

파이썬 라이브러리(Python library)

- 책이 여러 권 있는 도서관에서 원하는 책을 찾아 읽을 수 있듯이, 프로그래밍에서는 라이브러리를 통해 이미 만들어져 있는 소스코드를 가져와 사용할 수 있는 도구들의 모음
- 코드를 통해 직접 구현하지 않아도 되기 때문에 많은 시간을 절약할 수 있다.

데이터 분석에 가장 널리 사용되는 라이브러리들

- numpy(넘파이), pandas(판다스), Matplotlib(맷플롯립),seaborn(시본), Scikit-learn(사이킷 런) 등..
- 이러한 다양한 데이터 분석 라이브러리들의 용도와 사용법에 대해 배울 것

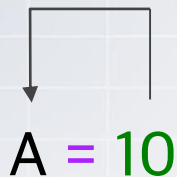
함수(function)란?

- 함수는 특정한 작업을 위해 재활용할 수 있도록 구현한 코드 블록
- 코드의 재사용성을 높여준다.

※ 예를 들어 print 함수는 모니터에 내용을 출력해주기 위해 설계된 코드의 집합

변수(variable)란?

- 변수는 값을 저장하는 메모리 공간
- 쉽게 말하면 무엇을 담는 그릇이라 생각할 수 있음.
- =을 기준으로 오른쪽에 위치한 값을 왼쪽에 대입
- 변수의 이름은 자기 마음대로 지어도 됨



A diagram illustrating variable assignment. A horizontal line with a downward-pointing arrow on the left and an upward-pointing arrow on the right is positioned above the text 'A = 10'. The text 'A' is in black, '=' is in purple, and '10' is in green.

변수 이름의 규칙

- 첫 글자는 영문, 혹은 언더바(_)로 시작
- 첫 글자 외의 나머지 글자는 영문,숫자,언더바 모두 사용 가능하며 언더바를 제외한 특수기호는 사용 불가능

예)

1. name - (○)
2. _name - (○)
3. 1name - (X) → error
4. name1 - (○)
5. ^name - (X) → error

변수 이름의 규칙

- 대문자, 소문자를 구분한다.
- 길이에 대한 제약은 없지만, 짧고 간단할수록 좋다

예)

1. name
 2. Name
- 서로 다른 변수

변수 이름의 규칙

- 이미 파이썬 문법에 특별한 목적을 위해 예약되어 사용되는 단어들을 예약어라고 부른다.
- 문법적인 기능을 수행하고 있기 때문에 변수명으로 사용할 수 없도록 되어있다.

True	False	None	if	elif
continue	def	finally	else	for
pass	while	with	try	except
break	class	return	import	as

“Hello, World!” 입력하기

- **print 함수**를 사용하면 모니터에 내용을 출력해준다.
- 내용을 출력할 땐 괄호()와 함께 print 함수가 기대하는 형태로 전달해야 된다.
- 텍스트는 출력하고자 할 때에는 꼭 큰 따옴표(“”) 혹은 작은 따옴표(”)를 통해 텍스트를 감싸줘야 한다.
- 따옴표를 사용하지 않으면 파이썬은 해당 부분을 텍스트가 아닌 어떤 **변수**로 인식하려고 한다.
- 한번에 많은 내용을 담고 싶다면, **쉼표(,)**를 사용하여 구분해준다.

```
In [1]: print("Hello, World!")
```

```
Hello, World!
```

```
In [1]: A = 10
```

```
In [2]: print("Hello, World!", A)
```

```
Hello, World! 10
```

변수를 만들어 값을 저장하기

- 변수에 값을 저장하지 않고 print함수만을 사용하면, 1회성으로 한 번 만 출력되고 사라진다.
- 변수에 값을 저장하면 긴 내용을 여러 번 타이핑하지 않아도 쉽게 출력이 가능하다.
- 변수 선언된 값을 확인하기 위해 print()함수를 사용한다.

```
In [1]: var = 100
```

```
In [2]: print(var)
```

```
100
```

```
In [3]: var1 = "Hello, World!"
```

```
In [5]: print(var1)
```

```
Hello, World!
```

변수와 사칙연산

- 산술을 위해 사용할 수 있는 연산 기호들은 많다. 이 중 산수의 기본이 되는 4가지 연산을 사칙연산이라 한다.
- 사칙연산에는 더하기(+), 빼기(-), 곱하기(*), 나누기(/)가 있다.
- 파이썬에는 이러한 연산 기능을 지원한다. 사칙연산을 해본 후 값을 변수에 저장해보기.

연산자	사용 예	설명
+	<code>num = 4+3</code>	4와 3을 더한 값을 num에 대입
-	<code>num = 4-3</code>	4와 3을 뺀 값을 num에 대입
*	<code>num = 4*3</code>	4와 3을 곱한 값을 num에 대입
/	<code>num = 4/3</code>	4를 3으로 나눈 값을 num에 대입

변수를 사용한 연산

- 사칙연산을 진행한 값을 변수에 담아 두 개의 변수를 만든다.
- 만들어진 두 개의 변수를 사용하여 연산 후 새로운 변수에 담아 본다.
- `print()`를 사용해 변수와 문자열을 포함시켜 사용하여 $a + b = c$ 를 출력해본다

```
In [7]: a = 100*10  
        b = 200*50
```



```
In [9]: c = a + b  
  
In [10]: print(a, "+", b, "=", c)  
         1000 + 250 = 1250
```

Pandas (판다스)

- pandas는 데이터 조작 및 분석을 위한 파이썬 프로그래밍 언어에서 사용되는 소프트웨어 라이브러리
- 금융, 신경과학, 경제학, 통계학, 광고, 웹 분석 등 다양한 학문과 사업 분야에 사용
- 숫자 테이블과 시계열을 조작하기 위한 데이터 구조와 연산을 제공
- 구조화된 데이터나 표 형식의 데이터를 빠르고 쉽게 표현하도록 설계된 고수준 자료구조 함수를 제공



Pandas (판다스)

- 웨스 맥키니(Wes McKinney)가 2008년에 개발 시작하여 2009년에 오픈 소스로 공개
- 현재 수많은 단체와 기여자들이 pandas를 사용하여 개발하고 지지
- 원래 금융부분 중 특히 시계열(time series) 데이터의 조작과 주가 정보 처리를 염두에 두고 개발
- 현재는 통계, 데이터과학, 머신러닝 등 분야에서 가장 중요한 소프트웨어로 성장
- 데이터 분석을 위한 검색,인덱스,정제,정돈,재형성,조합,슬라이싱 등 기능이 있는 유용한 도구

Pandas (판다스)

- DataFrame(데이터프레임)이라는 행과 열로 구성된 데이터 구조를 table(표) 처럼 사용할 수 있다.
- 쉽표로 구분된 값들, 예로 JSON, SQL, Excel 등 과 같은 다양한 파일 형식에서 데이터를 가져올 수 있다.
- 각종 데이터 처리 동작을 허용하며, 정렬, 병합, 결합, 필터링 등 다양한 기능을 사용할 수 있다.
- 데이터 분석은 엑셀과 비슷하게 표 데이터를 많이 다룬다. 표 데이터는 행과 열로 이루어진 데이터이다.
- 가로방향으로 나열되어 있는 건 행, 세로 방향으로 나열되어 있는 것은 열이다.

