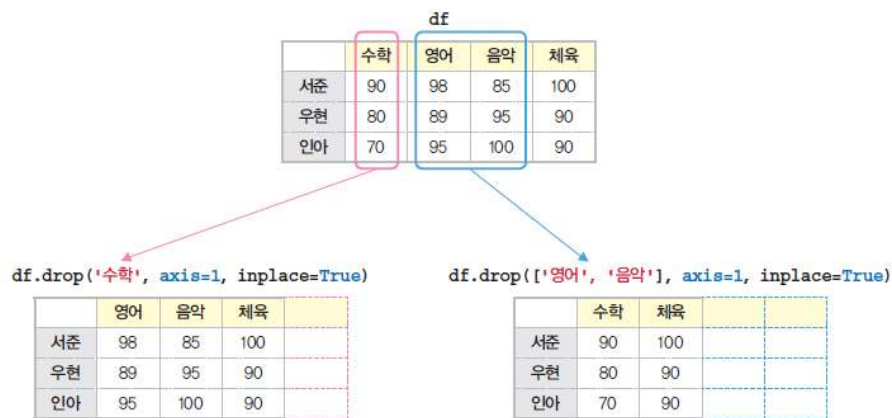
	수학	영어	음악	체육
서준	90	98	85	100

# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 행/열 삭제 (계속)

#### ② 열 삭제



[그림 1-10] 열 삭제

#### 〈예제 1-8〉 열 삭제

(File: example/part1/1.8\_remove\_column.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6 exam_data = {'수학' : [ 90, 80, 70], '영어' : [ 98, 89, 95],
7              '음악' : [ 85, 95, 100], '체육' : [ 100, 90, 90]}
8
9 df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
10 print(df)
11 print('\n')
12
13 # 데이터프레임 df를 복제하여 변수 df4에 저장. df4의 1개 열(column) 삭제
14 df4 = df[:]
15 df4.drop('수학', axis=1, inplace=True)
16 print(df4)
17 print('\n')
```

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

#### 〈실행 결과〉 코드 전부 실행

	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

	영어	음악	체육
서준	98	85	100
우현	89	95	90
인아	95	100	90

	수학	체육
서준	90	100
우현	80	90
인아	70	90



# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 행 선택

- 1) **loc**과 **iloc** 인덱서를 사용.
- 2) 인덱스 이름을 기준으로 행을 선택할 때는 loc을 이용하고, 정수형 위치 인덱스를 사용할 때는 iloc을 이용.

구분	loc	iloc
탐색 대상	인덱스 이름(index label)	정수형 위치 인덱스(integer position)
범위 지정	가능(범위의 끝 포함) 예) ['a':'c'] → 'a', 'b', 'c'	가능(범위의 끝 제외) 예) [3:7] → 3, 4, 5, 6 (* 7 제외)

[표 1-2] loc과 iloc

#### 예제 1-9

#### ① 1개의 행 선택

```
<예제 1-9> 행 선택 (File: example/part1/1.9_select_row.py)
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6 exam_data = {'수학': [ 90, 80, 70], '영어': [ 98, 89, 95],
7              '음악': [ 85, 95, 100], '체육': [ 100, 90, 90]}
8
9 df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
10 print(df)
11 print('\n')
12
13 # 행 인덱스를 사용하여 행 1개 선택
14 label1 = df.loc['서준']
15 position1 = df.iloc[0]
16 print(label1)
17 print('\n')
18 print(position1)
```

<실행 결과> 코드 1~18라인을 부분 실행

	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

