# Python 을 이용한

# 필수 라이브러리

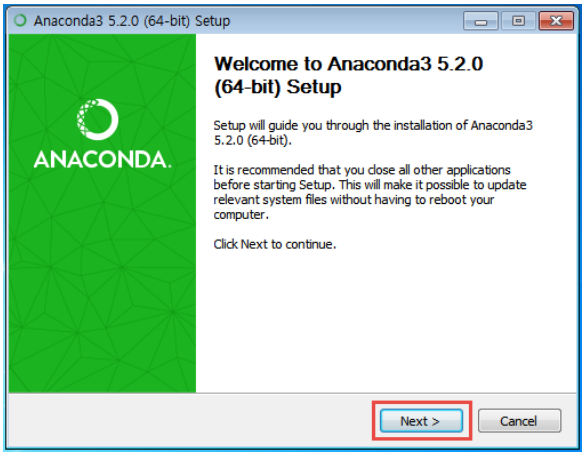
# 개발 환경 구축

1. Anaconda 다운로드 및 설치

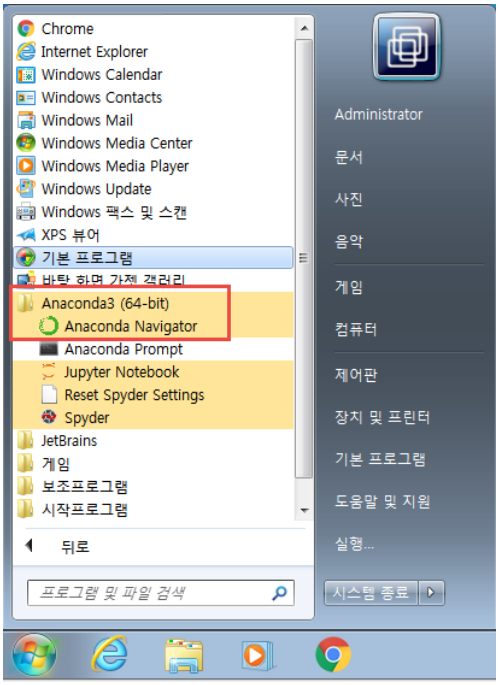
1-1.다운로드 주소

https://repo.continuum.io/archive/Anaconda3-2019.03-Windows-x86_64.exe

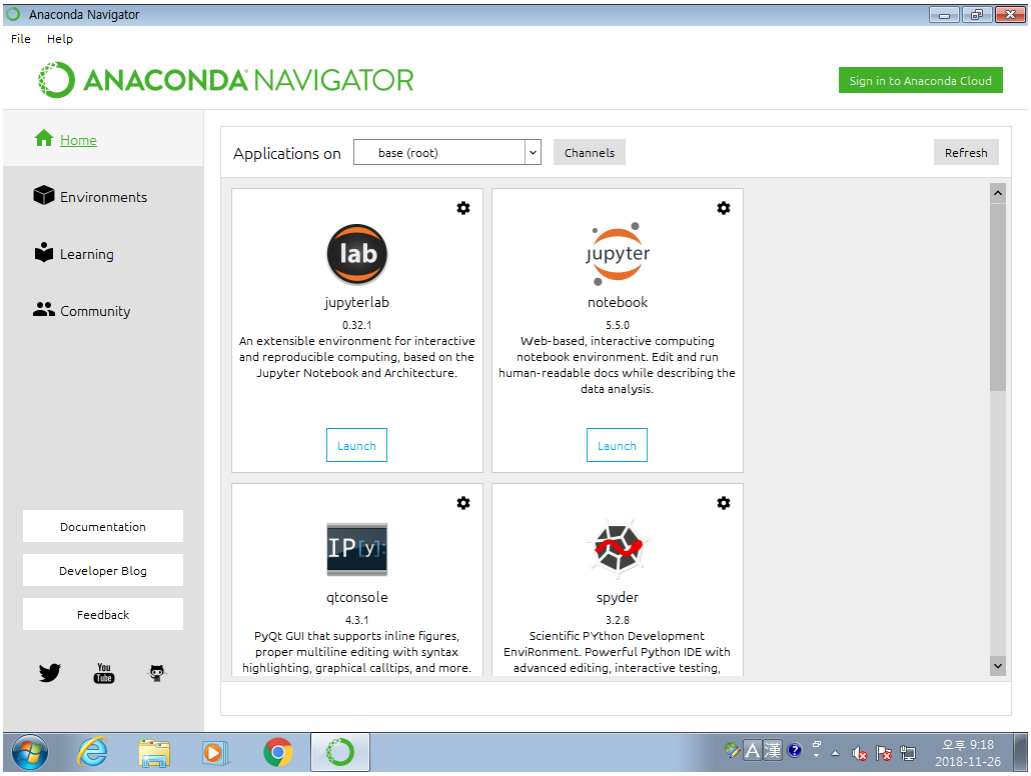
1-2. 설치 : **모두 기본값으로 설치**




1-3. 실행 : [시작] >> [Anaconda3 (64-bit)] >>[Anaconda Navigator]




1-4. 다음과 같이 창이 나오면 설치 완료. 종료할 것.

