



# PyTorch 코딩

---

ANN을 위한 Python 라이브러리

# Numpy와 Pillow

# 01

# Numpy

- Numpy란?

- 고차원 array를 전문적으로 다루는 라이브러리
- 선형대수학 및 빅데이터 처리에 특화됨
- 다른 Python 라이브러리와 함께 다양한 목적으로 활용 가능
  - Pillow 라이브러리와 함께 이미지 처리에 활용
  - Pandas 라이브러리와 함께 빅데이터 통계에 활용
  - matplotlib 라이브러리와 함께 데이터 시각화에 활용
  - 기타 등등

- Pillow란?

- Python에서 일반적으로 사용되는 이미지 관련 라이브러리
- Numpy와 호환성이 좋음

# Numpy

## • Numpy array 생성

> import <b>numpy</b> as <b>np</b>	# Numpy 라이브러리
> a = np. <b>array</b> ((5, 3, 3, 7, 8))	# 1차원 array 생성, tuple 또는 list로 원소 지정 가능
> b = np. <b>array</b> ([2, 9, 4, 1, 3])	
> print(a, b)	
> c = np. <b>zeros</b> (3)	# 0을 원소로 가지는 1차원 array 생성
> print(c)	# Numpy에서는 1차원 array의 방향(가로, 세로)을 구분하지 않음
> d = np. <b>zeros</b> ((3, 3))	# 0을 원소로 가지는 3 × 3 array 생성, zeros를 통해 다차원 array를 만들 땐 예시와 같이 dimension을 tuple로 묶어줘야 함
> print(d)	
> print(d. <b>shape</b> )	# c의 dimension 출력
> print(d. <b>dtype</b> )	# c의 자료형 출력
> e = np. <b>ones</b> ((3, 3))	# 1을 원소로 가지는 3 × 3 array 생성
> f = np. <b>eye</b> (3)	# 3 × 3 단위 행렬(identity matrix) 생성, 정방 행렬이므로 한 쪽 사이즈만 지정
> g = np. <b>random.rand</b> (3, 3)	# 임의의 값들로 이루어진 3 × 3 array 생성, zeros나 ones와 달리 다차원에서도 dimension을 tuple로 묶을 필요 없음
> h = np. <b>zeros</b> ((3, 3), <b>dtype</b> = np. <b>float32</b> )	# Array의 원소 자료형을 32bit float로 지정
> h = h. <b>astype</b> (np. <b>int8</b> )	# Array의 원소 자료형을 8bit integer로 변경, 변경된 array를 활용하려면 대입 연산자를 통해 저장해야 함

# Numpy

- Numpy array dimension 재정의

```
> import numpy as np  
  
> a = np.arange(4*4*4*4)    # 0부터 연속된 정수로 이루어진 1차원 array 생성  
> print(a)  
> print(a.shape)  
> print(a.dtype)  
  
> a = a.reshape(4, 4, 4, 4) # 4차원 tensor로 재정의, 재정의된 array를 활용하기 위해 대입 연산자를 통해 저장해야 함  
> print(a)  
> print(a.shape)
```

# Numpy

- Numpy array 연산

```
> import numpy as np

> v = np.arange(4)      # 1차원 array 생성
> print(v, v.shape)

> print(v + 1)          # 1차원 array와 스칼라 덧셈
> print(v + v)          # 1차원 array와 1차원 array를 덧셈, 차원이 같아야 함

> print(v * 2)          # 1차원 array와 스칼라 곱셈
> print(np.dot(v, v))   # Dot product, array의 원소 개수만 같으면 됨

> vt = np.transpose(v)  # v를 transpose한 1차원 array 생성
> print(vt, vt.shape)   # 1차원 array의 dimension은 방향과 상관없이 원소의 개수로만 정해짐, v와 vt는 같은 dimension을 가짐
> print(np.dot(vt, v))
```

