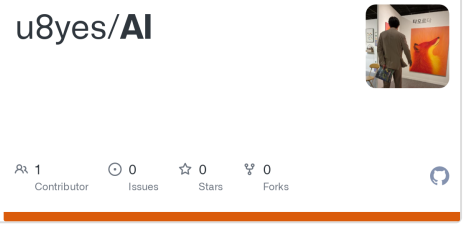
Day45; 20221109

날짜	@2022년 11월 9일
유형	@2022년 11월 9일
태그	

GitHub - u8yes/AI

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/u8yes/ai



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/8048b4a2-dae9-4144-9b64-629e3e9e7342/01_%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D_%EA%B0%9C%EB%85%90.pdf

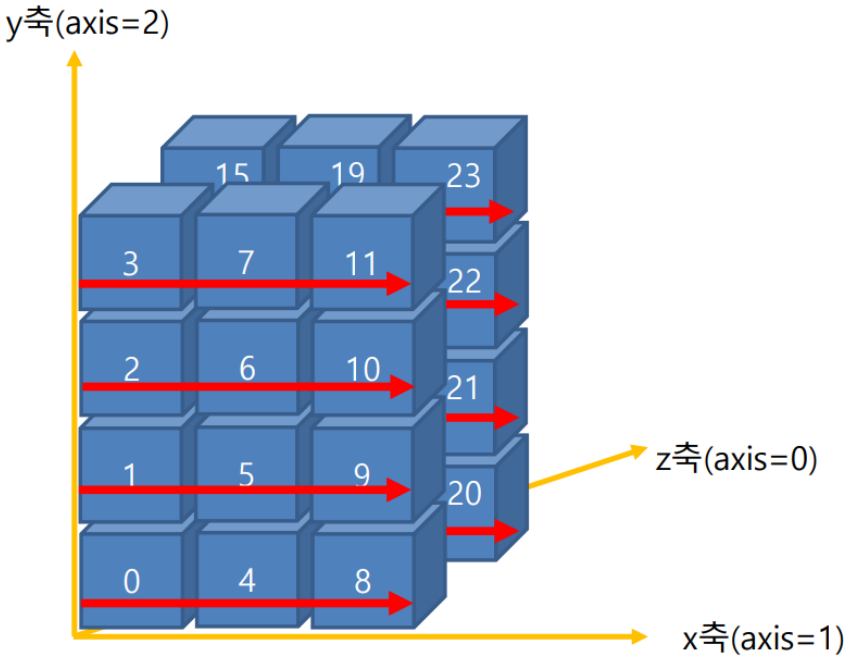
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/6627932d-c133-4fe4-bba6-a8c740baaf13/02_Numpy_20221107-08.ipynb

Numpy, Matplotlib

```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

3차원 기준) x축의 시작으로부터 벗어날수록 행의 모습의 형태를 보여줌(axis = 1)

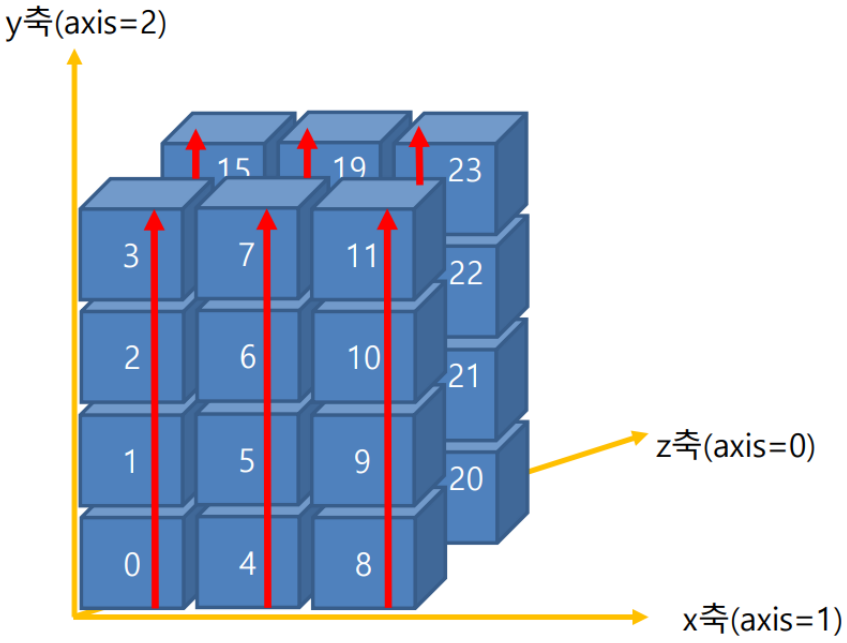
```
>>> res02=v.sum(axis=1) # axis=1 기준 합계
>>> res02.shape
(2, 4)
>>> res02
array([[ 12, 15, 18, 21],
       [48, 51, 54, 57]])
```



3차원 기준) y축의 시작으로부터 벗어날수록 열의 모습의 형태를 보여줌(axis = 2)

axis 이해