Pandas (판다스)

The diagram illustrates the structure of a Pandas DataFrame using a table. The table has 9 columns: an empty header cell, '이름', '번호', '반', '국어', '영어', '수학', '과학', and '지리'. It has 3 rows: an empty header row, and rows indexed 1, 2, and 3. Annotations include: a green dashed box around the '반' column with an arrow pointing to the text '열(column, 세로, 하나의 항목)'; a red dashed box around the row indexed 2 with an arrow pointing to the text '행(row, 가로, 한 건의 데이터)'; and a blue dashed box around the cell at row 2, column '반' with an arrow pointing to the text '요소(element, 한 칸)'. The cell at row 2, column '반' is shaded with blue diagonal lines.

	이름	번호	반	국어	영어	수학	과학	지리
1								
2								
3								

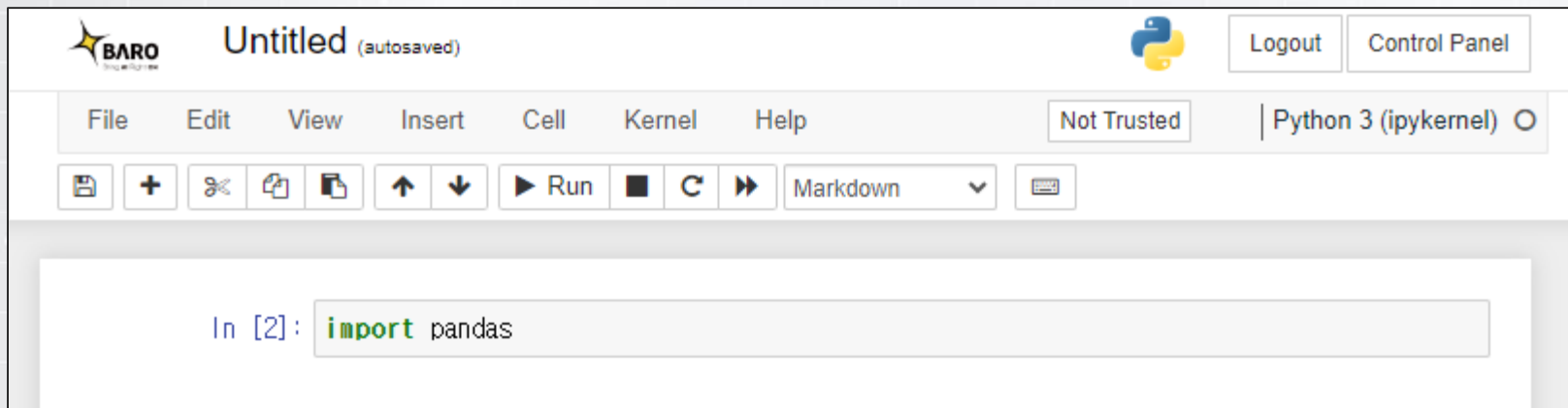
Pandas (판다스)

- 표 데이터 맨 위에는 대부분 항목 명이 나열되어 있어서 열이 무슨 항목인지 나타낸다.
- 표 데이터의 맨 왼쪽에는 대부분 번호가 나열되어 있어서 행이 몇 번째 데이터인지 나타낸다.

The diagram illustrates a Pandas DataFrame structure. It features a table with 9 columns and 4 rows. The first row is the header row, containing the column names: '이름', '번호', '반', '국어', '영어', '수학', '과학', and '지리'. This row is enclosed in a dashed purple border, and a purple arrow points to it from the label 'header(헤더, 항목명)'. The first column contains the row indices: '1', '2', and '3'. This column is enclosed in a dashed blue border, and a blue arrow points to it from the label 'Index(색인, 인덱스)'. The remaining 8 columns and 3 rows form the data area.

	이름	번호	반	국어	영어	수학	과학	지리
1								
2								
3								

Pandas (판다스) 불러오기

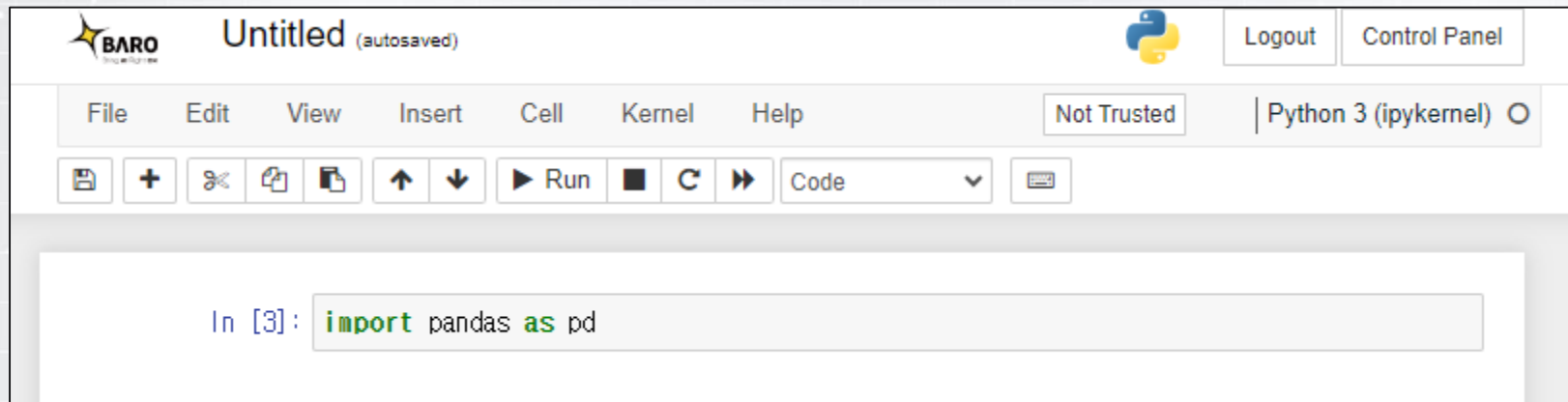


•import는 파이썬에서 다른 모듈이나 패키지를 현재 스크립트나 프로그램에서 사용할 수 있도록 가져오는 역할

※사용법

- `import` 모듈 /패키지 이름
- `import` 모듈 / 패키지 이름 `as` 별칭 ← 모듈 이름이 너무 길거나, 여러 번 사용해야 할 때 간단한 별칭을 사용

Pandas (판다스) 불러오기



- Pandas의 경우에는 import 할 때, pd라는 별칭을 사용하는 것이 일반적인 관례

※ 현재 Python 환경에 Pandas 라이브러리가 설치되어 있지 않을 때 발생하는 오류

```
In [1]: import pandas as pd
```

-
ModuleNotFoundError

Traceback (most recent call last)

Cell In[1], line 1

----> 1 import pandas as pd

ModuleNotFoundError: No module named 'pandas'