# Numpy

- Numpy array 연산

```
> import numpy as np

> v = np.arange(4)
> v = v.reshape(4,1)      # 1차원 벡터 v를 2차원 array로 재정의, 가로 세로 방향을 명시할 수 있음
> vt = np.transpose(v)

> print(v, v.shape)        # 4 × 1 세로 array
> print(vt, vt.shape)      # 1 × 4 가로 array

> print(np.dot(vt, v))     # 가로 array와 세로 array를 dot product하여 스칼라를 생성
> print(np.dot(v, vt))     # ???
```

# Numpy

- Numpy array 연산

```
> import numpy as np

> m = np.arange(2 * 3)
> m = m.reshape(2, 3)    # 2 × 3 array 생성
> l = np.eye(3)           # 3 × 3 단위 행렬 생성

> print(m)
> print(np.transpose(m)) # Array를 transpose함
> print(np.dot(m, l))    # 2 × 3 array m과 3 × 3 array l를 dot product
> print(np.dot(l, m))    # ???
```



# Numpy

- Numpy array indexing

```
> import numpy as np

> v = np.arange(4)      # 1차원 array

> print(v)
> print(v[0])          # v의 첫 번째 원소
> print(v[3])          # v의 네 번째 원소
> print(v[-1])         # v의 마지막 원소

> m = np.arange(4*4)
> m = m.reshape(4, 4)  # 2차원 array

> print(m)
> print(m[0, 0])       # m의 첫 번째 원소
> print(m[3, 3])       # m의 4번째 행과 4번째 열의 원소
> print(m[-1, -1])     # m의 마지막 원소

> print(m[0])          # ???
> print(m[-1])         # ???
```

# Numpy

## • Numpy array indexing

```
> import numpy as np

> m = np.arange(4*4)
> m = m.reshape(4, 4)    # 2차원 array
> print(m)

> print(m[0])             # m의 첫 번째 행
> print(m[-1])            # m의 마지막 행

> print(m[0, :])          # m의 첫 번째 행, : 은 해당 차원의 모든 원소를 뜻함
> print(m[-1, :])         # m의 마지막 행

> print(m[:, 0])          # m의 첫 번째 열
> print(m[:, -1])         # m의 마지막 열
```

m

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

m[0, 0]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

m[0, :]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

m[:, 0]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

# Numpy

## • Numpy array indexing

- > import numpy as np
- > t = np.arange(3\*3\*3)
- > t = t.reshape(3, 3, 3) # 3차원 array
- > print(t)
- > print(t[0]) # t의 첫 번째 행렬
- > print(t[0, :])
- > print(t[0, :, :])
- > print(t[:, 0]) # t의 각 행렬의 첫 번째 행
- > print(t[:, 0, :])
- > print(t[:, :, 0]) # t의 각 행렬의 첫 번째 열

t

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

t[0]

0	1	2
3	4	5
6	7	8

t[1]

9	10	11
12	13	14
15	16	17

t[2]

18	19	20
21	22	23
24	25	26

# Numpy

- Numpy array indexing

```
> import numpy as np

> v = np.arange(4)
> print(v[1:3])           # 두 번째 원소부터 세 번째 원소까지, Python container type 변수의 범위 인덱싱과 같이 마지막 인덱스에 주의

> m = np.arange(4*4)
> m = m.reshape(4, 4)
> print(m[1:3, :])        # 행렬의 두 번째 행부터 세 번째 행까지
> print(m[:, 1:3])        # 행렬의 두 번째 열부터 세 번째 열까지

> t = np.arange(4*4*4)
> t = t.reshape(4, 4, 4)
> print(t[1:3, 1:3, 1:3]) # ??
```

# Numpy

## • Numpy array masking

- > import numpy as np
- > m = np.arange(4\*4)
- > m = m.reshape(4, 4)
- > mask = np.eye(4, dtype = bool)    # 단위 행렬로 2차원 마스크 생성, 마스크의 자료형은 반드시 boolean 이어야 함
- > print(m)
- > print(mask)
- > print(m[mask])                    # Array에 마스크 적용

m

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

×

mask

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1



m[mask]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

# Numpy

## • Numpy array masking

- > import numpy as np
- > m = np.arange(4\*4)
- > m = m.reshape(4, 4)
- > mask = np.array((1, 0, 1, 1), dtype = bool)
- > print(m[mask, :]) # 마스크를 행에 적용
- > print(m[:, mask]) # 마스크를 열에 적용