```
>>> res03=v.sum(axis=2) # axis=2 기준 합계
>>> res03.shape
(2, 3)
>>> res03
array([[ 6, 22, 38],
       [54, 70, 86]])
```



sqld)

5.6.9 단일행 CASE 표현의 종류

특정 값에 대해서 조건에 따라 각기 다른 값을 리턴하도록 하는 것을 CASE 표현이라고 합니다. 이러한 CASE 표현은 CASE문의 사용 혹은 DECODE 함수를 이용하여 구현할 수 있습니다.

표 5-23 CASE 표현의 종류

종류	설명
CASE WHEN 조건 THEN 값 혹은 SQL문 ELSE 값 혹은 SQL문 END	조건이 맞으면 THEN절을 수행하고 그렇지 않으면 ELSE절을 수행한다.
DECODE(조건1, 값1, 조건2, 값2, 디폴트 값)	조건1이 TRUE이면 값1을 가져오고, 그렇지 않고 조건2가 TRUE이면 값2를 가져오고, 그렇지 않으면 디폴트 값을 가져온다.

In [82]:
```
v = arr.reshape([2,3,4]) # 차원변환([z축(depth), x축(row), y축(column)])
v
```

Out[82]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

In [83]:
```
print(v.shape) # 면, 행, 열
print(v.ndim) # n차원
print(v.sum())
```

```
(2, 3, 4)
3
276
```

In [84]:
```
res01 = v.sum(axis = 0) # z축 면을 기준으로 sum해줘라.
print(res01.shape)
print(res01)
```

```
(3, 4)
[[12 14 16 18]
 [20 22 24 26]
 [28 30 32 34]]
```

In [85]:
```
res02 = v.sum(axis = 1) # x축 면을 기준으로 sum해줘라.
print(res02.shape) # (2, 4) 2면 0123+0123+0123+(4열)012행 합계
print(res02)
```

```
(2, 4)
[[12 15 18 21]
 [48 51 54 57]]
```

In [87]:
```
res03 = v.sum(axis = 2) # y축 면을 기준으로 sum해줘라.
print(res03.shape) # (2, 3) 2면 0123+0123+(3행)0123열
print(res03)
```

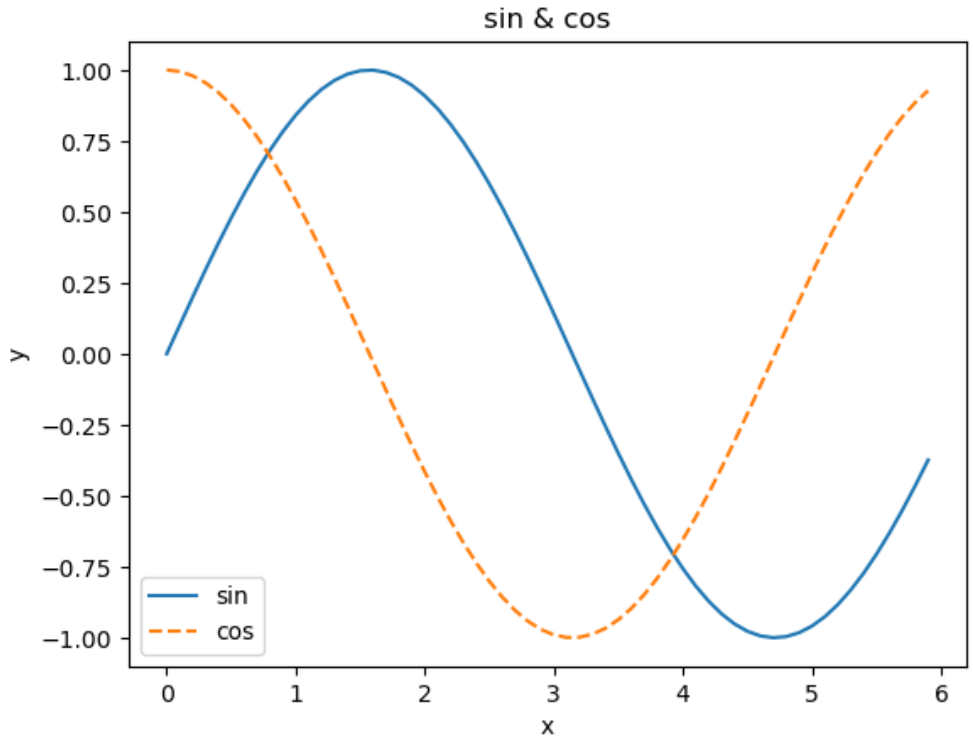
```
(2, 3)
[[ 6 22 38]
 [54 70 86]]
```

Matplotlib

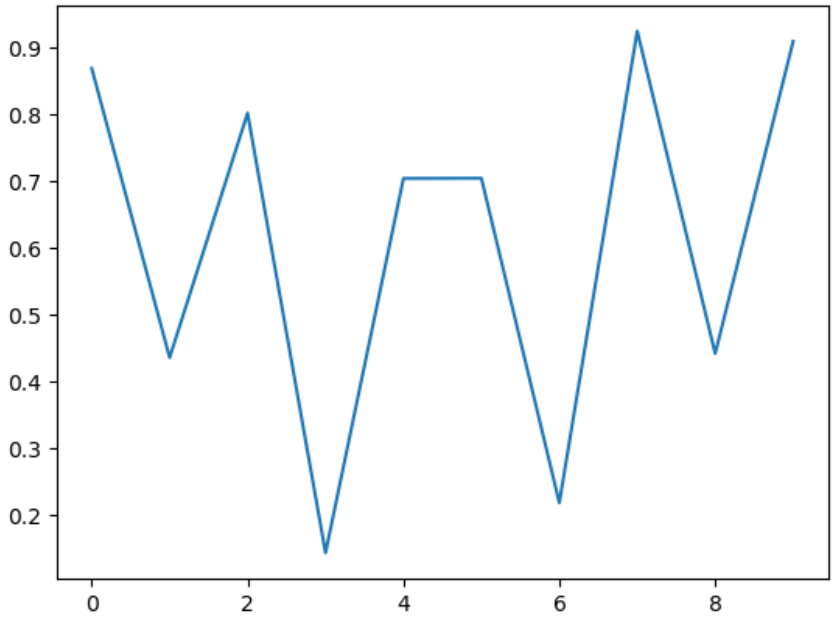
In [4]:
```
import matplotlib.pyplot as plt
import numpy as np # 전통적인 분석에서 일반적으로 사용하는 방법
```

In [11]:
```
# 단순한 그래프 그리기