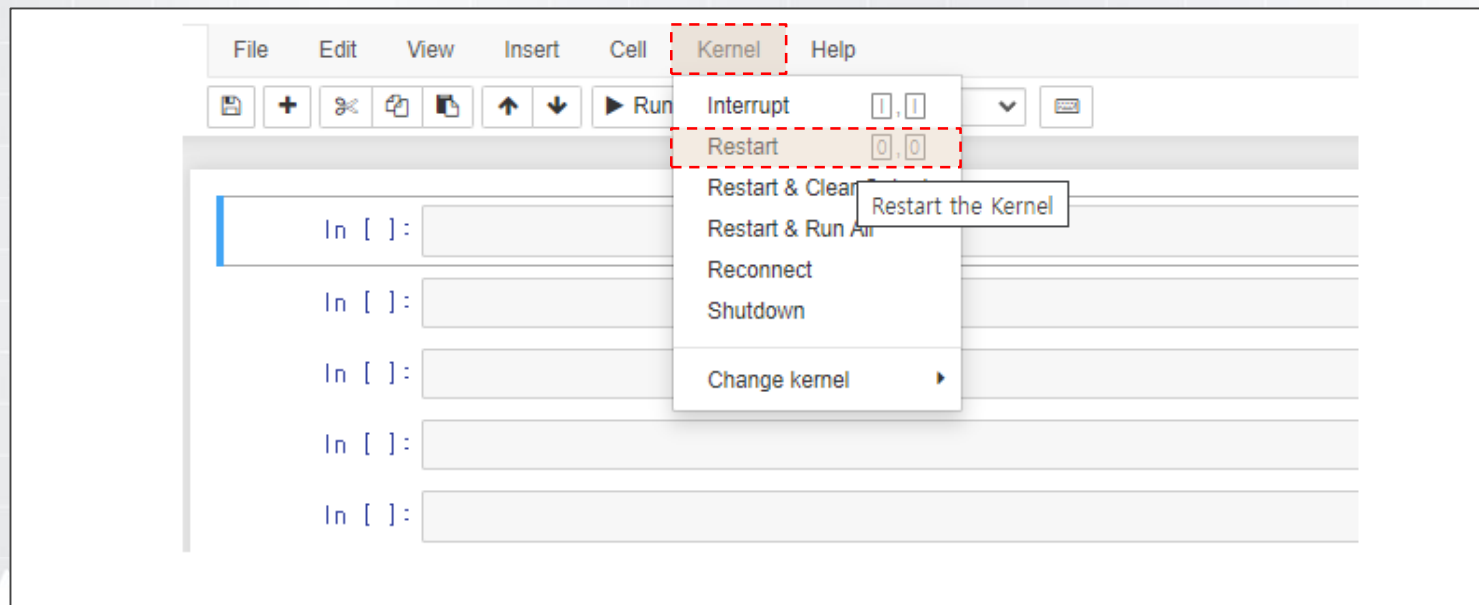
다시 import 하여 올바르게 설치되었는지 확인

※ "pip install pandas" 명령을 사용하면 Pandas 라이브러리를 설치할 수 있다.

```
In [2]: pip install pandas
```

"pip install"은 Python 패키지를 설치하기 위한 명령어

※ 여전히 import가 안된다면 패키지 설치 후 커널 재시작(restart)



Pandas(판다스) 이해하기

- Pandas에 대해 알기 위해서는 series(시리즈), dataframe(데이터프레임) 사용법에 익숙해질 필요가 있다.
- Pandas 에서 Series와 DataFrame은 데이터 조작 및 분석을 위한 핵심적인 자료구조.
- 특정한 상황과 요구에 맞게 데이터를 다루고 분석하는 데 사용

Series

Series 이해하기

- Pandas의 기본 데이터 구조
- Series는 일련의 객체를 담을 수 있는 1차원 배열 같은 자료 구조를 말한다.
- Index(색인, 인덱스)를 붙일 수 있으며, 데이터 자체는 value(값)이라고 표현한다.
- 하나의 열로만 구성된 테이블로, 각 인덱스 당 하나의 값만 가질 수 있다.

시리즈(series) = 인덱스(index) + 값(value)

Series 생성 방법(1)

- my_s 라는 변수를 만들어 series로 100,200,300,400의 값을 가진 구조를 만들어주기

```
In [6]: import pandas as pd  
my_s = pd.Series([100,200,300,400])
```

my_s 라는 변수(variable)를 만들어 오른쪽에 있는 값을 저장하고 다룰 수 있도록 해주는 것

Series 생성 방법(1)

- my_s 라는 변수를 만들어 series로 100,200,300,400의 값을 가진 구조를 만들어주기

```
In [6]: import pandas as pd  
my_s = pd.Series([100,200,300,400])
```




pd.Series는 pandas에서 series라는 기능을 가져와서 사용하겠다는 의미

Series 생성 방법(1)

- my_s 라는 변수를 만들어 series로 100,200,300,400의 값을 가진 구조를 만들어주기

```
In [6]: import pandas as pd  
  
my_s = pd.Series([100,200,300,400])
```



100, 200, 300, 400의 값을 가진 series를 만들어 주는 것

- 대괄호[] : 데이터를 리스트로 만들어주기 위해 사용
- 소괄호() : 함수 호출에 필요한 인자(argument)를 전달하기 위해 사용

※ series가 리스트만 받을 수 있는 것은 아니지만, 리스트는 수정가능한 자료형이기에 여러 개의 값들을 쉽게 다룰 수 있다.

Series 이해하기

- my_s 를 출력하여 4개의 값을 가지는 시리즈를 확인하기
- 인덱스에 해당하는 인덱스의 값을 확인하기
- 인덱스를 따로 지정해주지 않아 0부터 n-1개의 숫자가 자동으로 부여되어 표시된다

```
In [7]: my_s
Out[7]: 0    100
        1    200
        2    300
        3    400
        dtype: int64
```

0	100
1	200
2	300
3	400

dtype: int64
64비트 정수 데이터타입 이라는 것을 의미

인덱스(index) : 0, 1, 2, 3,
값(value) : 100, 200, 300, 400

Series index 설정

- 인덱스도 내가 원하는 값을 부여하고 싶을 때
- pd.Series() ← 괄호 안에 값을 적어준 다음, index라는 pandas의 기능을 사용하여 각 값에 해당하는 인덱스를 지정해주기

```
In [14]: my_s1 = pd.Series([100,200,300,400], index=['a','b','c','d'])
```

```
In [15]: my_s1
```

```
Out[15]: a    100  
         b    200  
         c    300  
         d    400  
         dtype: int64
```

Series 확인하기

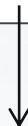
- 인덱스는 .index 속성을 사용하여 접근하여 확인할 수 있다.
- 값은 .values 속성을 사용하여 접근하여 확인할 수 있다.

```
In [16]: my_sl.index
```

```
Out[16]: Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
In [17]: my_sl.values
```

```
Out[17]: array([100, 200, 300, 400])
```



array는 배열값을 의미하며, 여러 개의 값을 하나의 변수에 저장하고 관리하기 위한 자료 구조를 말한다.

Series 접근하기

- 배열(array)은 여러 개의 요소(element)로 구성된다.
- 각 요소는 고유한 인덱스를 가지기 때문에, 해당 인덱스를 사용하여 접근할 수 있습니다.

```
In [17]: my_s1.values
```

```
Out [17]: array([100, 200, 300, 400])
```

array([100, 200, 300, 400])

배열(array)

요소(element)

Series 접근하기

- 인덱스 값을 사용하여 값을 불러올 수 있다.
- 인덱스 중 하나를 입력하면 해당하는 요소 값을 반환한다.

```
In [18]: my_sl['a']
```

```
Out[18]: 100
```

```
In [19]: my_sl['b']
```

```
Out[19]: 200
```

```
In [20]: my_sl['c']
```

```
Out[20]: 300
```

```
In [40]: my_sl['d']
```

```
Out[40]: 400
```

Series 접근하기

- 숫자 정보를 이용해 값을 불러올 수 있다.

n번째 값

```
In [36]: my_sl[0]
```

```
Out [36]: 100
```

```
In [37]: my_sl[1]
```

```
Out [37]: 200
```

```
In [38]: my_sl[2]
```

```
Out [38]: 300
```

```
In [39]: my_sl[3]
```

```
Out [39]: 400
```

Series 접근하기

- `.iloc[]`와 `.loc[]` 속성을 이용해 명시적으로 검색이 가능하며, 차이점은 데이터를 선택하는 기준
- `.iloc[]` : 정수 위치를 기반으로 데이터를 선택
- `.loc[]` : 인덱스 명을 기반으로 데이터를 선택

```
In [22]: my_sl.iloc[[1,2]]
```

```
Out [22]: b    200  
         c    300  
         dtype: int64
```

```
In [25]: my_sl.loc[['b','c']]
```

```
Out [25]: b    200  
         c    300  
         dtype: int64
```