m

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

×

mask

1	0	1	1
---	---	---	---



m[mask, :]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

m[:, mask]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

# Numpy

- 이미지 파일을 Numpy array로 전환하기

```
> import numpy as np                # Numpy 라이브러리 불러오기
> from PIL import Image             # Pillow 라이브러리에서 Image 클래스 불러오기, Image는 대문자로 시작

> image = Image.open('test.png')    # 이미지 파일 읽기
> #image.show()                     # 일반적인 이미지 출력 코드, OS의 이미지 파일 뷰어가 실행됨
> display(image)                    # Jupyter notebook 개발 환경에서 이미지를 바로 출력

> image_array = np.array(image)      # 이미지를 Numpy array로 변환
> print(image_array.shape, image_array.dtype) # 생성된 array의 정보 확인
```

- 일반적인 이미지의 **dimension**은 [width, height, channel]로 이루어짐
- 이미지를 Numpy array로 변환하면 dimension 이 [height, width, channel]순으로 변경됨
- 일반적인 이미지의 각 채널 픽셀 데이터는 **uint8**의 자료형을 가짐
  - unit8 자료형은 0부터 255까지의 정수를 저장할 수 있음

# Numpy

- RGB 이미지를 채널 별로 출력하기

```
> import numpy as np
> from PIL import Image

> image = Image.open('test.png')
> image_array = np.array(image)

> red_array   = image_array[:, :, 0]   # 첫 번째 channel은 red
> green_array = image_array[:, :, 1]   # 두 번째 channel은 green
> blue_array  = image_array[:, :, 2]   # 세 번째 channel은 blue
> alpha_array = image_array[:, :, 3]   # 네 번째 channel은 alpha, PNG 이미지에만 존재

> display(Image.fromarray(red_array))  # Numpy array를 Pillow Image로 변환
> display(Image.fromarray(green_array))
> display(Image.fromarray(blue_array))
> display(Image.fromarray(alpha_array))
```

- RGB 이미지는 **빛의 삼원색**에 따라 3개의 channel로 이루어짐
- PNG 이미지 파일들은 투명도(alpha)를 나타내는 4번째 channel도 있음



# Numpy

- Numpy를 이용해 RGB 이미지를 grayscale 이미지로 전환하기

```
> import numpy as np
> from PIL import Image

> image = Image.open('test.png')
> image_array = np.array(image)

> image_array = image_array.astype(np.float32)
> grayscale_array = (image_array[:, :, 0] + image_array[:, :, 1] + image_array[:, :, 2]) / 3
> grayscale_array = grayscale_array.astype(np.uint8)

> grayscale_image = Image.fromarray(grayscale_array)
> display(grayscale_image)
```

# Overflow를 방지하기 위해 자료형 변환  
# Red, green, blue channel의 평균값  
# 연산 후 원래 자료형으로 변환

- RGB 채널이 모두 같은 값을 가지면 grayscale 이미지가 됨
- 일반적으로 grayscale 이미지는 채널 차원이 없음
  - RGB channel의 평균값을 가지는 단일 channel로 저장함

# Numpy

- Pillow를 이용해 RGB 이미지를 grayscale 이미지로 전환하기

```
> numpy as np
> from PIL import Image

> image = Image.open('test.png')
> grayscale_image = image.convert('L')      # RGB 이미지를 grayscale로 전환

> display(grayscale_image)
> grayscale_image.save('grayscale.jpg', 'JPEG') # Grayscale 이미지를 jpg 형식으로 저장
```

- Pillow는 이미지와 관련된 편리한 기능들을 지원함
- Numpy를 통해 직접 연산을 수행하지 않아도 됨

# Pandas

# 02

# Pandas

- Pandas란?