```
수학      90
영어      98
음악      85
체육     100
Name: 서준, dtype: int64
```

```
수학      90
영어      98
음악      85
체육     100
Name: 서준, dtype: int64
```

데이터프레임의 첫 번째 행에는 '서준' 학생의 과목별 점수 데이터가 입력되어 있다. '서준' 학생의 과목별 점수 데이터를 행으로 추출하면 시리즈 객체가 반환된다.

loc 인덱서를 이용하려면 '서준'이라는 인덱스 이름을 직접 입력하고, iloc을 이용할 때는 첫 번째 정수형 위치를 나타내는 0을 입력한다. 각각 반환되는 값을 label1 변수와 position1 변수에 저장, 출력하면 같은 결과를 갖는다.

# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 행 선택 (계속)

#### ② 여러 개의 행을 선택(인덱스 리스트 활용)

〈예제 1-9〉 행 선택 (File: example/part1/1.9\_select\_row.py(이어서 계속))

~ ~ ~ (생략) ~ ~ ~

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

〈실행 결과〉 코드 21~26라인을 부분 실행

	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90

	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90

2개 이상의 행 인덱스를 배열로 입력하면, 매칭되는 모든 행 데이터를 동시에 추출한다.

데이터프레임 df의 첫번째와 두번째 행에 있는 '서준', '우현' 학생을 인덱싱으로 선택해 본다. loc 인덱서는 ['서준', '우현'] 과 같이 인덱스 이름을 배열로 전달하고, iloc을 이용할 때는 [0, 1] 과 같이 정수형 위치를 전달한다. 이때, label2 변수와 position2 변수에 저장된 값은 같다.

#### ③ 여러 개의 원소를 선택(인덱스 범위 지정)

〈예제 1-9〉 행 선택 (File: example/part1/1.9\_select\_row.py(이어서 계속))

~ ~ ~ (생략) ~ ~ ~

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

〈실행 결과〉 코드 29~34라인을 부분 실행

	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90

	수학	영어	음악	체육
서준	90	98	85	100

행 인덱스의 범위를 지정하여 여러 개의 행을 동시에 선택하는 슬라이싱 기법을 사용한다.

단, 인덱스 이름을 범위로 지정한 label3의 경우에는 범위의 마지막 값인 '우현' 학생의 점수가 포함되지만, 정수형 위치 인덱스를 사용한 position3에는 범위의 마지막 값인 '우현' 학생의 점수가 제외된다.



# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 열 선택

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

	수학
0	90
1	80
2	70

시리즈

	음악	체육
0	85	100
1	95	90
2	100	90

데이터프레임

[그림 1-11] 데이터프레임의 열 선택

#### 예제 1-10

##### ① 1개의 열 선택

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

<예제 1-10> 열 선택

(File: example/part1/1.10\_select\_column.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6 exam_data = {'이름' : [ '서준', '우현', '인아'],
7              '수학' : [ 90, 80, 70],
8              '영어' : [ 98, 89, 95],
9              '음악' : [ 85, 95, 100],
10             '체육' : [ 100, 90, 90]}
11 df = pd.DataFrame(exam_data)
12 print(df)
```

```
13 print(type(df))
14 print('\n')
15
16 # '수학' 점수 데이터만 선택. 변수 math1에 저장
17 math1 = df['수학']
18 print(math1)
19 print(type(math1))
20 print('\n')
21
22 # '영어' 점수 데이터만 선택. 변수 english에 저장
23 english = df.영어
24 print(english)
25 print(type(english))
```

<실행 결과> 코드 1~25라인을 부분 실행

```
이름  수학  영어  음악  체육
0  서준   90   98   85   100
1  우현   80   89   95   90
2  인아   70   95  100   90

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

0    90
1    80
2    70
Name: 수학, dtype: int64
<class 'pandas.core.series.Series'>

0    98
1    89
2    95
Name: 영어, dtype: int64
<class 'pandas.core.series.Series'>
```

type() 함수를 사용하여, 데이터프레임에서 1개의 열을 선택할 때 반환되는 객체의 자료형을 확인하면 시리즈이다.

# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 열 선택 (계속)

#### ② n개의 열 선택 (리스트 입력)

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

〈예제 1-10〉 열 선택 (File: example/part1/1.10\_select\_column.py(이어서 계속))

~ ~ ~ (생략) ~ ~ ~

```
28 # '음악', '체육' 점수 데이터를 선택. 변수 music_gym에 저장
29 music_gym = df[['음악', '체육']]
30 print(music_gym)
31 print(type(music_gym))
32 print('\n')
33
34 # '수학' 점수 데이터만 선택. 변수 math2에 저장
35 math2 = df[['수학']]
36 print(math2)
37 print(type(math2))
```

〈실행 결과〉 코드 28~37라인을 부분 실행

```
음악  체육
0   85   100
1   95    90
2  100    90
<class 'pandas.core.frame.DataFrame'>

수학
0   90
1   80
2   70
<class 'pandas.core.frame.DataFrame'>
```

이 때, 반환되는 객체의 자료형을 확인하면 데이터프레임이다.

### • 원소 선택

〈원소 1개 선택〉

	0	1	2	3
	수학	영어	음악	체육
0 서준	90	98	85	100
1 우현	80	89	95	90
2 인아	70	95	100	90

행, 열 좌표 { df.loc['서준', '음악'] } 0 서준 85  
df.iloc[0, 2] 원소

〈원소 2개(시리즈) 선택〉

	0	1	2	3
	수학	영어	음악	체육
0 서준	90	98	85	100
1 우현	80	89	95	90
2 인아	70	95	100	90

행, 열 좌표 { df.loc['서준', ['음악', '체육']]  
df.iloc[0, [2, 3]]  
df.loc['서준', '음악': '체육']  
df.iloc[0, 2:] } 0 서준 85 100  
시리즈

〈데이터프레임(df)의 일부분 선택〉

	0	1	2	3
	수학	영어	음악	체육
0 서준	90	98	85	100
1 우현	80	89	95	90
2 인아	70	95	100	90

행, 열 좌표 { df.loc[['서준', '우현'], ['음악', '체육']]  
df.iloc[[0, 1], [2, 3]]  
df.loc['서준': '우현', '음악': '체육']  
df.iloc[0:2, 2:] } 0 서준 85 100  
1 우현 95 90  
데이터프레임