x = np.arange(0, 6, 0.1)
y1 = np.sin(x) # 사인
y2 = np.cos(x) # 코사인
```

In [26]:
```
# 그래프 그리기 # R에서는 x축, y축으로 2개의 변수가 있었다.

plt.plot(x, y1, label = 'sin')
plt.plot(x, y2, linestyle='--', label = 'cos')
plt.xlabel('x') # x축 이름
plt.ylabel('y') # y축 이름
plt.title('sin & cos') # 제목
plt.legend() # 범례
plt.show()
```



In [36]:
```
# np.random.seed(1234)
# 1000분의 1초값을 난수로 발생시킴 # seed값을 초기화시키면 항상 똑같은 난수가 발생 # 난수를 고정
x = np.arange(10)
y = np.random.rand(10) # 0 ~ 1 사이의 균일한 분포

plt.plot(x, y) # 꺾은 선 그래프를 등록
plt.show()
```



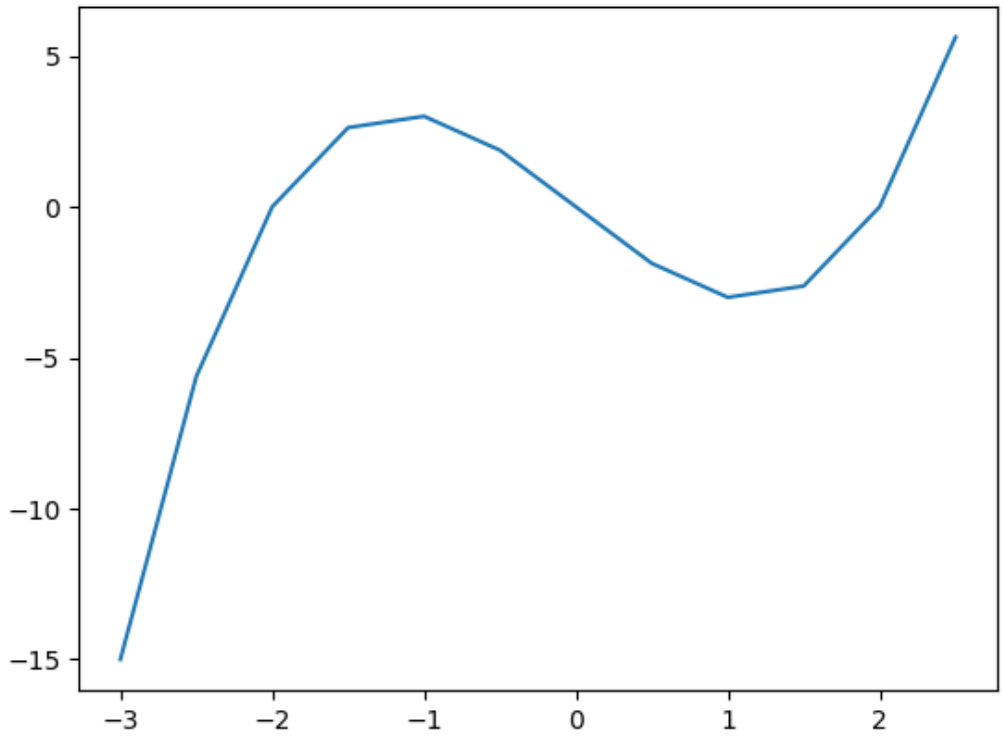
In [39]:
```
# 3차 함수 f(x) = (x-2) * x * (x+2)

def f(x):
    return (x-2) * x * (x+2)

print(f(0))
print(f(2))
print(f(-2))
```

```
0
0
0
```

In [42]:
```
# x 값에 대해 ndarray 배열이며 각각에 대응한 f를 한꺼번에 ndarry로 돌려줌
print(f(np.array([1,2,3]))) # 전달된 갯수만큼 계산돼서 리턴해줌.
print(type)
```