Series 수정하기

- 인덱스 정보를 사용하여 데이터 값 수정하기

```
In [41]: my_sl['b'] = 5000
```

```
In [42]: my_sl
```

```
Out [42]: a    100  
         b   5000  
         c    300  
         d    400  
         dtype: int64
```



Series 삭제하기 `del`

- `del` 함수와 인덱스 정보를 사용하여 데이터 값을 삭제하기
- `del`을 사용하면 해당하는 열이 원본에서 직접 삭제되어 이전 열을 사용할 수 없게 된다.

```
In [43]: del my_sl['b']
```



```
In [44]: my_sl
```

```
Out[44]: a    100  
         c    300  
         d    400  
         dtype: int64
```

Series name

- 변수.name 속성을 통해 자신의 series의 이름을 붙일 수 있다.

```
In [22]: my_s1.name = "my first series"
```

```
In [23]: my_s1
```

```
Out [23]: a    100  
         b    200  
         c    300  
         d    400  
         Name: my first series, dtype: int64
```

my first series

a	100
b	200
c	300
d	400

Series index name

- 변수.index.name 속성을 통해 자신의 series index에 이름을 붙일 수 있다.

```
In [24]: my_s1.index.name = "label"
```

```
In [25]: my_s1
```

```
Out [25]: label  
a      100  
b      200  
c      300  
d      400  
Name: my first series, dtype: int64
```

my first series

label	
a	100
b	200
c	300
d	400

Series 생성 방법(1)

- my_s 라는 변수를 만들어 series로 100,200,300,400의 값을 가진 구조를 만들기

```
In [6]: import pandas as pd  
  
my_s = pd.Series([100, 200, 300, 400])
```

100, 200, 300, 400의 값을 가진 series를 만들어 주는 것

my_s 라는 변수(variable)를 만들어 오른쪽에 있는 값을 저장하고 다룰 수 있도록 해주는 것

pd.Series는 pandas에서 series라는 기능을 가져와서 사용하겠다는 의미

Series 생성 방법(2)

•이전에 series를 생성했던 방법 외에도 변수를 사용하여 series를 생성할 수 있다.

이전 방법 :

```
In [14]: my_sl = pd.Series([100,200,300,400], index=['a','b','c','d'])
```

값은 data 변수에, 인덱스는 label 변수에



```
In [33]: data = [100, 80, 90, 50]
label = ['국어', '수학', '사회', '과학']
score = pd.Series(data, index=label)
```

```
In [35]: score
```

```
Out [35]: 국어    100
수학     80
사회     90
과학     50
dtype: int64
```

Series 생성 방법 (3)

•이전에 series를 생성했던 방법 외에도 변수를 사용하여 series를 생성할 수 있다.

이전 방법 :

```
In [14]: my_sl = pd.Series([100,200,300,400], index=['a','b','c','d'])
```



딕셔너리(Dictionary)라는 사전형식으로 중괄호{} 사용
키(key)와 값(value)를 쌍으로 저장하는 데이터 구조

```
In [60]: data = {'국어':100, '수학':80, '사회':90, '과학':50}  
score = pd.Series(data)
```

```
In [61]: score
```

```
Out [61]: 국어      100  
수학       80  
사회       90  
과학      50  
dtype: int64
```

※ series 생성과 index 부여 하기

- 어떤 데이터를 pandas series로 가져왔을 때, 데이터의 각 항목에 index를 할당
- 예를 들어, 값만 존재하는 데이터를 가져오게 되어 분석할 때 날짜, 코드 등의 데이터에 필요한 index를 지정

1. Pandas를 가져온다
2. 변수를 만들고, pd.Series를 사용한다.
3. 값을 가지는 하나의 series를 만든다.
4. 인덱스의 이름을 부여해준다.

※ series 생성과 index 부여 하기

- 어떤 데이터를 pandas series로 가져왔을 때, 데이터의 각 항목에 index를 할당
- 예를 들어, 값만 존재하는 데이터를 가져오게 되어 분석할 때 날짜, 코드 등의 데이터에 필요한 index를 지정

Pandas를 가져온 후 series를 만든다.

```
In [1]: import pandas as pd
```

```
In [2]: stock = pd.Series([50,60,70])
```

```
In [3]: stock
```

```
Out [3]: 0    50  
         1    60  
         2    70  
         dtype: int64
```


※ series 생성과 index 부여 하기

인덱스의 이름을 부여해준다.

```
In [1]: import pandas as pd
```

```
In [2]: stock = pd.Series([50, 60, 70])
```

```
In [3]: stock
```

```
Out [3]: 0    50  
         1    60  
         2    70  
         dtype: int64
```

index 부여 →

예) 주식 이름(Index) 과 가지고 있는 주식 수(Value)

```
In [3]: stock = pd.Series([50, 60, 70], index=['naver', 'apple', 'microsoft'])
```

```
In [4]: stock
```

```
Out [4]: naver      50  
         apple     60  
         microsoft  70  
         dtype: int64
```

※ series 생성과 index 부여 하기

예) 회사 이름(Value)과 회사 코드(Index)

```
In [8]: company = ['LG', 'KT', 'SK']  
company_code = ['A001', 'A002', 'A003']  
code = pd.Series(company, index = company_code)
```

```
In [9]: code
```

```
Out [9]: A001    LG  
A002    KT  
A003    SK  
dtype: object
```

※.index와 .values 속성 사용하기

- Pandas series index와 values를 사용하면 특정 데이터에 접근하여 분석과 식별에 도움을 준다.

예) 특정 시간의 주식 가격을 분석하기 위해 날짜와 가격을 확인, 특정 고객과 구매 금액 확인을 위해 접근 등

1. Pandas를 가져온다
2. 변수를 만들고, pd.Series를 사용한다.
3. 값과 인덱스 명을 가지고 있는 series를 만든다.
4. .index, .values 속성을 사용한다.
5. 특정 값에 접근하여 값을 확인한다.

※.index와 .values 속성 사용하기

예) 주식 날짜를 index, 해당 날짜의 마지막 시간의 가격을 value로 갖는 series

```
In [17]: stock_prices = pd.Series([1500, 1510, 1520 ],  
                                index=['2023-09-01', '2023-09-02', '2023-09-03'])
```

```
In [18]: print(stock_prices.index)  
Index(['2023-09-01', '2023-09-02', '2023-09-03'], dtype='object')
```

```
In [19]: print(stock_prices.values)  
[1500 1510 1520]
```

※.index와 .values 속성 사용하기

예) 고객을 index, 고객의 포인트를 value로 갖는 series

```
In [26]: point= [2500, 18000, 3000]
         ID = ['ID_001', 'ID_002', 'ID_003']
         member = pd.Series(point, index=ID)
```

```
In [27]: print(member.index)

Index(['ID_001', 'ID_002', 'ID_003'], dtype='object')
```

```
In [28]: print(member.values)

[ 2500 18000  3000]
```

※ iloc와 loc 속성 사용하여 다수의 시리즈 값 불러오기

- 특정 범위의 데이터를 선택하고 분석해야 할 때 .iloc와 .loc를 사용하여 데이터를 추출한다.

예) 주식 데이터에서 특정 기간의 수익률을 계산 또는 분석하기 위해 기간에 해당하는 데이터를 선택

1. pd.Series를 사용하여 value와 index를 가진 series를 생성한다.
2. 선택하고자 하는 데이터를 .index 혹은 .values 속성을 사용하여 선택한 후 새로운 변수에 저장한다.
3. 생성된 부분집합 변수를 출력한다.