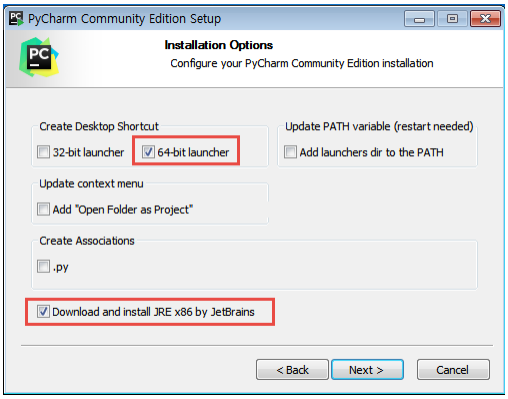



## 2. 파이참 다운로드 및 설치

### 2-1. 다운로드 주소

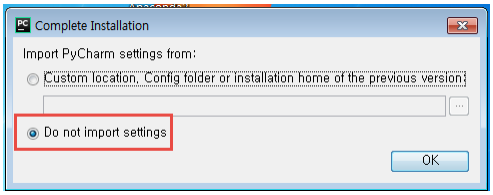
https://download-cf.jetbrains.com/python/pycharm-community-2019.1.exe

### 2-2. 설치시 옵션에서 다음과 같이 체크




### 2-3. 바탕화면의 [JetBrains PyCharm Community Edition 2018.3 x64] 아이콘을 실행

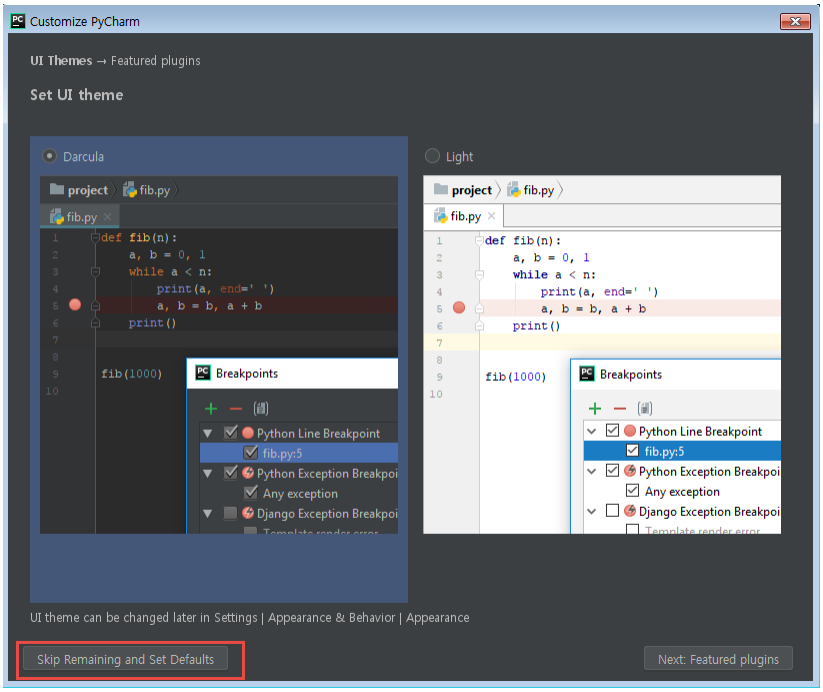
2-4. [Complete Installation] 에서 [Do not import setttings] 선택 후 <OK>



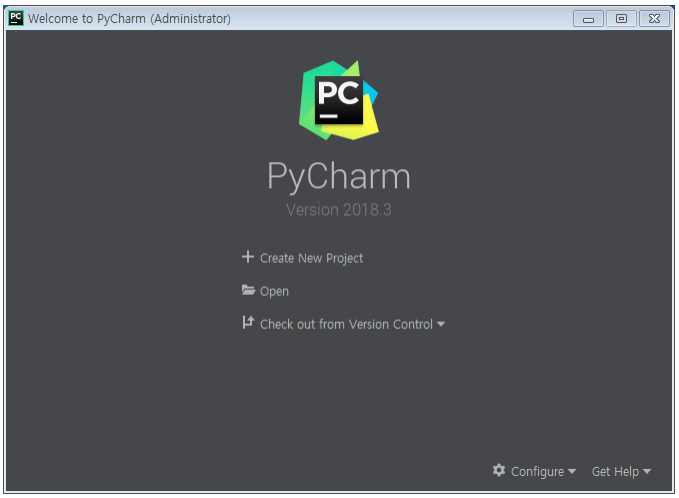

2-5. 라이선스 동의하고 <Continue> 클릭

2-6. [Data Sharing]에서 <Don't send> 클릭

2-7. [Customize Pycharm]에서 <Skip Reaining and Set Defaults> 클릭




2-8. 다음과 같이 창이 나오면 설치 완료. 종료할 것.