```
[-3  0 15]
<class 'type'>
```

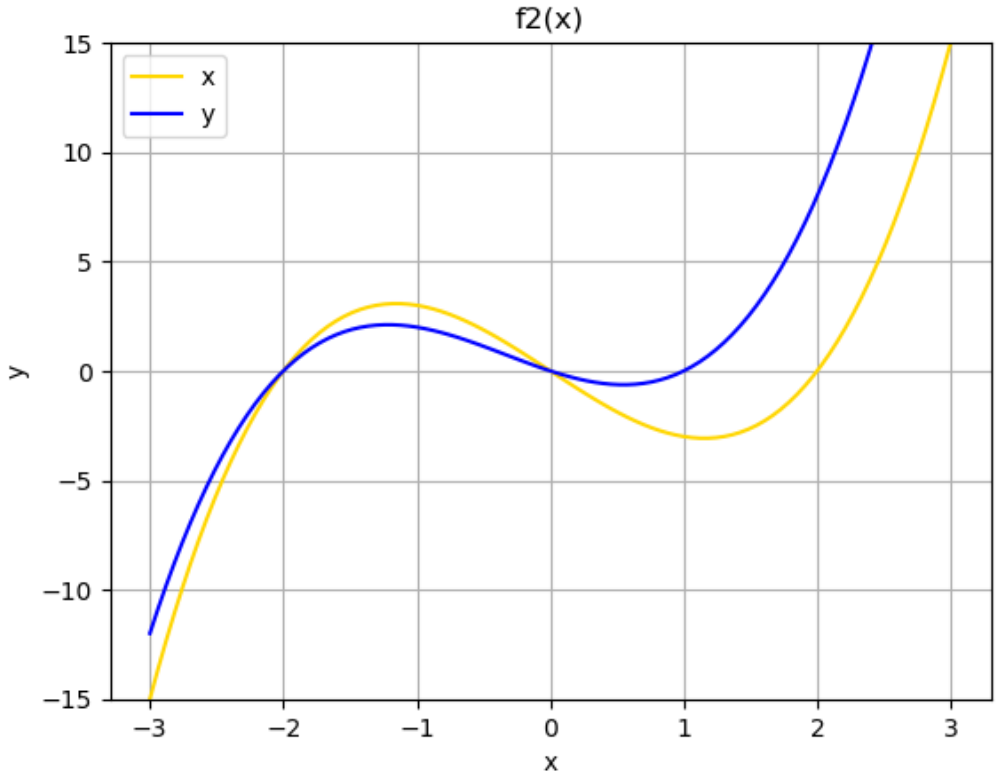
In [44]:
```
# 그래프를 그리는 x의 범위를 -3 - +3 까지로 하고, 간격 0.5
x = np.arange(-3, 3, 0.5)

plt.plot(x, f(x))
plt.show()
```



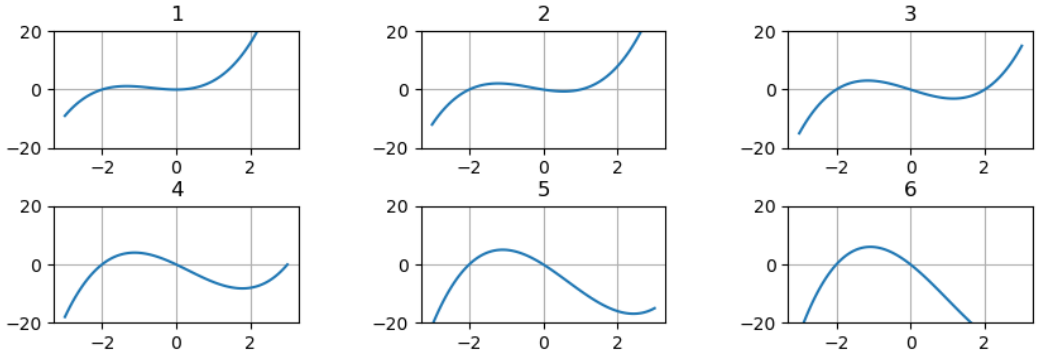
In [45]:
```
# 그래프를 장식
def f2(x, w):
    return (x-w) * x * (x+2) # 함수 정의
```

In [48]:
```
# x를 정의
x = np.linspace(-3,3,100) # -3 ~ 3 까지 x를 100개 균등하게 분할하기
```

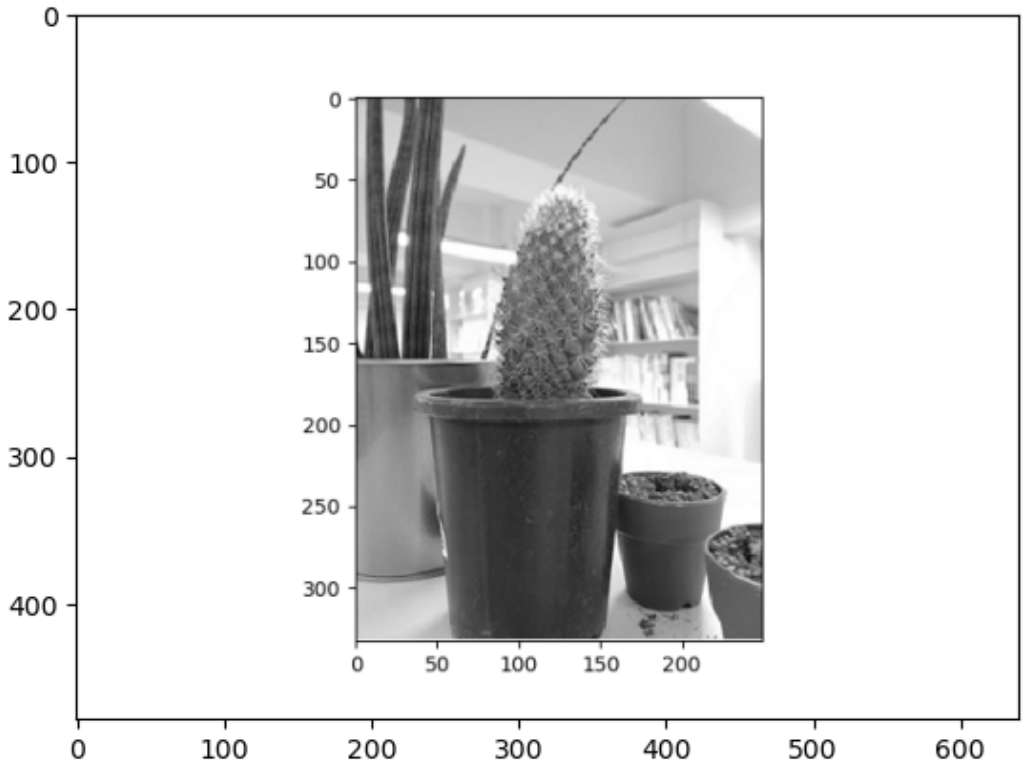
In [62]:
```
# 차트 묘사
plt.plot(x, f2(x, 2), color='gold', label='x')
plt.plot(x, f2(x, 1), color='blue', label='y')
plt.legend(loc='upper left') # 아래는 bottom left or bottom right
plt.ylim(-15, 15) # y축 범위
plt.title('f2(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True) # 그리드(눈금)
plt.show()
```



In [64]:
```
# 그래프를 여러 개 보여주기