- CSV, MS 엑셀, JSON 파일 등 **데이터 파일**을 읽고 쓰는데 특화된 라이브러리
- 데이터의 평균, 분산, 누적 등을 구해주는 다양한 **통계** 기능 지원
  - 다양한 기능은 <https://pandas.pydata.org/docs/reference/index.html#api> 에서 확인 가능
- 데이터를 여러 가지 형태로 저장하여 쉽게 관리할 수 있게 함
  - Series : 1차원 array로 저장된 데이터
  - Data frame : 2차원 array인 표(table)로 저장된 데이터
  - Panel : 3차원 array로 저장된 데이터
- 대부분의 데이터는 2차원 array로 저장되므로 **Data frame**을 주로 활용

# Pandas

- Python으로 바이너리 파일 읽기 / 쓰기

```
> input_file = open('test.png', 'rb')      # 파일을 바이너리 읽기 모드로 열기
> output_file = open('output_file.png', 'wb') # 파일을 바이너리 쓰기 모드로 열기

> while True:
>     data = input_file.read(1024)          # 파일을 최대 1024 bytes만큼 읽기
>     if len(data) > 0:                    # 파일에서 읽은 내용이 있다면 data의 길이가 0보다 큼
>         output_file.write(data)          # 파일에 내용 쓰기
>     else:
>         break                            # 파일의 내용을 모두 읽었다면 반복문 종료

> input_file.close()                       # 파일 닫기
> output_file.close()
```

- 파일의 형식에 상관없이, 바이너리 데이터를 그대로 읽고 쓸 수 있음

# Pandas

- Python으로 텍스트 파일 읽기 / 쓰기

```
> input_file = open('test.txt', 'r', encoding = 'UTF8')      # 파일을 유니코드 읽기 모드로 열기
> output_file = open('output_file.txt', 'w', encoding = 'UTF8') # 파일을 유니코드 쓰기 모드로 열기

> while True:
>     line = input_file.readline()                          # 텍스트 파일을 한 줄씩 읽기, 줄 마지막의 개행 문자 포함 됨
>     if len(line) > 0:
>         output_file.write(line)
>     else:
>         break

> input_file.close()
> output_file.close()
```

- 텍스트 파일은 readline()을 통해 줄 단위로 읽을 수 있음
- 텍스트가 유니코드로 이루어져 있을 경우, 인코딩 형식이 유니코드(UTF8)임을 알려줘야 함

# Pandas

- Python으로 CSV 파일 읽기

- CSV (Comma Separated Values)란, **쉼표**를 통해 데이터의 **열**을 구분한 **텍스트** 파일 형식

이름	통솔	무력	지력	정치	매력
유비	75	73	74	78	99
조조	96	72	91	94	96
제갈량	92	38	100	95	92
여포	87	100	26	13	40



이름,통솔,무력,지력,정치,매력

유비,75,73,74,78,99

조조,96,72,91,94,96

제갈량,92,38,100,95,92

여포,87,100,26,13,40

# Pandas

- Python으로 CSV 파일 읽기

```
> input_file = open('test.csv', 'r', encoding = 'UTF8')

> data_dictionary = {}                                # Dictionary 자료형 변수 생성

> while True:
>     line = input_file.readline()                    # CSV 파일을 한 줄씩 읽음, 마지막에 개행 문자 포함 됨
>     if len(line) > 0:
>         line = line.strip()                          # 문자열 앞 뒤에 공백 문자 및 개행 문자 제거
>         line_list = line.split(',')                  # 문자열을 쉼표를 기준으로 자름, list 자료형 반환 됨

>         data_dictionary[line_list[0]] = line_list[1:] # 첫 번째 열을 key로, 나머지는 value로 dictionary에 추가
>     else:
>         break

> input_file.close()
> print(data_dictionary)                             # 열의 이름인 헤더까지 모두 저장된 것을 볼 수 있음
```



# Pandas

- Pandas를 통해 CSV 파일 읽기 / 쓰기

```
> import pandas as pd

> data_frame = pd.read_csv('test.csv', encoding = 'UTF8')
> print(data_frame)
> print(type(data_frame))

> data_frame.to_csv('output_file1.csv', encoding = 'UTF8')
> data_frame.to_csv('output_file2.csv', index = False, encoding = 'UTF8')
```

```
# Pandas 라이브러리를 불러옴, 이후 pd란 이름으로 활용하겠다 선언
```

```
# Pandas를 통해 CSV 파일 읽기
# DataFrame 출력, CSV 파일의 첫 줄을 헤더로 인식
# Pandas의 DataFrame 클래스로 인스턴스로 저장됨
```

```
# Pandas를 통해 DataFrame 클래스 인스턴스를 CSV 파일로 저장
# 정수로 이루어진 첫 번째 index 열은 저장하지 않음
```