# 군집형 자료형 고급 활용

## 리스트 조작 함수

앞에서도 몇 개 살펴봤지만 리스트를 조작하는 함수는 여러 개가 지원된다. 우선 자주 사용되는 함수를 표로 정리해서 확인해 보겠다.

| 함수 | 설명 | 사용법 |
|---|---|---|
| append() | 리스트 제일 뒤에 항목을 추가한다. | 리스트이름.append(값) |
| pop() | 리스트 제일 뒤의 항목을 빼내고, 빼낸 항목은 삭제한다. | 리스트이름.pop() |
| sort() | 리스트의 항목을 정렬한다. | 리스트이름.sort() |
| reverse() | 리스트 항목의 순서를 역순으로 만든다. | 리스트이름.reverse() |
| index() | 지정한 값을 찾아서 그 위치를 반환한다. | 리스트이름.index(찾을 값) |
| insert() | 지정된 위치에 값을 삽입한다. | 리스트이름.insert(위치, 값) |
| remove() | 리스트에서 지정한 값을 제거한다. 단, 지정한 값이 여러 개일 경우 첫번째 값만 지운다. | 리스트이름.remove(지울 값) |
| extend() | 리스트의 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 동일한 기능을 한다. | 리스트이름.extend(추가할 리스트) |
| count() | 리스트에 해당 값의 개수를 센다. | 리스트이름.count(찾을 값) |
| clear() | 리스트의 내용을 모두 제거한다. | 리스트이름.clear() |
| del() | 리스트에서 해당 위치의 항목을 삭제한다. | del(리스트이름[위치]) |
| len() | 리스트에 포함된 전체 항목의 개수를 센다. | len(리스트이름) |
| copy() | 리스트의 내용을 새로운 리스트에 복사한다. | 새리스트=리스트이름.copy() |
| sorted() | 리스트의 항목을 정렬해서 새로운 리스트에 대압한다. | 새리스트=sorted(리스트) |

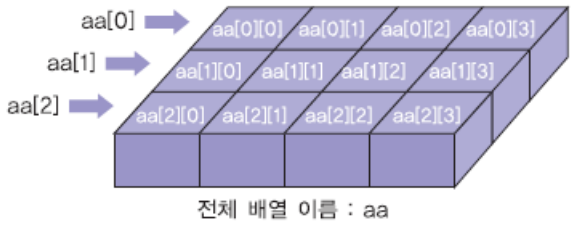
[표 7-1] 리스트 조작 함수

## 2 차원 리스트의 기본 개념

2차원 리스트는 1차원 리스트를 여러 개 연결한 것으로, 두 개의 첨자를 사용하는 리스트다.

```
aa = [ [ 1,   2,   3,   4] ,
       [5,   6,   7,   8] ,
       [9, 10,   11, 12] ]
```

이와 같이 생성하면 3행 4열짜리 리스트가 생성되며 총 요소의 개수는 3x4=12개가 된다.




[그림] 2 차원 리스트의 개념

```
1  list1 = [ ]
2  list2 = [ ]
3  value=1
4  for i in range(0, 3) :
5      for k in range(0, 4) :
6          list1.append(value)
7          value += 1
8      list2.append(list1)
9      list1 = []
10
11 for i in range(0, 3) :
12     for k in range(0, 4) :
13         print("%3d" % list2[i][k], end=" ")
14     print("")
```

*[출력 결과]*

*1  2  3  4*

*5  6  7  8*

*9 10 11 12*

## 딕셔너리의 정렬

딕셔너리안에는 순서가 없다. 순서가 필요하지 않은 이유는 키를 사용해서 추출하기 때문에 굳이 순서대로 있을 필요가 없기 때문이다. 하지만, 종종 딕셔너리 전체를 정렬해서 추출할 경우도 생긴다. 딕셔너리를 정렬하는 방법은 다음 코드처럼 사용하면 된다. 다음 코드는 키로 정렬한 후에 딕셔너리를 추출한 결과다. 하지만, 정렬된 각각의 결과는 튜플로 변경되며 리스트로 반환되는 점에 주의하자.

```
1  import operator
2
3  trainDic, trainList = {}, []
4
5  trainDic = { 'Thomas':'토마스', 'Edward':'에드워드', 'Henry':'헨리', 'Gothen':'고든', 'James':'제임스
   ' }
6  trainList = sorted(trainDic.items(), key=operator.itemgetter(0))
7
8  print(trainList)
```

*[출력 결과]*

*[('Edward', '에드워드'), ('Godon', '고든'), ('Henry', '헨리'), ('James', '제임스'), ('Thomas', '토마스')]*

6행의 operator.itemgetter() 함수를 사용하기 위해서, 1행에서 operator를 임포트 했다. 3행에서 빈 딕셔너리와 리스트를 준비했다. 5행에서 적당히 딕셔너리를 작성하고, 6행에서 딕셔너리를 키에 의해서 정렬했다. 만약 값으로 정렬하려면 operator.itemgetter(1)로 파라미터를 변경하면 된다. 8행의 출력 결과를 보면 리스트 내에 튜플로 딕셔너리의 항목이 변경된 것을 확인할 수 있다.

## 세트(Set)

세트(Set)는 딕셔너리의 특수한 형태로 키만 모아져 있는 것이다. 딕셔너리의 특성상 키는 중복이 되지 않으므로 세트에 들어 있는 값들은 항상 유일하다. 세트를 생성하려면 딕셔너리처럼 중괄호({ })를 사용하지만 콜론이 없이 값들을 입력해야 한다.

```
mySet1 = { 1 , 2 , 3, 3, 3, 4 }
mySet1
```

*[출력 결과]*

*{1, 2, 3, 4}*

## 컴프리헨션

컴프리헨션(Comprehension, 함축)은 순차적인 값을 가진 리스트를 한 줄의 코딩으로 만들기 위한 간단한 방법이다. 우선 지금까지 배운 방식으로 1부터 5까지 저장된 리스트를 다음과 같이 만들 수 있다.

```
numList = []
for num in range(1, 6) :
    numList.append(num)
numList
```