plt.figure(figsize=(10,3)) # 전체 영역의 크기를 지정
plt.subplots_adjust(wspace=0.5, hspace=0.5) # 그래프의 간격을 지정
for i in range(6): # 6번 반복하게 해줌
    plt.subplot(2,3,i+1) # 그래프 위치를 지정
    plt.title(i+1)
    plt.plot(x, f2(x, i)) # y축은 f2함수를 그려줌
    plt.ylim(-20, 20)
    plt.grid(True)

plt.show() # show는 한번만 출력
```



In [66]:
```
# 이미지 표시하기

from matplotlib.image import imread # imread함수 호출

img = imread('images/fig1.png') # 이미지 읽어오기

plt.imshow(img)
plt.show()
```



Pandas

Pandas는 무엇인가요

- 데이터 분석 및 가공에 사용되는 파이썬 라이브러리

In [14]:
```
import pandas as pd

pd.__version__
```

Out[14]:
```
'1.3.5'
```

In [18]:
```
data_frame = pd.read_csv('data/friend_list.csv')
data_frame
```

Out[18]:

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	30	teacher
3	Julia	40	dentist
4	Brian	45	manager
5	Chris	25	intern

데이터 프레임_20221109

- 가로축과 세로축이 있는 엑셀과 유사한 데이터 구조
- 가로축은 row(행), 세로축은 column(열)
- 데이터베이스의 테이블 구조

In [22]:
```
# 데이터프레임이 가지고 있는 함수의 예제
data_frame.head(3) # default : 5 개를 보여줌 (R은 6개)
```

Out[22]:

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	30	teacher

시리즈(Series)_20221109

- 데이터 프레임의 컬럼(열)은 모두 시리즈.
- 단순히 파이썬 리스트를 간직한 오브젝트.
- 리스트를 파라미터로 주면 바로 시리즈가 생성.
- 데이터 가공 및 분석이 파이썬 리스트보다 훨씬 많다.

[25]:
```
type(data_frame.job) # pandas.core.series.Series # 대문자 Series 클래스
```

[25]:
```
pandas.core.series.Series
```

[27]:
```
# 시리즈의 함수 예제
data_frame.job = data_frame.job.str.upper()
data_frame.head()
```

[27]:

	name	age	job
0	John	20	STUDENT
1	Jenny	30	DEVELOPER
2	Nate	30	TEACHER
3	Julia	40	DENTIST
4	Brian	45	MANAGER

[30]:
```
s1 = pd.core.series.Series(['one', 'two', 'three'])
s2 = pd.core.series.Series([1,2,3])
```

[31]:
```
pd.DataFrame(data=dict(word=s1, num=s2))
```

[31]:

	word	num
0	one	1
1	two	2
2	three	3

빅분기 실기)

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/b3a05d1a-8a52-48ac-bdcc-0eaa47edeb23/%EB%B9%85%EB%B6%84%EA%B8%B0_%EB%8B%A8%EB%8B%B5%ED%98%95%EB%8C%80%EB%B9%84.pdf

빅분기 실기 환경 체험하기 링크 (단답형, 작업형 1) : https://dataq.goorm.io/exam/116674/체험하기/quiz/1

빅분기 실기 환경 체험하기 링크 (작업형 2) : https://dataq.goorm.io/exam/116674/체험하기/quiz/3

빅분기 관련 연습 문제 모음 사이트

: https://www.datamanim.com/dataset/99_pandas/pandasMain.html

판다스 옵션 설정 관련 : https://runebook.dev/ko/docs/pandas/user_guide/options

< 문제에 따라 아래의 max_rows, max_columns에 대입되는 숫자를 변경해야 합니다>

import pandas as pd
pd.options.display.max_rows = 999
pd.op...

In [33]:
```
df = pd.read_csv('data/friend_list.txt')
df.head()
```

Out[33]:

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	30	teacher
3	Julia	40	dentist
4	Brian	45	manager

In [35]:
```
# 구분자에 의해 컬럼이 구분되어져 있는 데이터는 모두 지원
df = pd.read_csv('data/friend_list_tab.txt', delimiter='₩t') # delimiter = 구분자 # R에서도 delimiter 매개변수 사용
df.head()
```

Out[35]:

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	30	teacher
3	Julia	40	dentist
4	Brian	45	manager

delimiter

미국·영국[dilímitər]

명사

구획 문자 ((자기(磁氣) 테이프에서 데이터의 시작[끝]을 나타냄))

영어사전 결과 더보기

어학사전 더보기 →

지식백과

구분 문자 delimiter

일반적으로 텍스트 중의 문자열(string)을 목적에 따라서 특수문자로 구분하는 경우와 프로그래밍 언어 중의 문법의 일부로서 구분을 표시하는 경우와 데이터 전송에 있어서의 텍스트의 개시, 종료를 표시하는 것 등 명확히 용도를 정해둔 경우가 있다. 전자의 경우에는 ( ), ; 등이 흔히 사용된다. 후자의 경우에는 COBOL에서는 「.」를...

ut [39] :

	0	1	2
0	John	20	student
1	Jenny	30	developer
2	Nate	30	teacher
3	Julia	40	dentist
4	Brian	45	manager

n [40] :
```
df.columns = ['name','age','job']
df.head()
```

ut [40] :

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	30	teacher
3	Julia	40	dentist
4	Brian	45	manager

n [41] :
```
df = pd.read_csv('data/friend_list_no_head.csv', header = None, names=['name','age','job'])
df.head()
```

ut [41] :

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	30	teacher
3	Julia	40	dentist
4	Brian	45	manager

데이터 프레임을 파이썬 코드로 생성하기

[46]:
```
friend_dict_list = [{'name':'Jane','age':20,'job':'student'},
                    {'name':'Jenny','age':30,'job':'developer'},
                    {'name':'Nate','age':25,'job':'teacher'}]