〈그림 1-12〉 데이터프레임의 [행, 열] 데이터 선택

### 예제 1-11

〈예제 1-11〉 원소 선택

(File: example/part1/1.11\_select\_element.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6 exam_data = {'이름' : [ '서준', '우현', '인아'],
7              '수학' : [ 90, 80, 70],
8              '영어' : [ 98, 89, 95],
9              '음악' : [ 85, 95, 100],
10             '체육' : [ 100, 90, 90]}
11 df = pd.DataFrame(exam_data)
12
13 # '이름' 열을 새로운 인덱스로 지정하고, df 객체에 변경 사항 반영
14 df.set_index('이름', inplace=True)
15 print(df)
```

# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 원소 선택 (계속)

#### ① 1개의 원소를 선택

〈예제 1-11〉 원소 선택 (File: example/part1/1.11\_select\_element.py(0)에서 계속)

```
~ ~ ~ ~ 생략 ~ ~ ~ ~

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

〈실행 결과〉 코드 18~22라인을 부분 실행

```
85
85
```

#### ② 2개 이상의 원소를 선택 (시리즈)

〈예제 1-11〉 원소 선택 (File: example/part1/1.11\_select\_element.py(0)에서 계속)

```
~ ~ ~ ~ 생략 ~ ~ ~ ~

25 # 데이터프레임 df의 특정 원소 2개 이상 선택 ('서준'의 '음악', '체육' 점수)
26 c = df.loc['서준', ['음악', '체육']]
27 print(c)
28 d = df.iloc[0, [2, 3]]
29 print(d)
30 e = df.loc['서준', '음악': '체육']
31 print(e)
32 f = df.iloc[0, 2:]
33 print(f)
```

〈실행 결과〉 코드 25~33라인을 부분 실행

```
음악      85
체육     100
Name: 서준, dtype: int64
음악      85
체육     100
Name: 서준, dtype: int64
```

```
음악      85
체육     100
Name: 서준, dtype: int64
음악      85
체육     100
Name: 서준, dtype: int64
```

#### ③ 2개 이상의 원소를 선택 (데이터프레임)

〈예제 1-11〉 원소 선택 (File: example/part1/1.11\_select\_element.py(0)에서 계속)

```
~ ~ ~ ~ 생략 ~ ~ ~ ~

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

〈실행 결과〉 코드 36~44라인을 부분 실행

```
음악  체육
이름
서준  85  100
우현  95   90
음악  체육
이름
서준  85  100
우현  95   90
음악  체육
이름
서준  85  100
우현  95   90
음악  체육
이름
서준  85  100
우현  95   90
```

# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 열 추가

- 1) 추가하려는 열 이름과 데이터 값을 입력. 마지막 열에 덧붙이듯 새로운 열을 추가.

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

- 2) 이때 모든 행에 동일한 값이 입력되는 점에 유의.



[그림 1-13] 데이터프레임의 열 추가

### 예제 1-12

다음 예제에서 '국어' 열을 새로 추가하는데, 모든 학생들의 국어 점수가 동일하게 80점으로 입력되는 과정을 보여준다.

〈예제 1-12〉 열 추가 (File: example/part1/1.12\_add\_column.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6 exam_data = {'이름' : [ '서준', '우현', '인아'],
7              '수학' : [ 90, 80, 70],
8              '영어' : [ 98, 89, 95],
9              '음악' : [ 85, 95, 100],
10             '체육' : [ 100, 90, 90]}
11 df = pd.DataFrame(exam_data)
12 print(df)
13 print('\n')
14
15 # 데이터프레임 df에 '국어' 점수 열(column) 추가. 데이터 값은 80 지정
16 df['국어'] = 80
17 print(df)
```

〈실행 결과〉 코드 전부 실행

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

	이름	수학	영어	음악	체육	국어
0	서준	90	98	85	100	80
1	우현	80	89	95	90	80
2	인아	70	95	100	90	80

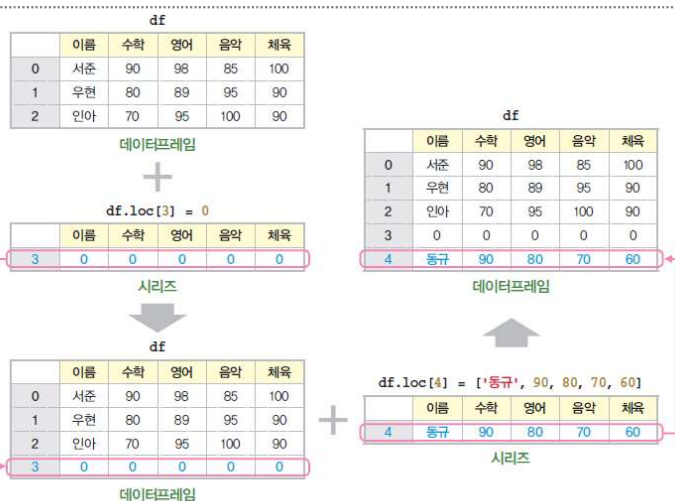
# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 행 추가

: 행 인덱스와 데이터 값을 loc 인덱서를 사용하여 입력.