※ iloc와 loc 속성 사용하여 다수의 시리즈 값 불러오기

- 특정 범위의 데이터를 선택하고 분석해야 할 때 .iloc와 .loc를 사용하여 데이터를 추출한다.

예) 주식 데이터에서 특정 기간의 수익률을 계산 또는 분석하기 위해 기간에 해당하는 데이터를 선택

- 특정 일자에 해당하는 주식 가격을 출력하기

```
In [41]: stock_prices = pd.Series([1500, 1510, 1520],  
                                   index=['2023-09-01', '2023-09-02', '2023-09-03'])
```

```
In [42]: print("※iloc: 정수 위치 기반")  
subset1 = stock_prices.iloc[0:1]  
print(subset1)
```

```
※iloc: 정수 위치 기반  
2023-09-01    1500  
dtype: int64
```

```
In [43]: print("※loc: 인덱스(라벨) 기반")  
subset2 = stock_prices.loc['2023-09-01':'2023-09-02']  
print(subset2)
```

```
※loc: 인덱스(라벨) 기반  
2023-09-01    1500  
2023-09-02    1510  
dtype: int64
```

※ Series index 이름 짓기

- 데이터 분석에서 series의 index에 의미 있는 이름을 지정하여 데이터를 더 쉽게 이해하고 문서화할 수 있다.
- `.index.name`를 사용하면 index의 설명을, `.name`를 사용하면 series의 설명을 제공할 수 있다.

1. 지금까지 생성했던 series 불러오기. 혹은 새로운 series 생성하기
2. 변수.`.index.name`를 통해 series의 index 이름을 설정한다.
3. 변수.`.name`를 통해 series의 이름을 설정한다.

`.name` → my first series

`.index.name` →

label	
a	100
b	200
c	300
d	400

※ Series index 이름 짓기

```
In [44]: stock_prices
```

```
Out [44]: 2023-09-01    1500  
          2023-09-02    1510  
          2023-09-03    1520  
          dtype: int64
```

```
In [45]: stock_prices.index.name = 'Date'
```

```
In [46]: stock_prices
```

```
Out [46]: Date_  
          2023-09-01    1500  
          2023-09-02    1510  
          2023-09-03    1520  
          dtype: int64
```

```
In [50]: stock_prices.name = 'stock prices and date'
```

```
In [51]: stock_prices
```

```
Out [51]: Date  
          2023-09-01    1500  
          2023-09-02    1510  
          2023-09-03    1520  
          Name: stock prices and date, dtype: int64
```

※ Series 데이터 삭제

- 필요가 없어진 데이터, 혹은 중복 데이터가 있을 경우 삭제하여 데이터를 업데이트할 수 있다.
- Del은 새로운 변수를 만들어 이전 데이터를 저장하지않고 바로 데이터를 삭제할 때 주로 사용된다.

예) 민감한 정보를 다루는 보안 시스템, 관심 없는 종목 즐겨찾기 삭제, 구매 취소/환불, 멤버십 회원권 종료/탈퇴 등

1. 지금까지 생성했던 series 불러오기. 혹은 새로운 series 생성하기
2. Del을 사용하여 삭제하고자 하는 고유의 index를 선택하기
3. 더 이상 존재하지 않는지 series를 확인하기

※ Series 데이터 삭제

- 필요가 없어진 데이터, 혹은 중복 데이터가 있을 경우 삭제하여 데이터를 업데이트할 수 있다.
- Del은 새로운 변수를 만들어 이전 데이터를 저장하지 않고 바로 데이터를 삭제할 때 주로 사용된다.

예) 민감한 정보를 다루는 보안 시스템, 관심 없는 종목 즐겨찾기 삭제, 구매 취소/환불, 멤버십 회원권 종료/탈퇴 등

```
In [52]: member
Out[52]: ID_001    2500
          ID_002   18000
          ID_003    3000
          dtype: int64
```



ID_002 회원 탈퇴

```
In [53]: del member['ID_002']

In [54]: member
Out[54]: ID_001    2500
          ID_003    3000
          dtype: int64
```

※ Series 수정 및 추가

- 데이터 중 잘못된 정보가 있을 경우, 해당 값을 수정하거나, 내용을 추가할 수 있다.

예) 잘못된 제공된 주식 가격을 수정, 가계부 지출 내역 금액 수정, 내역 업데이트 등

1. 지금까지 생성했던 series 불러오기. 혹은 새로운 series 생성하기
2. **(수정)** Series 변수['수정하고 싶은 index'] = 수정할 value
3. **(추가)** Series 변수['추가하고 싶은 index'] = 추가할 value
4. 출력하여 변경된 series 확인하기

※ Series 수정 및 추가: 수정

- 데이터 중 잘못된 정보가 있을 경우, 해당 값을 수정하거나, 내용을 추가할 수 있다.

예) 잘못된 제공된 주식 가격을 수정, 가계부 지출 내역 금액 수정, 내역 업데이트 등

```
In [55]: stock_prices
```

```
Out [55]: Date
2023-09-01    1500
2023-09-02    1510
2023-09-03    1520
Name: stock prices and date, dtype: int64
```



잘못된 제공된 주식 가격을 수정

```
In [56]: stock_prices["2023-09-03"] = 1800
```

```
In [57]: stock_prices
```

```
Out [57]: Date
2023-09-01    1500
2023-09-02    1510
2023-09-03    1800
Name: stock prices and date, dtype: int64
```

※ Series 수정 및 추가: 추가

- 데이터 중 잘못된 정보가 있을 경우, 해당 값을 수정하거나, 내용을 추가할 수 있다.

예) 잘못된 제공된 주식 가격을 수정, 가계부 지출 내역 금액 수정, 내역 업데이트 등

주식 날짜와 가격 업데이트

```
In [57]: stock_prices
```

```
Out [57]: Date
2023-09-01    1500
2023-09-02    1510
2023-09-03    1800
Name: stock prices and date, dtype: int64
```



```
In [58]: stock_prices["2023-09-04"] = 2000
```

```
In [59]: stock_prices
```

```
Out [59]: Date
2023-09-01    1500
2023-09-02    1510
2023-09-03    1800
2023-09-04    2000
Name: stock prices and date, dtype: int64
```

Series 연산

- Series의 값에 연산을 할 수 있다. 값에만 연산이 가능하기 때문에 인덱스는 변하지 않는다.
- `+`, `-`, `*`, `/` 와 같은 산술 연산 외에도 다양한 연산 수행 가능

```
In [36]: score/10
```