# Pandas

- Pandas가 지원하는 파일들의 읽기 / 쓰기

파일 형식	읽기	쓰기
CSV	<code>read_csv('파일 이름')</code>	<code>to_csv('파일 이름')</code>
JSON	<code>read_json('파일 이름')</code>	<code>to_json('파일 이름')</code>
HTML	<code>read_html('파일 이름')</code>	<code>to_html('파일 이름')</code>
Local clipboard	<code>read_clipboard('파일 이름')</code>	<code>to_clipboard('파일 이름')</code>
MS Excel	<code>read_excel('파일 이름')</code>	<code>to_excel('파일 이름')</code>
HDF5	<code>read_hdf('파일 이름')</code>	<code>to_hdf('파일 이름')</code>
SQL	<code>read_sql('파일 이름')</code>	<code>to_sql('파일 이름')</code>

# Pandas

- Pandas를 통해 MS Excel 파일 읽기 / 쓰기

```
> import pandas as pd

> data_frame = pd.read_excel('test.xlsx')
> print(data_frame)
> print(type(data_frame))

# Pandas를 통해 MS 엑셀 파일 읽기
# 첫 행을 헤더로 자동 인식, 첫 열로 행들의 index 추가 됨
# Pandas의 DataFrame 클래스 인스턴스로 저장됨

> data_frame.to_excel('output_file1.xlsx')
> data_frame.to_excel('output_file2.xlsx', index = False)

# Pandas를 통해 DataFrame 클래스 인스턴스를 MS 엑셀 파일로 저장
# 정수로 이루어진 첫 번째 index 열은 저장하지 않음
```

# Pandas

- Pandas를 통해 MS Excel 파일 읽기 / 쓰기

```
> import pandas as pd

> data_frame = pd.read_excel('test.xlsx', index_col = 0) # 첫 번째 열을 각 행들의 이름으로 인식하게 함
> print(data_frame) # 첫 열로 행들의 index 추가 되지 않음
> print(type(data_frame)) # Pandas의 DataFrame 클래스 인스턴스로 저장됨

> data_frame.to_excel('output_file1.xlsx') # Pandas를 통해 DataFrame 클래스 인스턴스를 MS 엑셀 파일로 저장
```

# Pandas

- Pandas DataFrame 생성

- 열 기준으로 생성

```
> import pandas as pd

> data_dictionary = {}
> data_dictionary['이름'] = ('유비', '조조')      # Key를 열 이름으로, value를 열 내용으로 저장
> data_dictionary['통솔'] = (75, 96)
> data_dictionary['무력'] = (73, 72)
> data_dictionary['지력'] = (74, 91)
> data_dictionary['정치'] = (78, 94)
> data_dictionary['매력'] = (99, 96)
> print(data_dictionary, '\n')

> data_frame = pd.DataFrame(data_dictionary) # Dictionary 변수를 DataFrame 클래스 인스턴스로 변환
> print(data_frame, '\n')
```

# Pandas

- Pandas DataFrame 생성

- 행 기준으로 생성

```
> import pandas as pd

> data_dictionary = {}
> data_dictionary['유비'] = (75, 73, 74, 78, 99)
> data_dictionary['조조'] = (96, 72, 91, 94, 96)
> print(data_dictionary, '\n')
```

# 행 단위로 key-value pair 저장

```
> data_frame = pd.DataFrame(data_dictionary)
> print(data_frame, '\n')
```

# 우리가 원하는 형태를 얻으려면 dataframe을 transpose해야 함

```
> data_frame = data_frame.transpose()
> data_frame.columns = ('통술', '무력', '지력', '정치', '매력')
> print(data_frame, '\n')
```