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



[그림 1-14] 데이터프레임의 행 추가

#### <예제 1-13> 행 추가

(File: example/part1/1.13\_add\_row.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6 exam_data = {'이름' : ['서준', '우현', '인아'],
7              '수학' : [ 90, 80, 70],
8              '영어' : [ 98, 89, 95],
9              '음악' : [ 85, 95, 100],
10             '체육' : [ 100, 90, 90]}
11 df = pd.DataFrame(exam_data)
12 print(df)
```

```
13 print('\n')
14
15 # 새로운 행(row) 추가 - 같은 원소 값 입력
16 df.loc[3] = 0
17 print(df)
18 print('\n')
19
20 # 새로운 행(row) 추가 - 원소 값 여러 개의 배열 입력
21 df.loc[4] = ['동규', 90, 80, 70, 60]
22 print(df)
23 print('\n')
24
25 # 새로운 행(row) 추가 - 기존 행 복사
26 df.loc['행5'] = df.loc[3]
27 print(df)
```

#### <실행 결과> 코드 전부 실행

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90
3	0	0	0	0	0

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90
3	0	0	0	0	0
4	동규	90	80	70	60

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90
3	0	0	0	0	0
4	동규	90	80	70	60
행5	0	0	0	0	0



# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 원소 값 변경

: 원소를 선택하고 새로운 데이터 값을 지정.

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

#### ① 1개의 원소를 변경

〈예제 1-14〉 원소 값 변경

(File: example/part1/1.14\_modify\_dataframe\_element.py)

```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6  exam_data = {'이름' : [ '서준', '우현', '인아'],
7              '수학' : [ 90, 80, 70],
8              '영어' : [ 98, 89, 95],
9              '음악' : [ 85, 95, 100],
10             '체육' : [ 100, 90, 90]}
11  df = pd.DataFrame(exam_data)
12
13  # '이름' 열을 새로운 인덱스로 지정하고, df 객체에 변경사항 반영
14  df.set_index('이름', inplace=True)
15  print(df)
16  print('\n')
17
18  # 데이터프레임 df의 특정 원소를 변경하는 방법: '서준'의 '체육' 점수
19  df.iloc[0][3] = 80
20  print(df)
21  print('\n')
22
23  df.loc['서준']['체육'] = 90
24  print(df)
25  print('\n')
26
27  df.loc['서준', '체육'] = 100
28  print(df)
```

〈실행 결과〉 코드 1~28라인을 부분 실행

	수학	영어	음악	체육
이름				
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

	수학	영어	음악	체육
이름				
서준	90	98	85	80
우현	80	89	95	90
인아	70	95	100	90

	수학	영어	음악	체육
이름				
서준	90	98	85	90
우현	80	89	95	90
인아	70	95	100	90

	수학	영어	음악	체육
이름				
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

앞의 예제에서 '서준' 학생의 '체육' 점수를 선택하는 여러 방법을 시도하였다. 각 방법을 비교하기 위해, 각기 다른 점수를 새로운 값으로 입력하여 원소를 변경하였다. (변경된 값을 원으로 표시)

# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 원소값 변경 (계속)

#### ② 1개 이상의 원소를 변경

〈예제 1-14〉 원소 값 변경 (File: example/part1/1.14\_modify\_dataframe\_element.py(이어서 계속))

~ ~ ~ (생략) ~ ~ ~

```
31 # 데이터프레임 df의 원소 여러 개를 변경하는 방법: '서준'의 '음악', '체육' 점수
32 df.loc['서준', ['음악', '체육']] = 50
33 print(df)
34 print('\n')
35
36 df.loc['서준', ['음악', '체육']] = 100, 50
37 print(df)
```

〈실행 결과〉 코드 31~37라인을 부분 실행

	수학	영어	음악	체육
이름				
서준	90	98	50	50
우현	80	89	95	90
인아	70	95	100	90

	수학	영어	음악	체육
이름				
서준	90	98	100	50
우현	80	89	95	90
인아	70	95	100	90

### • 행, 열의 위치 바꾸기

데이터프레임의 행과 열을 서로 맞바꾸는 방법이다. 전치의 결과로 새로운 객체를 반환하므로, 기존 객체를 변경하기 위해서는 `df = df.transpose()` 또는 `df = df.T` 와 같이 입력한다.

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



[그림 1-15] 행, 열 바꾸기

### 예제 1-15

〈예제 1-15〉 행, 열 바꾸기

(File: example/part1/1.15\_transpose.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6 exam_data = {'이름' : ['서준', '우현', '인아'],
7              '수학' : [ 90, 80, 70],
8              '영어' : [ 98, 89, 95],
9              '음악' : [ 85, 95, 100],
10             '체육' : [ 100, 90, 90]}
11 df = pd.DataFrame(exam_data)
12 print(df)
13 print('\n')
14
15 # 데이터프레임 df를 전치하기(메소드 활용)
16 df = df.transpose()
17 print(df)
18 print('\n')
19
20 # 데이터프레임 df를 다시 전치하기(클래스 속성 활용)
21 df = df.T
22 print(df)
```



# Part 1. 판다스 입문

## 2-2. 데이터프레임

### • 행, 열의 위치 바꾸기 (계속)

〈실행 결과〉 코드 전부 실행

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

	0	1	2
이름	서준	우현	인아
수학	90	80	70
영어	98	89	95
음악	85	95	100
체육	100	90	90

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

# Part 1. 판다스 입문

## 3. 인덱스 활용

### • 특정 열을 행 인덱스로 설정

: set\_index() 메소드를 사용하여 데이터프레임의 특정 열을 행 인덱스로 설정한다. 새로운 객체를 반환한다.