df = pd.DataFrame(friend_dict_list)
df.head()
```

[46]:

	name	age	job
0	Jane	20	student
1	Jenny	30	developer
2	Nate	25	teacher

[48]:
```
df = df[['name','age','job']] # 만약에 순서가 다르게 나왔을 경우 따로 이름을 지정을 해줘서 변수로 담아줘야함.
df.head()
```

[48]:

	name	age	job
0	Jane	20	student
1	Jenny	30	developer
2	Nate	25	teacher

OrderedDict으로 데이터 프레임 생성하기

- OrderedDict 자료구조로 데이터프레임을 생성하면, 컬럼의 순서가 뒤바뀌지 않음.

51]:
```
from collections import OrderedDict
```

58]:
```
friend_ordered_dict = OrderedDict([('name',['John','Jenny','Nate']),
                                   ('age',[20, 30, 25]),
                                   ('job',['student','developer','teacher'])])

df = pd.DataFrame.from_dict(friend_ordered_dict) # 메서드 체이닝 방식으로 from_dict메서드 호출
df.tail(2)
```

58]:

	name	age	job
1	Jenny	30	developer
2	Nate	25	teacher

list로 데이터프레임 생성하기

```
friend_list= [['John', 20, 'student'],
              ['Jenny',30,'developer'],
              ['Nate',25,'teacher']]

column_name = ['name','age','job']

df = pd.DataFrame.from_records(friend_list, columns=column_name) # list에서 DataFrame으로 바꿔주는 메서드는 from_records
df.head()
```

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	25	teacher

```
friend_dict = {'name':['John','Jenny','Nate'],
               'age': [20, 30, 25],
               'job': ['student','developer','teacher']
               }

df = pd.DataFrame.from_dict(friend_dict) # 메서드 체이닝 방식으로 from_dict메서드 호출
df.head()
```

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	25	teacher

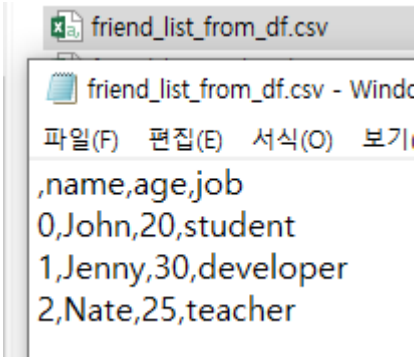
[64]:
```
friend_dict = {'name':['John','Jenny','Nate'],
               'age': [20, 30, 25],
               'job': ['student','developer','teacher']
               }