# Transpose 수행, 대입 연산자를 통해 저장해야 함  
# 열 이름 변경

# Pandas

## • Pandas DataFrame 행 추가 / 삭제

```
> import pandas as pd

> data_dictionary = {}
> data_dictionary['유비'] = (75, 73, 74, 78, 99)
> data_dictionary['조조'] = (96, 72, 91, 94, 96)
> data_frame = pd.DataFrame(data_dictionary)
> data_frame = data_frame.transpose()
> data_frame.columns = ('통술', '무력', '지력', '정치', '매력')

> additional_frame = pd.DataFrame(
>     index      = ['제갈량'],
>     columns    = ['통술', '무력', '지력', '정치', '매력'],
>     data       = [(92, 38, 100, 95, 92)]
> )
> print(additional_frame, '\n')
```

# 새로운 DataFrame 생성, columns와 data list의 원소 개수가 같아야 함

```
> data_frame = pd.concat([data_frame, additional_frame], axis = 'index')
> print(data_frame, '\n')
```

# 행 추가, axis = 'index'는 행 단위로 concatenate하라는 뜻

```
> data_frame = data_frame.drop(labels = '조조', axis = 'index')
> print(data_frame, '\n')
```

# 행 제거

# Pandas

## • Pandas DataFrame 열 추가 / 삭제

```
> import pandas as pd

> data_dictionary = {}
> data_dictionary['유비'] = (75, 73, 74, 78, 99)
> data_dictionary['조조'] = (96, 72, 91, 94, 96)
> data_frame = pd.DataFrame(data_dictionary)
> data_frame = data_frame.transpose()
> data_frame.columns = ('통술', '무력', '지력', '정치', '매력')

> additional_frame = pd.DataFrame(
>     index      = ['유비', '조조'],
>     columns    = ['무기'],
>     data       = ['쌍고검', '의천검']
> )
> print(additional_frame, '\n')

> data_frame = pd.concat([data_frame, additional_frame], axis = 'columns')
> print(data_frame, '\n')

> data_frame = data_frame.drop(labels = '무기', axis = 'columns')
> print(data_frame, '\n')
```

# 새로운 DataFrame 생성, index와 data list의 원소 개수가 같아야 함

# 열 추가, axis = 'columns'는 열 단위로 concatenate하라는 뜻

# 열 제거



# Pandas

- Pandas DataFrame 자료형 변환

```
> import pandas as pd  
  
> data_frame = pd.read_excel('test.xlsx', index_col = 0)  
> print(data_frame, 'Wn')  
> print(data_frame.dtypes, 'Wn')           # 각 열의 자료형 출력, dtypes의 s 누락하지 않도록 주의  
  
> data_frame = data_frame.astype('float')   # 모든 열의 자료형 변환  
> print(data_frame.dtypes, 'Wn')
```

- 전체 자료형 변환 실행 시 에러 발생

- 첫 번째 열이 문자열이라 float로 변경 불가

# Pandas

- Pandas DataFrame 자료형 변환

```
> import pandas as pd
> data_frame = pd.read_excel('test.xlsx', index_col = 0)
> print(data_frame, '\n')
> print(data_frame.dtypes, '\n') # 각 열의 자료형 출력, dtypes의 s 누락하지 않도록 주의

> data_frame = data_frame.astype({'통솔' : 'float', '무력' : 'float'}) # 특정 열의 자료형 변환, 중괄호 및 따옴표 누락하지 않도록 주의
> print(data_frame.dtypes, '\n')
```

# Pandas

- Pandas DataFrame 행 / 열 가져오기

```
> import pandas as pd

> data_frame = pd.read_excel('test.xlsx', index_col = 0)
> print(data_frame, '\n')

> print(data_frame['통솔'], '\n')      # 특정 열을 이름으로 가져오기
> print(data_frame.loc['유비'], '\n')  # 특정 행을 이름으로 가져오기
> print(data_frame.iloc[0], '\n')     # 특정 행을 index로 가져오기
```