특정 열을 행 인덱스로 설정: DataFrame 객체.set\_index( [ '열 이름' ] 또는 '열 이름' )



[그림 1-16] 데이터프레임의 특정 열을 행 인덱스로 설정

#### <예제 1-16> 특정 열을 행 인덱스로 설정

(File: example/part1/1.16\_set\_index.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
6 exam_data = {'이름' : [ '서준', '우현', '인아'],
7              '수학' : [ 90, 80, 70],
8              '영어' : [ 98, 89, 95],
9              '음악' : [ 85, 95, 100],
10             '체육' : [ 100, 90, 90]}
11 df = pd.DataFrame(exam_data)
12 print(df)
```

```
13 print('\n')
14
15 # 특정 열(column)을 데이터프레임의 행 인덱스(index)로 설정
16 ndf = df.set_index(['이름'])
17 print(ndf)
18 print('\n')
19 ndf2 = ndf.set_index('음악')
20 print(ndf2)
21 print('\n')
22 ndf3 = ndf.set_index(['수학', '음악'])
23 print(ndf3)
```

<실행 결과> 코드 전부 실행

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

	수학	영어	음악	체육
이름				
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

	수학	영어	체육
음악			
85	90	98	100
95	80	89	90
100	70	95	90

	수학	음악
영어		
90	85	98
80	95	89
70	100	95

※ 이때, 기존 행 인덱스는 삭제된다.

# Part 1. 판다스 입문

## 3. 인덱스 활용

### • 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 추가 되는 경우, 그 행의 데이터 값은 NaN 값이 입력된다.

〈예제 1-17〉 새로운 배열로 행 인덱스 재지정

(File: example/part1/1.17\_reindex.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 딕셔너리 정의
6 dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
7
8 # 딕셔너리를 데이터프레임으로 변환. 인덱스를 [r0, r1, r2]로 지정
9 df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
10 print(df)
11 print('\n')
12
13 # 인덱스를 [r0, r1, r2, r3, r4]로 재지정
14 new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
15 ndf = df.reindex(new_index)
16 print(ndf)
17 print('\n')
18
```

```
19 # reindex로 발생한 NaN값을 숫자 0으로 채우기
20 new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
21 ndf2 = df.reindex(new_index, fill_value=0)
22 print(ndf2)
```

〈실행 결과〉 코드 전부 실행

	c0	c1	c2	c3	c4
r0	1	4	7	10	13
r1	2	5	8	11	14
r2	3	6	9	12	15

	c0	c1	c2	c3	c4
r0	1.0	4.0	7.0	10.0	13.0
r1	2.0	5.0	8.0	11.0	14.0
r2	3.0	6.0	9.0	12.0	15.0
r3	NaN	NaN	NaN	NaN	NaN
r4	NaN	NaN	NaN	NaN	NaN

	c0	c1	c2	c3	c4
r0	1	4	7	10	13
r1	2	5	8	11	14
r2	3	6	9	12	15
r3	0	0	0	0	0
r4	0	0	0	0	0

예제의 12행에서 새롭게 추가된 'r3', 'r4' 인덱스에 해당하는 모든 열에 대해 NaN 값이 입력된다. 이럴 경우, 데이터가 존재하지 않는다는 뜻의 NaN 대신 유효한 값으로 채우려면 예제의 21행과 같이 fill\_value 옵션에 원하는 값(0)을 입력한다.

NaN은 “Not a Number” 라는 뜻이다. 유효한 값이 존재하지 않는 누락 데이터를 말한다.

# Part 1. 판다스 입문

## 3. 인덱스 활용

### • 행 인덱스 초기화

- reset\_index() 메소드로 정수형 위치 인덱스로 초기화. 기존 행 인덱스는 열로 이동. 새로운 데이터프레임 객체를 반환.