*[출력 결과]*

*[1, 2, 3, 4, 5]*

위 코드를 컴프리헨션으로 다음과 같이 한줄로 작성할 수 있다.

```
numList = [num for num in range(1,6)]
numList
```

*[출력 결과]*

*[1, 2, 3, 4, 5]*

## zip() 함수

zip() 함수는 동시에 여러 개의 리스트에 접근하는 기능을 수행한다. 다음의 간단한 예를 보자.

```
foods = ['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살']
sides = ['오뎅', '단무지', '김치']
for food, side   in   zip (foods, sides) :
    print( food, ' -->', side)
```

*[출력 결과]*

*떡볶이 --> 오뎅*

*짜장면 --> 단무지*

*라면 --> 김치*

결과를 보면 foods 리스트와 sides 리스트의 개수가 다르지만, 개수가 가장 작은 것 까지만 접근한 후에 종료한다.

## 리스트의 복사

리스트를 복사할 때 주의할 점이 있다. 다음 코드를 보자.

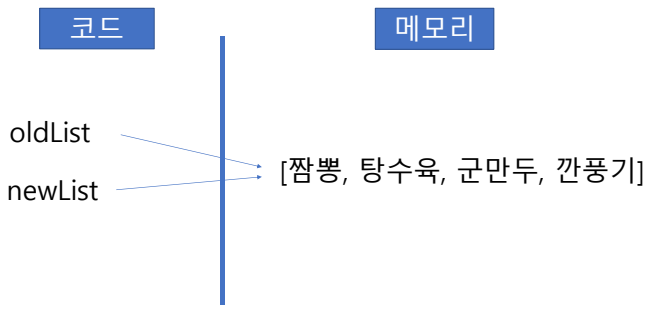
```
oldList = [ '짜장', '탕수육', '군만두' ]
newList = oldList
print(newList)
oldList[0] = '짬뽕'
oldList.append('깐풍기')
print(newList)
```

*[출력 결과]*

*['짜장', '탕수육', '군만두']*

*['짬뽕', '탕수육', '군만두', '깐풍기']*

2행에서 newList를 oldList와 동일하게 대입했다. 그런데, 4행과 5행에서 newList가 아닌 oldList를 변경했는데 결과적으로 newList도 변경된 것이 확인된다. 즉, **oldList = newList** 코드는 다음과 같이 newList와 oldList가 동일한 메모리 공간을 공유하는 것이다.

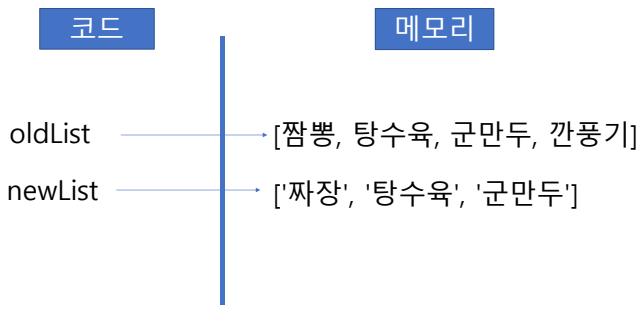



[그림] newList = oldList 코드를 사용한 리스트의 복사 (Shallow Copy)

위와 같은 현상을 얕은 복사(Shallow Copy)라고 부른다. 이를 방지하려면 다음과 같이 코드를 코드를 변경하면 된다.

```
oldList = [ '짜장', '탕수육', '군만두' ]
newList = oldList[:]
print(newList)
oldList[0] = '짬뽕'
oldList.append('깐풍기')
print(newList)
```

**newList = oldList[:]** 코드는 다음 그림과 같이 메모리의 공간을 복사해서 새롭게 만든다. 이를 깊은 복사(Deep Copy)라고 부른다.




[그림] newList = oldList[:] 코드를 사용한 리스트의 복사 (Deep Copy)

프로그래밍 기법상 얕은 복사를 사용하는 경우는 상당히 드물기 때문에, 리스트를 복사할 때는 어떤 복사가 실행되는지 주의해서 사용해야 한다.

# Beautifulsoup 데이터 수집

▶ 크롤러 만들기

1. 크롬 설치

2. urllib 패키지

- 인터넷에서 데이터를 받아오는 기능
- URL을 넣고 실행하면 데이터를 텍스트 형태로 받아옴
- 크롤링 : HTML 텍스트를 받아오는 것
- 파싱 : 필요한 정보를 추출하는 것