df = pd.DataFrame.from_dict(friend_dict) # 메서드 체이닝 방식으로 from_dict메서드 호출
df.head()
```

[64]:

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	25	teacher

[65]:
```
df.to_csv('data/friend_list_from_df.csv')
```

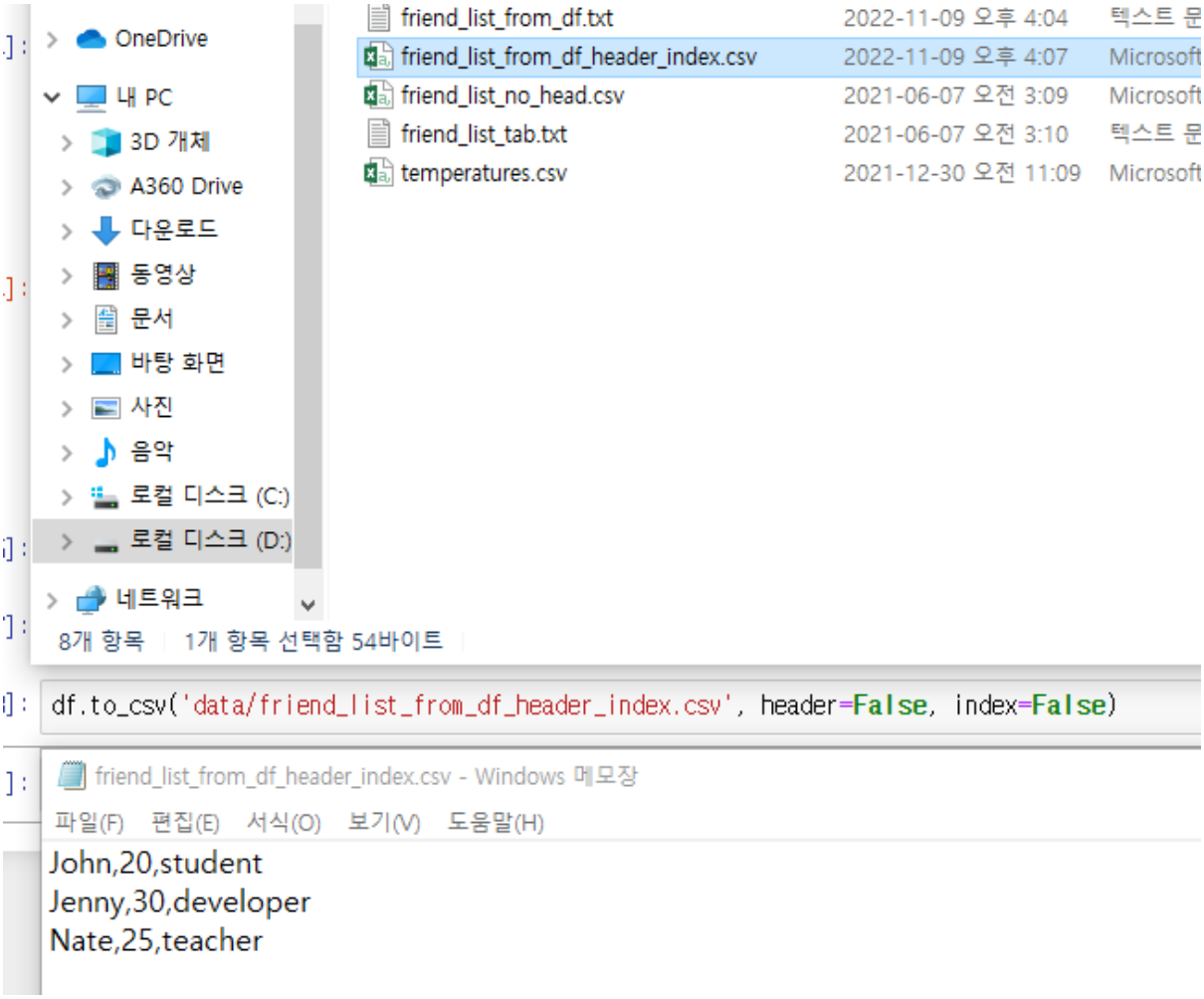
friend_list_from_df.csv




```
df.to_csv('data/friend_list_from_df.csv') # 만들어짐
```

```
df.to_csv('data/friend_list_from_df.txt') # csv포맷만 만들어지는 것이 아니라 txt도 생성 가능함.
```

```
df.to_csv('data/friend_list_from_df_header_index.csv', header=False, index=False)
```



header와 index 사라짐

9]:
```
friend_dict = {'name':['John','Jenny','Nate'],
               'age': [20, None, 25], # Not a Number = NaN
               'job': ['student','developer','teacher']
               }

df = pd.DataFrame.from_dict(friend_dict)
df.head()
```

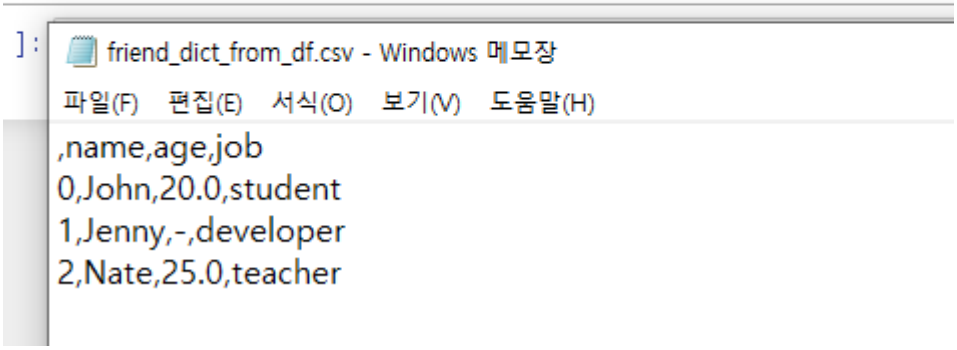
9]:

	name	age	job
0	John	20.0	student
1	Jenny	NaN	developer
2	Nate	25.0	teacher

1]:
```
df.to_csv('data/friend_dict_from_df.csv')
```

2]:
```
df.to_csv('data/friend_dict_from_df.csv', na_rep='-')
```

]:



데이터 접근 방법_20221109

- 인덱스로 row 선택하기

In [76]:
```
friend_dict = {'name':['John','Jenny','Nate'],
               'age': [20, 30, 25],
               'job': ['student','developer','teacher']
               }