정수형 위치 인덱스로 초기화: DataFrame 객체.reset\_index( )

〈예제 1-18〉 정수형 위치 인덱스로 초기화

(File: example/part1/1.18\_reset\_index.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 딕셔너리 정의
6 dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
7
8 # 딕셔너리를 데이터프레임으로 변환. 인덱스를 [r0, r1, r2]로 지정
9 df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
10 print(df)
11 print('\n')
12
13 # 행 인덱스를 정수형으로 초기화
14 ndf = df.reset_index()
15 print(ndf)
```

〈실행 결과〉 코드 전부 실행

	c0	c1	c2	c3	c4
r0	1	4	7	10	13
r1	2	5	8	11	14
r2	3	6	9	12	15

index	c0	c1	c2	c3	c4	
0	r0	1	4	7	10	13
1	r1	2	5	8	11	14
2	r2	3	6	9	12	15

### • 행 인덱스를 기준으로 데이터프레임 정렬

- sort\_index() 메소드로 행 인덱스를 기준으로 정렬.
- ascending 옵션을 사용하여 오름차순 또는 내림차순 설정.
- 새롭게 정렬된 데이터프레임 객체를 반환.

행 인덱스 기준 정렬: DataFrame 객체.sort\_index( )

〈예제 1-19〉 데이터프레임 정렬

(File: example/part1/1.19\_sort\_index.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 딕셔너리 정의
6 dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
7
8 # 딕셔너리를 데이터프레임으로 변환. 인덱스를 [r0, r1, r2]로 지정
9 df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
10 print(df)
11 print('\n')
12
13 # 내림차순으로 행 인덱스 정렬
14 ndf = df.sort_index(ascending=False)
15 print(ndf)
```

〈실행 결과〉 코드 전부 실행

	c0	c1	c2	c3	c4
r0	1	4	7	10	13
r1	2	5	8	11	14
r2	3	6	9	12	15

	c0	c1	c2	c3	c4
r2	3	6	9	12	15
r1	2	5	8	11	14
r0	1	4	7	10	13

# Part 1. 판다스 입문

## 4. 산술연산

- 판다스 객체의 산술연산은 내부적으로 **3단계 프로세스**를 거친다.
  - ① 행/열 인덱스를 기준으로 모든 원소를 정렬한다.
  - ② 동일한 위치에 있는 원소끼리 일대일로 대응시킨다.
  - ③ 일대일 대응이 되는 원소끼리 연산을 처리한다. 이때, 대응되는 원소가 없으면 NaN으로 처리한다.

### 4-1. 시리즈 연산

- 시리즈 vs. 숫자

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

〈예제 1-21〉 시리즈를 숫자로 나누기

(File: example/part1/1\_21\_series\_to\_number.py)

```
1 # -*- coding: utf-8 -*-
2
3 # 라이브러리 불러오기
4 import pandas as pd
5
6 # 딕셔너리 데이터로 판다스 시리즈 만들기
7 student1 = pd.Series({'국어':100, '영어':80, '수학':90})
8 print(student1)
9 print('\n')
10
11 # 학생의 과목별 점수를 200으로 나누기
12 percentage = student1/200
13
14 print(percentage)
15 print('\n')
16 print(type(percentage))
```

〈실행 결과〉 코드 전부 실행

```
국어    100
영어     80
수학     90
dtype: int64
```

```
국어    0.50
영어    0.40
수학    0.45
dtype: float64
```

```
<class 'pandas.core.series.Series'>
```

시리즈 객체에 어떤 숫자를 더하면, 시리즈의 개별 원소에 각각 숫자를 더하고 계산한 결과를 시리즈 객체로 반환한다. 덧셈, 뺄셈, 곱셈, 나눗셈 모두 가능하다. 앞의 예제에서는 시리즈 객체의 각 원소를 200으로 나누는 과정을 살펴본다.

# Part 1. 판다스 입문

## 4-1. 시리즈 연산

### • 시리즈 vs. 시리즈

- 시리즈와 시리즈 사이에 사칙연산 처리.
- 같은 인덱스를 가진 원소끼리 계산.
- 해당 인덱스에 연산 결과를 매칭하여 새 시리즈를 반환.

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

〈예제 1-22〉 시리즈 사칙연산

(File: example/part1/1.22\_series\_to\_series.py)

```
1  -*- coding: utf-8 -*-
2
3  # 라이브러리 불러오기
4  import pandas as pd
5
6  # 딕셔너리 데이터로 판다스 시리즈 만들기
7  student1 = pd.Series({'국어':100, '영어':80, '수학':90})
8  student2 = pd.Series({'수학':80, '국어':90, '영어':80})
9
10 print(student1)
11 print('\n')
12 print(student2)
13 print('\n')
14
15 # 두 학생의 과목별 점수로 사칙연산 수행
16 addition = student1 + student2          # 덧셈
17 subtraction = student1 - student2       # 뺄셈
18 multiplication = student1 * student2    # 곱셈
19 division = student1/student2            # 나눗셈
20 print(type(division))
21 print('\n')
22
23 # 사칙연산 결과를 데이터프레임으로 합치기(시리즈 -> 데이터프레임)
24 result = pd.DataFrame([addition, subtraction, multiplication, division],
25                        index=['덧셈', '뺄셈', '곱셈', '나눗셈'])
26 print(result)
```

〈실행 결과〉 코드 전부 실행

```
국어    100
영어     80
수학     90
dtype: int64
```

```
수학     80
국어     90
영어     80
dtype: int64
```