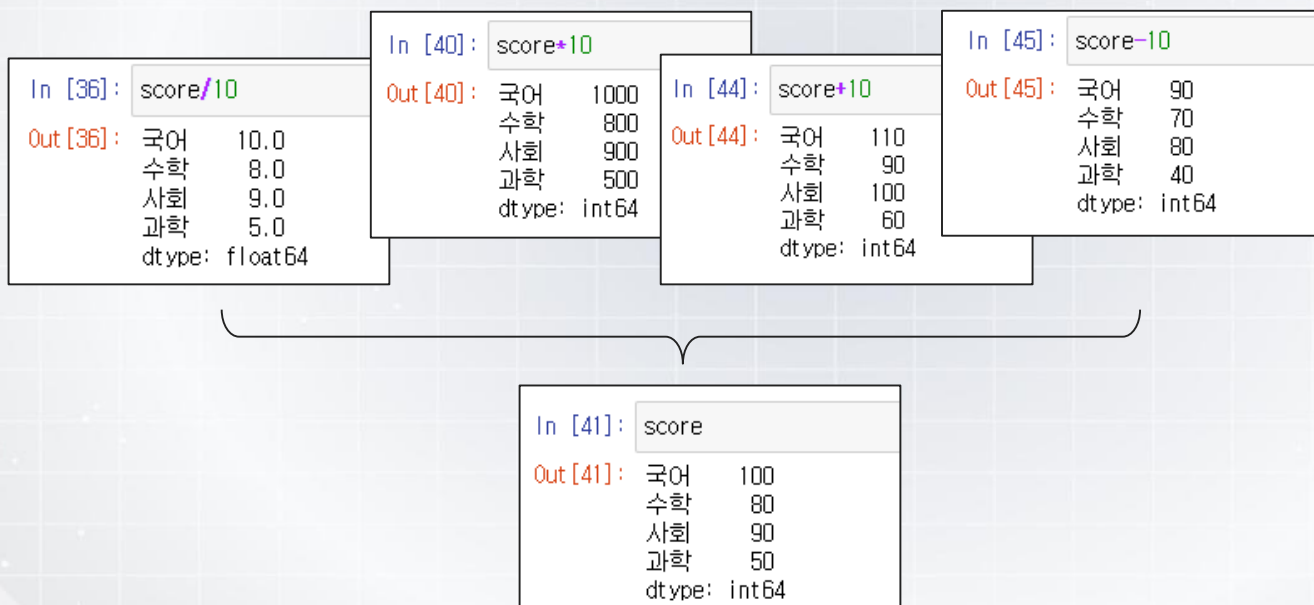
```
Out[36]: 국어      10.0  
         수학      8.0  
         사회      9.0  
         과학      5.0  
         dtype: float64
```

```
In [40]: score*10
```

```
Out[40]: 국어     1000  
         수학      800  
         사회      900  
         과학      500  
         dtype: int64
```

Series 연산

- 위에서 실행한 연산은 실행을 통해 연산 결과가 출력되지만, 실제 데이터에는 영향(수정)을 주지 않기에 원본 series의 값은 그대로 유지
- 연산결과를 반영한 새로운 series를 생성하고 싶다면, 새로운 변수를 만들어 저장해주어야 함



Series 연산

- 연산 결과를 반영한 새로운 series를 생성하고 싶다면, 새로운 변수를 만들어 연산 결과를 저장해주어야 함



```
In [47]: score2 = score-10
```

```
In [48]: score2
```

```
Out [48]: 국어    90  
          수학    70  
          사회    80  
          과학    40  
          dtype: int64
```

Series 연산

- series 간의 연산 가능
- 시리즈 간의 연산을 수행할 때는 인덱스가 서로 일치해야 함

```
In [53]: result = (score + score2)/2
```

||

```
In [41]: score
```

```
Out [41]: 국어    100
          수학     80
          사회     90
          과학     50
          dtype: int64
```

+

```
In [48]: score2
```

```
Out [48]: 국어     90
          수학     70
          사회     80
          과학     40
          dtype: int64
```

/ 2

=

```
In [54]: result
```

```
Out [54]: 국어    95.0
          수학    75.0
          사회    85.0
          과학    45.0
          dtype: float64
```

※ Series 연산 실습

•Pandas Series를 사용하여 값에 연산을 수행하면 실생활에서 다양한 계산 및 분석 작업을 수행할 수 있다.

예) 보유한 각 주식 계좌를 계산하여 전체 보유 금액을 파악할 수 있다.

1. 보유 수량이 다른 2개의 통장이 있다. (같은 index, 다른 value를 가진 2개의 series)
2. 2개의 통장을 새로운 통장을 만들어 합친다 (새로운 변수를 만들어 두개의 series를 더한 값을 저장)
3. 최종 합계 금액을 확인한다.

※ Series 연산 실습

•Pandas Series를 사용하여 값에 연산을 수행하면 실생활에서 다양한 계산 및 분석 작업을 수행할 수 있다.

예) 보유한 각 주식 계좌를 계산하여 전체 보유 금액을 파악할 수 있다.

1번 계좌 주식 보유 현황

```
In [67]: stock_1
```

```
Out [67]: naver      100
          apple      150
          microsoft  200
          dtype: int64
```

+

2번 계좌 주식 보유 현황

```
In [68]: stock_2
```

```
Out [68]: naver      50
          apple      30
          microsoft  20
          dtype: int64
```

=

총 주식 보유 현황

```
In [69]: total = stock_1 + stock_2
```

```
In [70]: total
```

```
Out [70]: naver      150
          apple      180
          microsoft  220
          dtype: int64
```

Serie 복사하기

- `.copy()`를 통해 원본 데이터와 완전히 같은 복사본을 생성할 수 있음
- 시리즈의 인덱스와 데이터 모두가 복사가 됨
- 데이터 조작 시 원본 데이터를 보존하여 안전하게 작업할 수 있음

```
In [32]: new_score = score.copy()
```

```
In [33]: new_score
```

```
Out [33]: 국어    100  
          수학    80  
          사회    90  
          과학    50  
          dtype: int64
```

Serie 복사하기

예) 퀴즈점수를 중간고사에 반영해야 될 때, 원본 중간고사 점수 데이터에는 영향을 주지 않아야 한다.

중간고사 성적이 담긴 series 생성

```
In [73]: m_scores = pd.Series([80, 50, 70], index=['Student1', 'Student2', 'Student3'])
```

```
In [78]: m_scores
```

```
Out [78]: Student1    80  
         Student2    50  
         Student3    70  
         dtype: int64
```

Serie 복사하기

예) 퀴즈점수를 중간고사에 반영해야 될 때, 원본 중간고사 점수 데이터에는 영향을 주지 않아야 한다.

```
In [73]: m_scores = pd.Series([80,50,70], index=['Student1', 'Student2', 'Student3'])
```

```
In [78]: m_scores
```

```
Out [78]: Student1    80  
Student2    50  
Student3    70  
dtype: int64
```

중간고사 점수 복사

```
In [80]: quiz = m_scores.copy()
```

```
In [81]: quiz
```

```
Out [81]: Student1    80  
Student2    50  
Student3    70  
dtype: int64
```

Serie 복사하기

예) 퀴즈점수를 중간고사에 반영해야 될 때, 원본 중간고사 점수 데이터에는 영향을 주지 않아야 한다.

퀴즈 점수 반영(수정)

```
In [75]: quiz['Student1'] = 90
```

퀴즈 점수를 반영한 중간고사 점수와 원본 점수

```
In [76]: quiz
```

```
Out [76]: Student1  90  
          Student2  50  
          Student3  70  
          dtype: int64
```

```
In [77]: m_scores
```

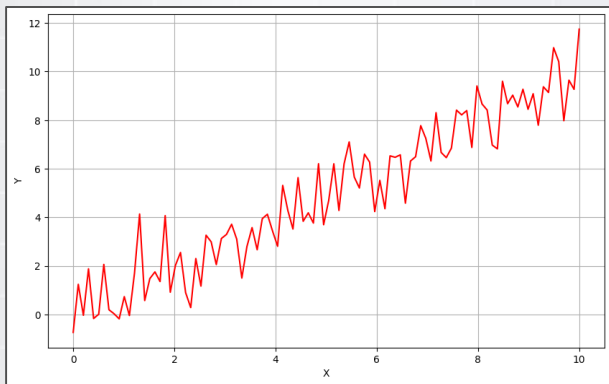
```
Out [77]: Student1  80  
          Student2  50  
          Student3  70  
          dtype: int64
```