# Pandas

## • Pandas DataFrame 데이터 수정

```

> import pandas as pd

> data_frame = pd.read_excel('test.xlsx', index_col = 0)
> print(data_frame, '\n')

> data_frame = data_frame.sample(frac = 1)                # 행 단위로 shuffle
> print(data_frame, '\n')

> data_frame = data_frame.sort_values(by = ['통솔', '매력'], ascending = False) # 특정 열들의 값을 내림차순으로 정렬
> print(data_frame, '\n')

> data_frame          = data_frame + 1                    # 모든 데이터에 1을 더함
> data_frame          = data_frame * 0.01                # 모든 데이터에 0.01을 곱함
> data_frame['무력']   = data_frame['무력'] - 0.1         # 특정 열의 모든 데이터에 0.5를 뺌
> data_frame['정치']   = data_frame['정치'] / 2           # 특정 열의 모든 데이터를 2로 나눔
> data_frame.loc['유비'] = data_frame.loc['유비'] + 0.5   # 특정 행의 모든 데이터에 0.5를 더함
> print(data_frame, '\n')
    
```

# Pandas

- Pandas DataFrame 데이터 통계

```
> import pandas as pd

> data_frame = pd.read_excel('test.xlsx', index_col = 0)
> print(data_frame, '\n')

> print(data_frame.count()) # 각 열의 길이 출력
> print(data_frame.mean()) # 각 열의 평균값 출력
> print(data_frame.median()) # 각 열의 중앙값 출력
> print(data_frame.max()) # 각 열의 최대값 출력
> print(data_frame.min()) # 각 열의 최소값 출력
> print(data_frame.std()) # 각 열의 표준편차 출력
```

# Pandas

- Pandas DataFrame 데이터 검색

```
> import pandas as pd

> data_frame = pd.read_excel('test.xlsx', index_col = 0)
> print(data_frame, '\n')

> result_frame = data_frame[data_frame.통솔 > 90]          # 대괄호를 이용한 검색, 열 이름에 따옴표 없음
> print(result_frame, '\n')

> result_frame = data_frame[data_frame.통솔 > data_frame.무력]  # 두 열의 값을 비교하는 검색도 수행 가능
> print(result_frame, '\n')

> result_frame = data_frame.query('통솔 > 무력')            # query()를 통한 검색, 검색 조건 전체를 문자열로 지정
> print(result_frame, '\n')
```

# matplotlib

# 03

# matplotlib

---

- matplotlib란?
  - 데이터 시각화를 전문적으로 다루는 라이브러리
    - 빅데이터를 분석하거나 실험 결과를 표현하는데 굉장히 편리함
    - MS Excel 등의 데이터 관리 툴을 활용하기 어려운 환경에서 유용함
  - 2D 뿐만 아니라 3D 그래프 기능도 지원
  - 애니메이션 기능과, Zoom이나 pan 등의 상호작용 기능을 지원함
    - 다양한 기능은 <https://matplotlib.org/stable/gallery/index.html>에서 확인 가능



# matplotlib

---

- matplotlib로 그래프 표현하기

```
> import matplotlib.pyplot as plt    # matplotlib 그래프 라이브러리
> plt.plot([1, 2, 5, 2, 7, 1, 10, 9]) # 주어진 데이터를 통해 그래프 생성, 기본적으로 선 그래프 생성
> plt.show()                         # 그래프 출력
```

# matplotlib

- matplotlib로 그래프 표현하기

```
> import matplotlib.pyplot as plt

> plt.plot([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'ro-', label = 'List 1')
> plt.plot([5, 4, 3, 2, 1, 2, 3, 4, 5, 6], 'g^-', label = 'List 2')
> plt.plot([2, 4, 6, 8, 10, 12, 14, 16, 18, 20], 'bs-', label = 'List 3')
> plt.plot([8, 6, 10, 12, 14, 18, 16, 2, 4, 20], 'k--', label = 'List 4')

> plt.title('Graph')
> plt.xlabel('Index')
> plt.ylabel('Value')
> plt.legend()

> plt.show()
```