<class 'pandas.core.series.Series'>

	국어	수학	영어
덧셈	190.000000	170.000	160.0
뺄셈	10.000000	10.000	0.0
곱셈	9000.000000	7200.000	6400.0
나눗셈	1.111111	1.125	1.0



[그림 1-17] 시리즈의 산술연산(덧셈)



# Part 1. 판다스 입문

## 4-1. 시리즈 연산

### • 시리즈 vs. 시리즈 (계속)

- 두 시리즈의 원소 개수가 다르거나, 혹은 시리즈의 크기가 같더라도 인덱스 값이 다를 수 있다. 이런 경우, 유효한 값이 존재하지 않는다는 의미로 NaN으로 처리한다.
- 한편, 같은 인덱스가 양쪽에 모두 존재하여 서로 대응되어도 어느 한 쪽의 데이터 값이 NaN인 경우가 있다. 이때, 연산의 대상인 데이터가 존재하지 않기 때문에, NaN으로 처리한다.

sr1		+	sr2		=	sr3	
index	value		index	value		index	value
A	10		?	?		A	NaN
B	20		B	10		B	30
C	30		C	20		C	50
D	NaN		D	30		D	NaN
?	?		E	40		E	NaN

[그림 1-18] NaN값이 있는 시리즈의 산술연산(덧셈)

<예제 1-23> NaN값이 있는 시리즈 연산

(File: example/part1/1.23\_series\_to\_series2.py)

```
1 # -*- coding: utf-8 -*-
2
3 # 라이브러리 불러오기
4 import pandas as pd
5 import numpy as np
6
7 # 딕셔너리 데이터로 판다스 시리즈 만들기
8 student1 = pd.Series({'국어':np.nan, '영어':80, '수학':90})
9 student2 = pd.Series({'수학':80, '국어':90})
10
```

```
11 print(student1)
12 print('\n')
13 print(student2)
14 print('\n')
15
16 # 두 학생의 과목별 점수로 사칙연산 수행 (시리즈 vs 시리즈)
17 addition = student1 + student2          # 덧셈
18 subtraction = student1 - student2      # 뺄셈
19 multiplication = student1 * student2   # 곱셈
20 division = student1/student2           # 나눗셈
21 print(type(division))
22 print('\n')
23
24 # 사칙연산 결과를 데이터프레임으로 합치기 (시리즈 -> 데이터프레임)
25 result = pd.DataFrame([addition, subtraction, multiplication, division],
26                        index=['덧셈', '뺄셈', '곱셈', '나눗셈'])
27 print(result)
```

<실행 결과> 코드 전부 실행

```
국어      NaN
영어      80.0
수학      90.0
dtype: float64

수학      80
국어      90
dtype: int64

<class 'pandas.core.series.Series'>

      국어   수학   영어
덧셈   NaN  170.000   NaN
뺄셈   NaN   10.000   NaN
곱셈   NaN  7200.000   NaN
나눗셈  NaN    1.125   NaN
```



# Part 1. 판다스 입문

## 4-1. 시리즈 연산

### • 연산 메소드 활용

- 객체 사이에 공통 인덱스가 없는 경우에 NaN으로 반환.
- 이런 상황을 피하려면 연산 메소드에 fill\_value 옵션을 설정.

※ 다음 예제는 누락 데이터 NaN 대신 숫자 0을 입력하는 것을 예시로 설명.

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

〈예제 1-24〉 연산 메소드 사용 – 시리즈 연산

(File: example/part1/1.24\_series\_to\_series3.py)

```
1 # -*- coding: utf-8 -*-
2
3 # 라이브러리 불러오기
4 import pandas as pd
5 import numpy as np
6
7 # 딕셔너리 데이터로 판다스 시리즈 만들기
8 student1 = pd.Series({'국어':np.nan, '영어':80, '수학':90})
9 student2 = pd.Series({'수학':80, '국어':90})
10
11 print(student1)
12 print('\n')
13 print(student2)
14 print('\n')
15
16 # 두 학생의 과목별 점수로 사칙연산 수행 (연산 메소드 사용)
17 sr_add = student1.add(student2, fill_value=0) # 덧셈
18 sr_sub = student1.sub(student2, fill_value=0) # 뺄셈
19 sr_mul = student1.mul(student2, fill_value=0) # 곱셈
20 sr_div = student1.div(student2, fill_value=0) # 나눗셈
21
22 # 사칙연산 결과를 데이터프레임으로 합치기 (시리즈 -> 데이터프레임)
23 result = pd.DataFrame([sr_add, sr_sub, sr_mul, sr_div],
24                        index=['덧셈', '뺄셈', '곱셈', '나눗셈'])
25 print(result)
```

〈실행 결과〉 코드 전부 실행

국어	NaN		
영어	80.0		
수학	90.0		
dtype:	float64		
수학	80		
국어	90		
dtype:	int64		
	국어	수학	영어
덧셈	90.0	170.000	80.000000
뺄셈	-90.0	10.000	80.000000
곱셈	0.0	7200.000	0.000000
나눗셈	0.0	1.125	inf

연산 메소드에 fill\_value=0 옵션을 설정하여 student1의 국어 점수와 student2의 영어 점수는 NaN 대신 0으로 입력된다. 한편 result의 '영어' 열의 나눗셈 값은 무한대(inf)이다. student1의 영어 점수 80을 student2의 영어 점수 0으로 나눈 값이기 때문이다.