3. BeautifulSoup

- 데이터를 추출하는데 필요한 기능이 있는 라이브러리 = 파싱 라이브러리

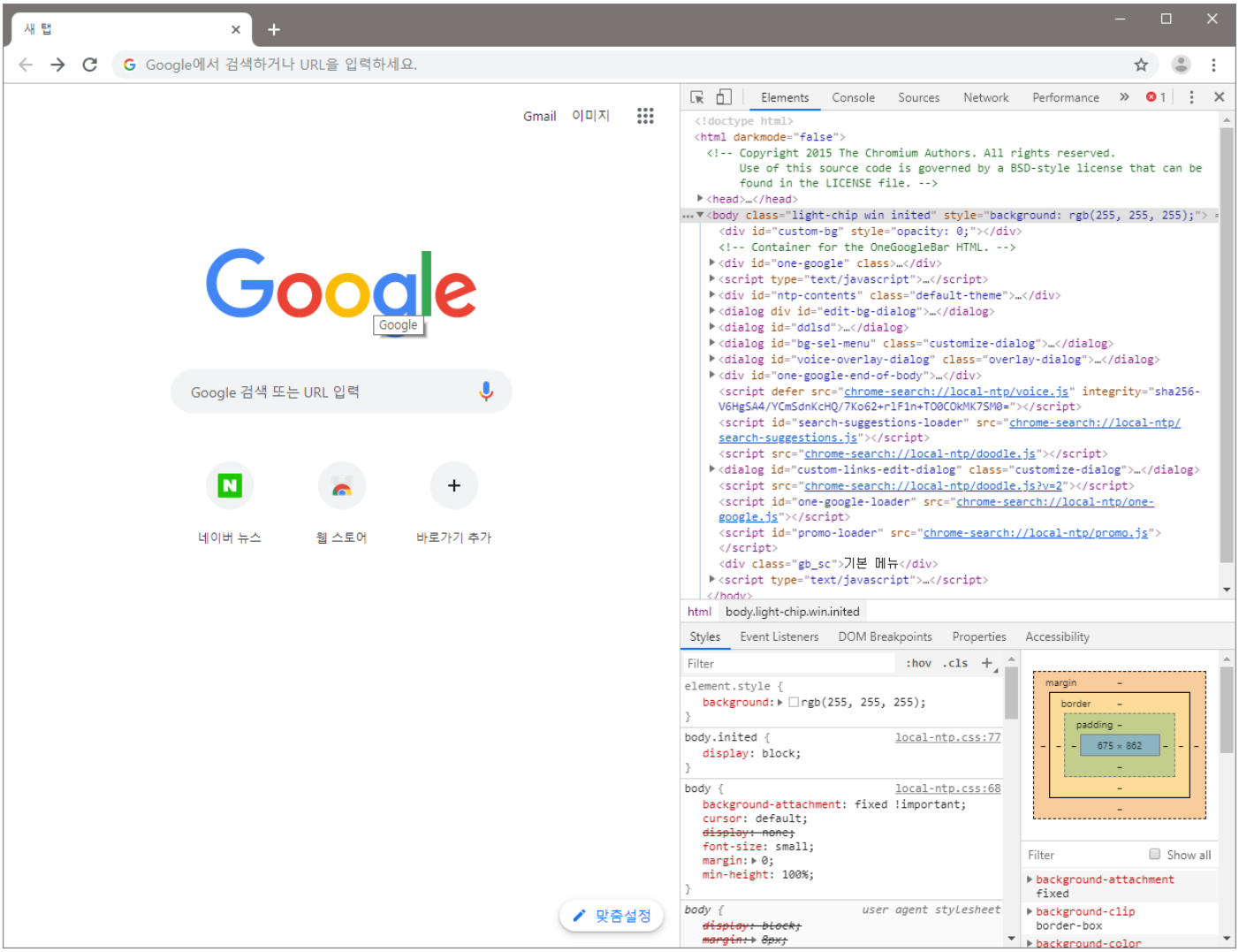
3.1 설치 : 파이참에서 bs4 검색 후 설치

3.2 HTML은 트리구조임

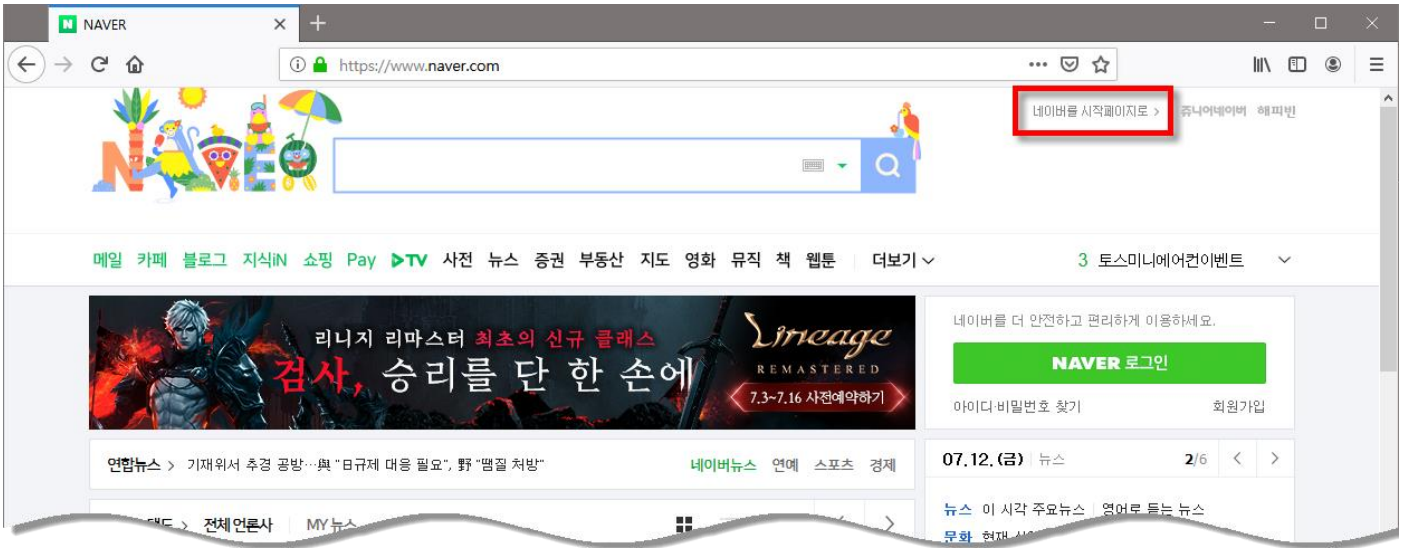
▶ 실습 : Beautifulsoup 연습

```
1  import bs4
2  html_str = """
3  <html>
4      <body>
5          <ul>
6              <li>hello</li>
7              <li>bye</li>
8              <li>welcome</li>
9          </ul>
10     </body>
11 </html>
12 """
13 bs_obj = bs4.BeautifulSoup(html_str, "html.parser")
14
15 ul = bs_obj.find("ul")
16 lis = ul.findAll("li")
17 print(lis[1].text)
```