결측치 : 누락된 데이터

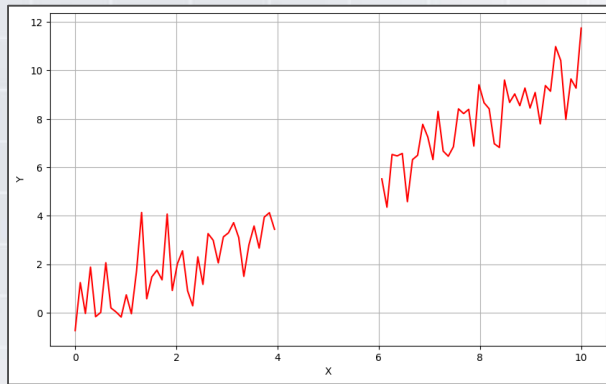
- 결측치는 데이터셋에서 일부 값이 존재하지 않거나 비어 있는 상태를 말한다.
- Pandas에서 이러한 결측치(누락된 데이터)는 NaN(Not a Number의 약자), none 등으로 표시된다.
- 데이터 수집 과정, 데이터 입력 오류, 연산 오류 등 다양한 문제로 인해 발생할 수 있다.
- 데이터에서 누락된 데이터가 있는 경우에는 해당 값을 채우거나 처리해야 함.
 - 결측치를 생성하는 방법
 - 결측치를 확인하는 방법
 - 결측치를처리하는 방법

결측치 : 누락된 데이터

정상 데이터로 이루어진 그래프



데이터가 누락된 그래프



결측치 : 누락된 데이터

- 아래 코드를 작성하면 NaN값이 생성된다.
- “영어”에 해당하는 값이 없기때문에 영어 값은 누락된다.
- 매칭되지 않는 Index가 추가 되었기 때문에 결측치가 생기는 것.

```
In [76]: data = {'국어':100, '수학':80, '사회':90, '과학':50}
          label = ['국어', '수학', '사회', '과학', '영어']
          score = pd.Series(data, index=label)
```

→ 매칭되는 값은 4개
→ 인덱스는 5개

```
In [77]: score
```

```
Out [77]: 국어    100.0
          수학     80.0
          사회     90.0
          과학     50.0
          영어      NaN
          dtype: float64
```

→ 누락 데이터 생성

None : 결측치를 생성하는 방법

- 임의로 결측치를 포함시키는 방법은 다양하다
- None을 값으로 할당시키면 출력 시 Nan값을 확인할 수 있다.
- None: 아무런 값도 없음을 의미. 특정 값 대신 'None'을 할당하여 명시적으로 결측치를 포함시키는 방법

```
In [13]: import pandas as pd

data = {'국어':100, '수학':80, '사회':90, '과학':50, '영어': None}
score = pd.Series(data)

In [14]: score
Out[14]: 국어    100.0
수학     80.0
사회     90.0
과학     50.0
영어      NaN
dtype: float64
```

np.nan : 결측치를 생성하는 방법

- Import numpy as np
- np.nan : NumPy(Numerical Python) 라이브러리에서 제공하는 특수한 값으로, 데이터의 누락, 부재, 혹은 정의되지 않은 값을 나타내기 위해 사용됨

```
In [11]: import numpy as np  
data = [100, 80, 90, 50, np.nan]  
label = ['국어', '수학', '사회', '과학', '영어']  
score = pd.Series(data, index = label)
```

```
In [12]: score
```

```
Out[12]: 국어      100.0  
수학       80.0  
사회       90.0  
과학      50.0  
영어        NaN  
dtype: float64
```

- np.nan 또는 None으로 직접 대입하는 방법은 누락된 데이터를 생성하는 방법

```
In [37]: score['과학'] = np.nan
```

```
In [38]: score
```

```
Out[38]: 국어    100.0  
수학     80.0  
사회     90.0  
과학      NaN  
영어      NaN  
dtype: float64
```

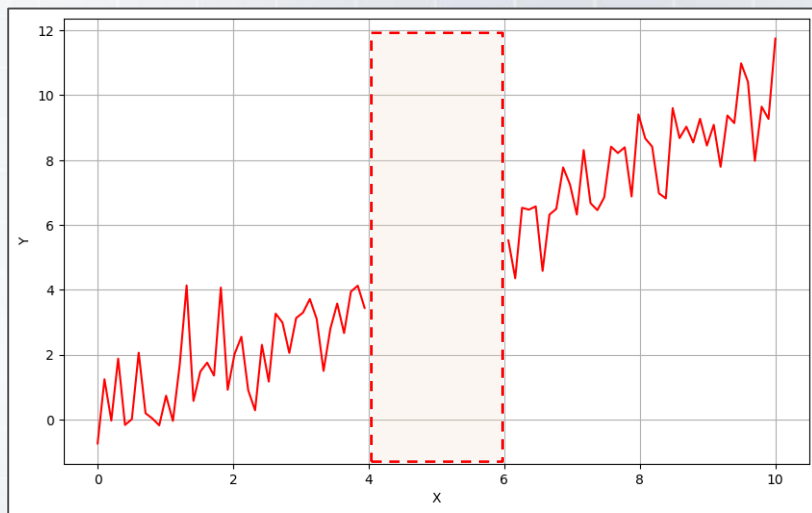
```
In [39]: score['사회'] = None
```

```
In [40]: score
```

```
Out[40]: 국어    100.0  
수학     80.0  
사회      NaN  
과학      NaN  
영어      NaN  
dtype: float64
```

결측치를 찾는 방법

- Pandas에서 제공하는 다양한 함수로 결측치를 찾을 수 있다.
- 결측치를 그대로 두면 분석 결과가 왜곡될 가능성이 있다.
- 잘못된 결론을 방지하고 모델의 정확도를 위해 처리해주어야 한다
- 그러기 위해서는 먼저 결측치 존재 여부를 확인해야 한다.



IsNull, notnull : 결측치를 확인하는 방법

- pandas에서 결측치를 찾을 때 사용되는 함수 중 하나
- isnull() : 주어진 series(혹은 dataframe)의 각 원소 중 결측치의 여부를 확인함
: NaN값이 있다면 True값 반환

```
In [84]: pd.isnull(score)
```

```
Out [84]: 국어      False  
          수학      False  
          사회      False  
          과학      False  
          영어      True  
          dtype: bool
```


IsNull, notnull : 결측치를 확인하는 방법

- pandas에서 결측치를 찾을 때 사용되는 함수 중 하나
- notnull() : isnull과 반대로 작동
 - : 주어진 series(혹은 dataframe)의 각 원소 중 결측치의 여부를 확인함
 - : NaN값이 없다면 True값 반환

```
In [85]: pd.notnull(score)
```

```
Out [85]: 국어      True  
          수학      True  
          사회      True  
          과학      True  
          영어      False  
          dtype: bool
```

IsNull, notnull, sum : 결측치의 개수를 확인하는 방법

- .sum()은 값의 series(혹은 dataframe) 내의 모든 값을 합산할 수 있다.
- isnull() 혹은 notnull() 뒤에 .sum()를 붙여 결합하여 사용하면 누락된 데이터, 누락되지않은 데이터의 개수를 셀 수 있다.

```
In [72]: score
Out[72]: 국어      100.0
          수학      80.0
          사회      90.0
          과학      50.0
          영어      NaN
          dtype: float64
```

4
1

```
In [70]: score.isnull().sum()
Out[70]: 1
```

```
In [71]: score.notnull().sum()
Out[71]: 4
```