df = pd.DataFrame.from_dict(friend_dict)
df.head()
```

Out [76]:

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	25	teacher

In [79]:
```
df[1:3] # row로 지정해줌 # 데이터프레임은 슬라이싱으로
```

Out [79]:

	name	age	job
1	Jenny	30	developer
2	Nate	25	teacher

In [ ]:
```
df[0,2] # 행렬처럼 될 수 있기 때문에 KeyError 오류 # KeyError: (0, 2)
```

In [ ]:
```
df[0,2] # KeyError: (0, 2) # 행렬처럼 될 수 있기 때문에 KeyError 오류
```

In [87]:
```
# 순차적이지 않은 row를 선택
df.loc[[0,2]] # location(위치) 제공 # list 자료형으로 표현하면 행만 출력된다.
```

Out [87]:

	name	age	job
0	John	20	student
2	Nate	25	teacher

In [88]:
```
df.loc[[0:2]] # SyntaxError: invalid syntax # loc는 슬라이싱 기능 지원 안 됨.
```

```
  File "C:₩Users₩tjouen-jr₩AppData₩Local₩Temp₩ipykernel_4224₩4278472225.py", line 1
    df.loc[[0:2]]
           ^
SyntaxError: invalid syntax
```

컬럼값에 따른 row 선택하기_20221109

- 마치 데이터베이스에 쿼리를 전달하듯, 특정한 컬럼값을 충족하는 row만 선택.

In [94]:
```
df_filtered = df[df.age > 25]
df_filtered
```

Out [94]:

	name	age	job
1	Jenny	30	developer

In [96]:
```
df_query = df.query('age > 25')
df_query
```

Out [96]:

	name	age	job
1	Jenny	30	developer

In [102]:
```
df_filtered = df[(df.age >= 25) & (df.name == 'Nate')]
df_filtered
```

Out [102]:

	name	age	job
2	Nate	25	teacher

---

loc는 슬라이싱이 지원되지 않음.

pandas.데이타프레임은 [1:3] 형식으로 순차적으로 볼 수 있다.

컬럼 필터하기_20221109

인덱스로 필터하기

In [105]:
```
friend_list = [['John',20,'student'],
               ['Jenny',30,'developer'],
               ['Nate', 25, 'teacher']]

df = pd.DataFrame.from_records(friend_list)
df
```

Out[105]:

	0	1	2
0	John	20	student
1	Jenny	30	developer
2	Nate	25	teacher

In [111]:
```
# 모든 row를 보여주되, 컬럼은 0 ~ 1까지만 출력.

df.iloc[:, :2] # i location는 행,열 전체를 컨트롤 할 수 있게 만들어준 메서드
```

Out[111]:

	0	1
0	John	20
1	Jenny	30
2	Nate	25

In [112]:
```
df.iloc[:, [0,2]]
```

Out[112]:

	0	2
0	John	student
1	Jenny	developer
2	Nate	teacher

---

iloc[행,열]는 구조로 접근 가능해진다.

컬럼 이름으로 필터링하기

In [121]:
```
df = pd.read_csv('data/friend_list_no_head.csv', header=None, names=['name','age','job']) # pd.read_table()
df
```

Out[121]:

	name	age	job
0	John	20	student
1	Jenny	30	developer
2	Nate	30	teacher
3	Julia	40	dentist
4	Brian	45	manager
5	Chris	25	intern

In [122]:
```
df_filtered = df[['name', 'age']]
df_filtered
```

Out[122]:

	name	age
0	John	20
1	Jenny	30
2	Nate	30
3	Julia	40
4	Brian	45
5	Chris	25

In [123]:
```
df.filter(items=['age','job'])
```

Out[123]:

	age	job
0	20	student
1	30	developer
2	30	teacher
3	40	dentist
4	45	manager
5	25	intern

지도 분석 - 상관분석

비지도 분석 - 군집분석, 연관분석

In [125]:
```
df.filter(like='a', axis=1) # (axis=1 기준으로 검색해서 보면) header 이름에 a가 들어가있으면 열 기준으로 출력
```

Out [125]:

	name	age
0	John	20
1	Jenny	30
2	Nate	30
3	Julia	40
4	Brian	45
5	Chris	25

In [126]:
```
# 정규식 # regular expression 정규 표현식
df.filter(regex='b$', axis=1) # b$로 해도 되고 b 들어가있는 거 검색해줌
```

Out [126]:

	job
0	student
1	developer
2	teacher
3	dentist
4	manager
5	intern