▶ 실습 : 크롬 개발자 도구

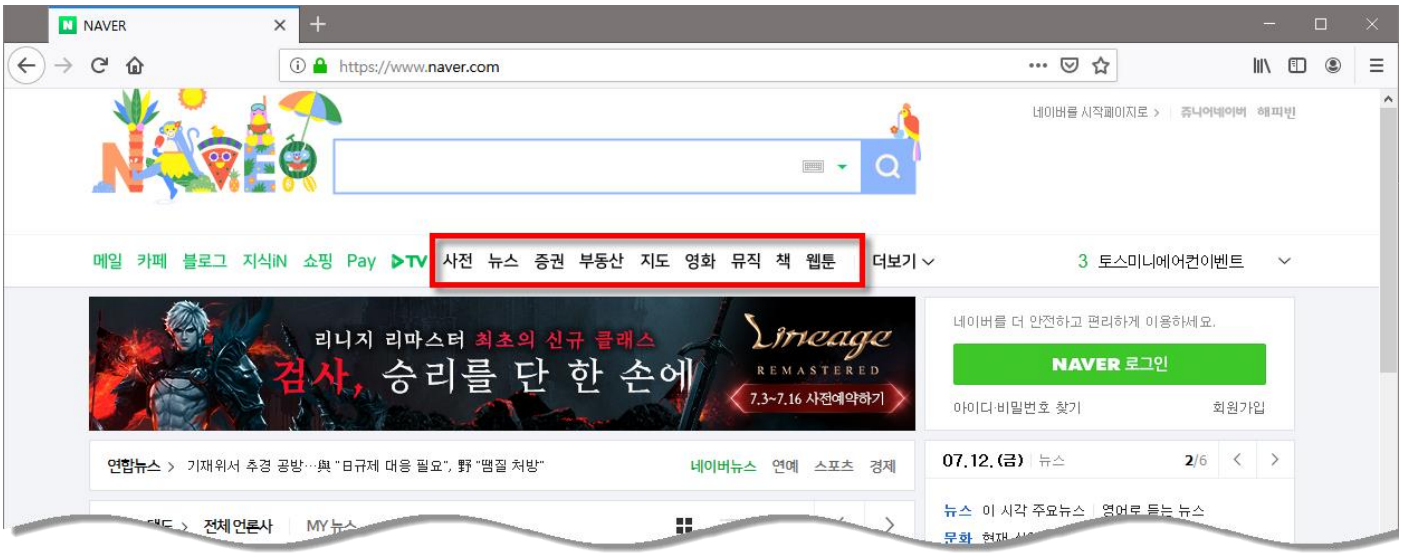



▶ 실습 : 네이버 글자 추출




```
1  import urllib.request
2  import bs4
3
4  url = "https://www.naver.com/"
5  html = urllib.request.urlopen(url)
6
7  bs_obj = bs4.BeautifulSoup(html, "html.parser")
8
9  top_right = bs_obj.find("div", {"class":"area_links"})
10 first_a = top_right.find("a")
11 print(first_a.text)
```

▶ 실습 : 네이버 메뉴 추출



```
1  import urllib.request
2  import bs4
3
4  url = "https://www.naver.com/"
5  html = urllib.request.urlopen(url)
6
7  bs_obj = bs4.BeautifulSoup(html, "html.parser")
8
9  ul = bs_obj.find("ul", {"class":"an_l"})
10
11 lis = ul.findAll("li")
12
13 for li in lis:
14       a_tag = li.find("a")
15       span = a_tag.find("span", {"class":"an_txt"})
16       print(span.text)
```

▶ 실습 : yes24 쇼핑몰 크롤러 1

```
1  import requests
2  import bs4
3
4  url = "http://www.yes24.com/24/Category/Display/001001046001"
5  result = requests.get(url)
6  bs_obj = bs4.BeautifulSoup(result.content, "html.parser")
7
8  ul = bs_obj.find("ul", {"class":"clearfix"})  # 전체 목록
9  boxes = ul.findAll("div", {"class":"goods_name"}) # 각 상품 이름
10
11 for box in boxes:
12       ptag = box.findAll("a")  # a 태그가 여러개임.
13       print(ptag[0].text)
```