누락된 데이터를 처리하는 방법

- 누락된 데이터를 확인하고, 대체하지 않아도 되는 데이터라면 바로 삭제해도 된다.
- 하지만 원래의 값을 알고 있는 상황이라면 해당하는 값으로 대체할 수 있다.
- 원래의 값을 모르는 상황이라면 주변 데이터를 추정하는 방법 등으로 보간할 수 있다.

dropna : 누락된 데이터를 처리하는 방법

- dropna() : 누락된 데이터가 있는 열 또는 행에 해당하는 항목을 삭제하는 방법
- 대체되거나 보간될 필요가 없거나, 상대적으로 적을 때 사용하는 방법

```
Out [22]: 국어      100.0  
          수학      80.0  
          사회      90.0  
          과학      50.0  
          영어      NaN  
          dtype: float64
```



※ 새로운 변수를 만들어 저장해주어야 한다.

```
In [24]: score_drop = score.dropna()  
  
In [25]: score_drop  
Out [25]: 국어      100.0  
          수학      80.0  
          사회      90.0  
          과학      50.0  
          dtype: float64
```

fillna : 결측치를 처리하는 방법

- fillna() : 결측치를 특정 값으로 채워 넣어 대체한다.

```
Out [22]: 국어    100.0  
          수학    80.0  
          사회    90.0  
          과학    50.0  
          영어     NaN  
          dtype: float64
```



```
In [46]: score_zero = score.fillna(0)
```

```
In [47]: score_zero
```

```
Out [47]: 국어    100.0  
          수학    80.0  
          사회    90.0  
          과학    50.0  
          영어     0.0  
          dtype: float64
```

fillna : 결측치를 처리하는 방법

- method란 대체 방법을 지정할 수 있는 옵션이다.
- .fillna(method='ffill') : ffill은 forward fill 이라는 의미로, 누락된 데이터의 앞 데이터 값으로 채우는 방법
- .fillna(method='bfill') : bfill은 backward fill 이라는 의미로, 누락된 데이터의 뒤 데이터 값으로 채우는 방법

```
Out [22]:
```

국어	100.0
수학	80.0
사회	90.0
과학	50.0
영어	NaN

dtype: float64



```
In [76]: score_ffill = score.fillna(method='ffill')
```

```
In [77]: score_ffill
```

```
Out [77]:
```

국어	100.0
수학	80.0
사회	90.0
과학	50.0
영어	50.0

dtype: float64

```
In [78]: score_bfill = score.fillna(method='bfill')
```

```
In [79]: score_bfill
```

```
Out [79]:
```

국어	100.0
수학	80.0
사회	90.0
과학	50.0
영어	NaN

dtype: float64

→ 뒤에 존재하는 데이터가
없어서 대체되지 못함

그 외의 다양한 방법으로 결측치를 처리할 수 있다.

※ 사용한 “score”는 변수 이름이며, 다른 변수이름을 사용했다면 그에 맞게 수정해주면 된다.

•평균(mean) : `score.fillna(score.mean())`

•중앙값(median) : `score.fillna(score.median())`

•최빈값(mode) : `score.fillna(score.mode().values[0])` →

모두 다른 값을 가지고 있다면 가까운 순서,

[0]은 첫번째 최빈값을 뜻하며,

최빈값이 하나일 경우에는 인덱스 0이상의 값은 사용할 수 없다.

•최소값(min) : `score.fillna(score.min())`

•최대값(max) : `score.fillna(score.max())`

etc.

+원본 데이터의 값을 알고 있는 경우

- 인덱스 정보를 사용하여 임의의 값을 대입해 결측치를 제거할 수 있다.

```
Out[22]: 국어    100.0  
         수학    80.0  
         사회    90.0  
         과학    50.0  
         영어     NaN  
         dtype: float64
```



```
In [99]: score["영어"] = 80  
  
In [100]: score  
Out[100]: 국어    100.0  
          수학    80.0  
          사회    90.0  
          과학    50.0  
          영어    80.0  
          dtype: float64
```


Series의 크기와 형태

• Series의 크기와 형태를 확인할 수 있는 여러 방법이 있다.

① `len()` : 파이썬의 내장함수로 객체의 길이를 반환한다. series의 경우 원소의 개수를 반환하여 길이를 알 수 있음

```
In [6]: score
```

```
Out [6]: 국어      100  
         수학      80  
         사회      90  
         과학      50  
         dtype: int64
```

```
In [7]: len(score)
```

```
Out [7]: 4
```

Series의 크기와 형태

• Series의 크기와 형태를 확인할 수 있는 여러 방법이 있다.

② size : series에 포함된 전체 원소 개수를 확인하여 반환한다.

```
In [6]: score
```

```
Out [6]: 국어      100  
         수학      80  
         사회      90  
         과학      50  
         dtype: int64
```

```
In [8]: score.size
```

```
Out [8]: 4
```

Series의 크기와 형태

• Series의 크기와 형태를 확인할 수 있는 여러 방법이 있다.

③ shape : 튜플 형태로 series의 차원 크기를 반환한다.

```
In [6]: score
```

```
Out [6]: 국어      100  
         수학      80  
         사회      90  
         과학      50  
         dtype: int64
```

```
In [9]: score.shape
```

```
Out [9]: (4,)
```




Series는 하나의 열로만 구성된 1차원 자료 구조형이기 때문에 (4,) 와 같이 출력되며,
Dataframe의 경우 2차원 자료 구조형이기 때문에 (행의 개수, 열의 개수)로 출력될 것.

Serie 데이터 미리보기

- Pandas에서 처음 일부, 혹은 마지막 일부 값을 모아주는 기능
- 데이터의 시작 부분을 확인하고, 데이터의 구조와 내용을 간략히 파악하는 데 유용
- `.head(n)` : 처음 `n`개의 행을 출력하며 기본값으로 5로 설정되어 있다.

```
In [14]: score.head(2)
```

```
Out [14]: 국어    100  
          수학    80  
          dtype: int64
```




국어	100
수학	80
사회	90
과학	50

Serie 데이터 미리보기

- Pandas에서 처음 일부, 혹은 마지막 일부 값을 모아주는 기능
- 데이터의 시작 부분을 확인하고, 데이터의 구조와 내용을 간략히 파악하는 데 유용
- `.tail(n)` : 마지막 `n`개의 행을 출력하며 기본값으로 5로 설정되어 있다.

```
In [16]: score.tail(2)
```

```
Out [16]: 사회      90  
          과학      50  
          dtype: int64
```



국어	100
수학	80
사회	90
과학	50

Series slicing(슬라이싱)

- 슬라이싱은 판다스에서 부분집합(subset, 서브셋)을 추출할 수 있는 방법
즉, 데이터를 범위로 선택하는 방법으로, 원하는 부분을 추출하거나 필요한 데이터만을 선택할 수 있다.
- 파이썬의 리스트나 문자열 슬라이싱과 유사하다.
 - start: 슬라이싱을 시작할 위치 인덱스
 - end: 슬라이싱을 끝낼 위치 인덱스 (이 인덱스는 포함되지 않음)
 - step: 슬라이싱을 건너뛴 간격을 지정 (기본값은 1)

Series[start:end:step]

```
In [28]: my_sl[1:4]
```

```
Out[28]: b    200  
         c    300  
         d    400  
         dtype: int64
```

```
In [29]: my_sl[:3]
```

```
Out[29]: a    100  
         b    200  
         c    300  
         dtype: int64
```

```
In [30]: my_sl[2:]
```

```
Out[30]: c    300  
         d    400  
         dtype: int64
```

```
In [31]: my_sl[::2]
```

```
Out[31]: a    100  
         c    300  
         dtype: int64
```