# Part 1. 판다스 입문

## 4-2. 데이터프레임 연산

데이터프레임은 여러 시리즈가 한데 모인 것이므로, 시리즈 연산을 확장하는 개념으로 이해한다. 먼저 행/열 인덱스를 기준으로 정렬하고, 일대일 대응되는 원소끼리 연산한다.

### • 데이터프레임 vs. 숫자

- 데이터프레임에 어떤 숫자를 더하면, 모든 원소에 숫자를 더한다. 덧셈, 뺄셈, 곱셈, 나눗셈 모두 가능하다.
- 기존 데이터프레임의 형태를 그대로 유지한 채, 원소 값만 새로운 값으로 바뀐다.

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

〈예제 1-25〉 데이터프레임에 숫자 더하기

(File: example/part1/1.25\_df\_to\_number.py)

```
1  # -*- coding: utf-8 -*-
2
3  # 라이브러리 불러오기
4  import pandas as pd
5  import seaborn as sns
6
7  # titanic 데이터셋에서 age, fare 2개 열을 선택하여 데이터프레임 만들기
8  titanic = sns.load_dataset('titanic')
9  df = titanic.loc[:, ['age', 'fare']]
10 print(df.head())          # 첫 5행만 표시
11 print('\n')
12 print(type(df))
13 print('\n')
14
15 # 데이터프레임에 숫자 10 더하기
16 addition = df + 10
17 print(addition.head())    # 첫 5행만 표시
18 print('\n')
19 print(type(addition))
```

〈실행 결과〉 코드 전부 실행

```
   age  fare
0  22.0  7.2500
1  38.0 71.2833
2  26.0  7.9250
3  35.0 53.1000
4  35.0  8.0500
```

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

```
   age  fare
0  32.0 17.2500
1  48.0 81.2833
2  36.0 17.9250
3  45.0 63.1000
4  45.0 18.0500
```

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

### ※ 타이타닉('titanic') 데이터셋

Seaborn 라이브러리에서 제공하는 데이터셋으로, 타이타닉호 탑승자에 대한 인적사항과 구조 여부 등을 정리한 자료이다.

`load_dataset()` 함수로 불러온다.

[Seaborn 내장 데이터셋의 종류]

'anscombe', 'attention', 'brain\_networks', 'car\_crashes',  
'diamonds', 'dots', 'exercise', 'flights', 'fmri', 'gammas', 'iris',  
'mpg', 'planets', 'tips', 'titanic'

# Part 1. 판다스 입문

## 4-2. 데이터프레임 연산

### • 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`

df1					df2				df1+df2			
	열 A	열 B	열 C		열 A	열 B			열 A	열 B	열 C	
행 0	15	100	1	+	행 0	10	200	=	행 0	25	300	NaN
행 1	20	500	2		행 1	10	100		행 1	30	600	NaN
행 2	50	200	3		행 2	10	200		행 2	60	400	NaN
행 3	NaN	500	2		행 3	20	100		행 3	NaN	600	NaN
행 4	NaN	200	3		행 4	30	100		행 4	NaN	300	NaN

[그림 1-19] 데이터프레임의 산술연산(덧셈)

<예제 1-26> 데이터프레임끼리 더하기

(File: example/part1/1.26\_df\_to\_df.py)

```
1 # -*- coding: utf-8 -*-
2
3 # 라이브러리 불러오기
4 import pandas as pd
5 import seaborn as sns
6
7 # titanic 데이터셋에서 age, fare 2개 열을 선택하여 데이터프레임 만들기
8 titanic = sns.load_dataset('titanic')
9 df = titanic.loc[:, ['age', 'fare']]
10 print(df.tail())          # 마지막 5행 표시
11 print('\n')
12 print(type(df))
```

```
15 # 데이터프레임에 숫자 10 더하기
16 addition = df + 10
17 print(addition.tail())      # 마지막 5행 표시
18 print('\n')
19 print(type(addition))
20 print('\n')
21
22 # 데이터프레임끼리 연산하기 (addition - df)
23 subtraction = addition - df
24 print(subtraction.tail())   # 마지막 5행 표시
25 print('\n')
26 print(type(subtraction))
```

<실행 결과> 코드 전부 실행

```
   age  fare
886  27.0  13.00
887  19.0  30.00
888   NaN  23.45
889  26.0  30.00
890  32.0   7.75

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

   age  fare
886  37.0  23.00
887  29.0  40.00
888   NaN  33.45
889  36.0  40.00
890  42.0  17.75

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

   age  fare
886  10.0  10.0
887  10.0  10.0
888   NaN  10.0
889  10.0  10.0
890  10.0  10.0

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