천년의 질문 1

천년의 질문 2

천년의 질문 3

~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~

▶ 실습 : yes24 쇼핑몰 크롤러 2 → 함수로 분리

```
1  import requests
2  import bs4
3
4  def get_proudct_info(box):
5      ptag = box.findAll("a")  # a 태그가 여러개임.
6      return ptag[0].text
7
8  url = "http://www.yes24.com/24/Category/Display/001001046001"
9  result = requests.get(url)
10 bs_obj = bs4.BeautifulSoup(result.content, "html.parser")
11
12 ul = bs_obj.find("ul", {"class":"clearfix"})  # 전체 목록
13 boxes = ul.findAll("div", {"class":"goods_name"}) # 각 상품 이름
14
15
16 for box in boxes:
17     product_info = get_proudct_info(box)
18     print(product_info)
```

▶ 실습 : yes24 쇼핑몰 크롤러 3 → 제품 이름, 가격, 링크를 추출 (딕셔너리로 리턴)

```
1  import requests
2  import bs4
3
4  def get_proudct_info(info):
5      names = info.find("div", {"class": "goods_name"})  # 각 상품 이름
6      name = names.findAll("a")[0].text
7      link = names.findAll("a")[0]['href'] # 링크 추출
8      price = info.find("em", {"class": "yes_b"}).text # 각 상품 가격
9      link = "http://www.yes24.com" + link
10     return {"name": name, "price": price, "link":link}
```

```
11
12 url = "http://www.yes24.com/24/Category/Display/001001046001"
13 result = requests.get(url)
14 bs_obj = bs4.BeautifulSoup(result.content, "html.parser")
15
16 ul = bs_obj.find("ul", {"class":"clearfix"})  # 전체 목록
17 infos = ul.findAll("div", {"class":"goods_info"}) # 각 상품 정보
18
19 for info in infos:
20     product_info = get_proudct_info(info)
21     print(product_info)
```

▶ 실습 : yes24 쇼핑몰 크롤러 4 → 모든 페이지 긁어오기

```
1  import requests
2  import bs4
3
4  def get_proudct_info(info):
5      names = info.find("div", {"class": "goods_name"})  # 각 상품 이름
6      name = names.findAll("a")[0].text
7      link = names.findAll("a")[0]['href'] # 링크 추출
8      price = info.find("em", {"class": "yes_b"}).text # 각 상품 가격
9      link = "http://www.yes24.com" + link
10     return {"name": name, "price": price, "link":link}
11
12 def get_page_products(url) :
13     result = requests.get(url)
14     bs_obj = bs4.BeautifulSoup(result.content, "html.parser")
15     ul = bs_obj.find("ul", {"class": "clearfix"})  # 전체 목록
16     infos = ul.findAll("div", {"class": "goods_info"})  # 각 상품 정보
17     product_info_list = [get_proudct_info(info) for info in infos]
18     return product_info_list
19
20 baseUrl = "http://www.yes24.com/24/Category/Display/001001046001?PageNumber="
21
22 num = 1
23 while True:  # 끝페이지를 모르므로 충분히 크게...
24     try:
```

```
25        url = baseUrl +str(num);
26        page_products = get_page_products(url)
27        for page_product in page_products:
28              print(page_product['name'] + "," + page_product['price'])
29        num += 1
30       except:
31        print("끝~~")
```

▶ 실습 : yes24 쇼핑몰 크롤러 5 → 최종 크롤러 완성

```
1  import requests
2  import bs4
3
4  def get_proudct_info(info):
5        names = info.find("div", {"class": "goods_name"})  # 각 상품 이름
6        name = names.findAll("a")[0].text
7        link = names.findAll("a")[0]['href'] # 링크 추출
8        price = info.find("em", {"class": "yes_b"}).text # 각 상품 가격
9        link = "http://www.yes24.com" + link
10       return {"name": name, "price": price, "link":link}
11
12 def get_page_products(url) :
13       result = requests.get(url)
14       bs_obj = bs4.BeautifulSoup(result.content, "html.parser")
15       ul = bs_obj.find("ul", {"class": "clearfix"})  # 전체 목록
16       infos = ul.findAll("div", {"class": "goods_info"})  # 각 상품 정보
17       product_info_list = [get_proudct_info(info) for info in infos]
18       return product_info_list
19
20
21 baseUrl =
   "http://www.yes24.com/24/Category/Display/001001046001?PageNumber="
22
23 import time
24 import datetime
25
26 while True :
27
```

```
28      print("--------------------")
29      now = datetime.datetime.now()
30      print(now.strftime('%Y-%m-%d %H:%M:%S'))
31      print("--------------------")
32
33      for num in range(1, 5):   # 끝페이지를 모르므로 충분히 크게...
34          try:
35           url = baseUrl +str(num);
36           page_products = get_page_products(url)
37           for page_product in page_products:
38                print(page_product['name'] + "," + page_product['price'])
39          except:
40                print("끝~~")
41
42      time.sleep(30)
```

# CSV 데이터 분석

CSV(영어: comma-separated values)는 몇 가지 필드를 쉼표(,)로 구분한 텍스트 데이터 및 텍스트 파일이다. 확장자는 .csv이며 MIME 형식은 text/csv이다. comma-separated variables라고도 한다.

오래전부터 스프레드시트나 데이터베이스 소프트웨어에서 많이 쓰였으나 세부적인 구현은 소프트웨어에 따라 다르다.