# 데이터 포인트의 모양과 이름 설정

# 그래프 제목  
# 그래프 x축 이름 설정  
# 그래프 y축 이름 설정  
# 그래프 범례 추가

# matplotlib

- matplotlib 그래프를 이미지 파일로 저장

```
> import matplotlib.pyplot as plt

> plt.plot([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'ro-', label = 'List 1')
> plt.plot([5, 4, 3, 2, 1, 2, 3, 4, 5, 6], 'g^-', label = 'List 2')
> plt.plot([2, 4, 6, 8, 10, 12, 14, 16, 18, 20], 'bs-', label = 'List 3')
> plt.plot([8, 6, 10, 12, 14, 18, 16, 2, 4, 20], 'k--', label = 'List 4')

> plt.title('Graph')
> plt.xlabel('Index')
> plt.ylabel('Value')
> plt.legend()

> plt.savefig('outputGraph.png') # 그래프를 이미지 파일로 저장, 반드시 show() 함수 호출 전에 저장해야 함
> plt.show()
```

# matplotlib

- matplotlib로 여러 그래프 동시에 표현하기

```
> import matplotlib.pyplot as plt

> plt.subplot(2, 1, 1)      # 하나의 plot에 subplot 배치, 그래프의 세로 및 가로 grid, 배치 번호
> plt.title('Graph 1')
> plt.plot([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'ro-', label = 'List 1')
> plt.plot([5, 4, 3, 2, 1, 2, 3, 4, 5, 6], 'g^-', label = 'List 2')
> plt.xlabel('Index')
> plt.ylabel('Value')
> plt.legend()

> plt.subplot(2, 1, 2)
> plt.title('Graph 2')
> plt.plot([2, 4, 6, 8, 10, 12, 14, 16, 18, 20], 'bs-', label = 'List 3')
> plt.plot([8, 6, 10, 12, 14, 18, 16, 2, 4, 20], 'k--', label = 'List 4')
> plt.xlabel('Index')
> plt.ylabel('Value')
> plt.legend()

> plt.show()
```

# matplotlib

- matplotlib로 Pandas DataFrame을 그래프 표현하기

```
> import matplotlib.pyplot as plt
> import pandas as pd

> data_frame = pd.read_excel('test.xlsx')
> nameList = list(data_frame['이름'])
> data_frame = data_frame.drop(labels = '이름', axis = 'columns')
> data_frame.index = nameList

> plt.plot(data_frame['통솔'], 'ro-', label = '통솔')
> plt.plot(data_frame['무력'], 'g^-', label = '무력')
> plt.plot(data_frame['지력'], 'bs-', label = '지력')
> plt.legend()
> plt.show()
```

# DataFrame의 열 한 개를 추출하여 데이터로 입력

- matplotlib에서 한글을 표시하기 위해 추가적인 설정 및 코드가 필요함
  - OS 및 환경에 따라 방법이 다르기 때문에, 본 과목에서는 다루지 않음

# matplotlib

---

- matplotlib로 이미지 표시하기

```
> from PIL import Image
> import matplotlib.pyplot as plt

> image = Image.open('test.png')

> plt.imshow(image) # 그래프에 Pillow Image 추가
> plt.axis('off')   # 그래프에 축 제거
> plt.show()        # 그래프 출력
```

# matplotlib

## • matplotlib로 여러 이미지 표시하기

```

> import numpy as np
> from PIL import Image
> import matplotlib.pyplot as plt

> image = Image.open('test.png')
> image_array = np.array(image)

> plt.subplot(1, 4, 1)
> plt.imshow(image_array[:, :, 0], cmap = 'gray')    # 이미지의 첫 번째 채널 이미지 추가, 출력 모드는 grayscale
> plt.axis('off')
> plt.title('Red channel')                          # 제목 추가

> plt.subplot(1, 4, 2)
> plt.imshow(image_array[:, :, 1], cmap = 'gray')
> plt.axis('off')
> plt.title('Green channel')

> plt.subplot(1, 4, 3)
> plt.imshow(image_array[:, :, 2], cmap = 'gray')
> plt.axis('off')
> plt.title('Blue channel')

> plt.subplot(1, 4, 4)
> plt.imshow(image_array[:, :, 3], cmap = 'gray')    # 실제 값은 255로 흰색이 되어야 하지만, matplotlib의 해석 방식에 따라 검은색으로 표시 됨
> plt.axis('off')
> plt.title('Alpha channel')

> plt.show()
    
```

Have a nice day!