비슷한 포맷으로는 탭으로 구분하는 'tab-separated values'(TSV)나, 반각 스페이스로 구분하는 'space-separated values'(SSV) 등이 있으며, 이것들을 합쳐서 character-separated values (CSV), delimiter-separated values라고 부르는 경우가 많다.

▶ 실습 : CSV 데이터 읽기/쓰기 → 문자열로 취급해서 처리

```
1  input_file = "csv/supplier_data.csv"
2  output_file = "csv/output₩₩result02.csv"
3
4  filereader = open(input_file, 'r', newline='')
5  filewriter = open(output_file, 'w', newline='')
6
7  header = filereader.readline()
8  header   = header.strip() # 앞뒤 공백제거
9  header_list = header.split(',')
10 # 리스트를 다시 콤마(,)로 분리된 문자열로 만들고 싶다.
11 header_str = ','.join(map(str, header_list))
12 filewriter.write(header_str + '₩n')
13 for  row  in  filereader :  # 모든행은 row 에 넣고 돌리기.
14       row = row.strip()
15       row_list = row.split(',')
16       row_str = ','.join(map(str, row_list))
17       print(row_str)
18       filewriter.write(row_str + '₩n')
19
20 filereader.close()
21 filewriter.close()
22 print('Ok~~')
```

▶ 실습 : CSV 데이터 읽기/쓰기 → 내장 라이브러리 사용

```
1  import csv # 내장 라이브러리
2
3  input_file = 'csv/supplier_data.csv'
4  output_file = 'csv/output_files/supplier_data2.csv'
5
6  with   open(input_file, 'r', newline='') as csv_in_file :
7        with open(output_file, 'w', newline='') as csv_out_file :
8              filereader = csv.reader(csv_in_file, delimiter=',')
9              filewriter = csv.writer(csv_out_file, delimiter=',')
10
11             for row_list in filereader :
12                   filewriter.writerow(row_list)
```

▶ 실습 : $600 이상인 데이터만 뽑아서, 10% 상승 시킨 후에 저장하기.

```
1  import csv # 내장 라이브러리
2
3  input_file = 'csv/supplier_data.csv'
4  output_file = 'csv/output_files/supplier_data3.csv'
5
6  with   open(input_file, 'r', newline='') as csv_in_file :
7        with open(output_file, 'w', newline='') as csv_out_file :
8              filereader = csv.reader(csv_in_file, delimiter=',')
9              filewriter = csv.writer(csv_out_file, delimiter=',')
10             # 헤더 처리
11             header = next(filereader)
12             filewriter.writerow(header)
13             # 데이터 행들 처리
14             for row_list in filereader :
15                   cost = float(row_list[3].strip().strip('$')) # "$500.00" --> "500.00" --> 500.00
16                   if cost >= 600 :
17                         cost = cost + cost*0.1
18                         cost_str = "${0:.2f}".format(cost)
19                         row_list[3] = cost_str
```

```
20                      filewriter.writerow(row_list)
21  print('끝~~')
```

# JSON 데이터 분석

JSON(제이슨, JavaScript Object Notation)은 속성-값 쌍 또는 "키-값 쌍"으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷이다. 비동기 브라우저/서버 통신 (AJAX)을 위해, 넓게는 XML(AJAX가 사용)을 대체하는 주요 데이터 포맷이다. 특히, 인터넷에서 자료를 주고 받을 때 그 자료를 표현하는 방법으로 알려져 있다. 자료의 종류에 큰 제한은 없으며, 특히 컴퓨터 프로그램의 변수값을 표현하는 데 적합하다.

본래는 자바스크립트 언어로부터 파생되어 자바스크립트의 구문 형식을 따르지만 언어 독립형 데이터 포맷이다. 즉, 프로그래밍 언어나 플랫폼에 독립적이므로, 구문 분석 및 JSON 데이터 생성을 위한 코드는 C, C++, C#, 자바, 자바스크립트, 펄, 파이썬 등 수많은 프로그래밍 언어에서 쉽게 이용할 수 있다.

JSON의 파일 확장자는 .json이다.

```
{
    "이름": "홍길동",
    "나이": 25,
    "성별": "여",
    "주소": "서울특별시 양천구 목동",
    "특기": ["농구", "도술"],
    "가족관계": {"#": 2, "아버지": "홍판서", "어머니": "춘섬"},
    "회사": "경기 수원시 팔달구 우만동"
}
```

▶ 실습 : Json 읽기

```
import json
jsonDic = {}
jsonList = []
csvList = []

filereader = open('TEST01.json', 'r', encoding='utf-8')
jsonDic = json.load(filereader)
```

```
csvName = list(jsonDic.keys())
jsonList = jsonDic[ csvName[0]]
# 헤더 추출
header_list = list(jsonList[0].keys())
csvList.append(header_list)
# 행들 추출
for   tmpDic in jsonList :
      tmpList = []
      for   header in   header_list :
            data = tmpDic[header]
            tmpList.append(data)
      csvList.append(tmpList)
print(csvList)

filereader.close()
```

TEST01.json

```
{
   "userTable" :
            [
               { "ID" : "AAA", "NAME":"에이", "ADDR": "서울"},
               { "ID" : "BBB", "NAME":"삐이", "ADDR": "경기"},
               { "ID" : "CCC", "NAME":"씨이", "ADDR": "강원"}
            ]
}
```

# Excel 데이터 처리

xlrd / xlwt / xlswriter 등의 외부 라이브러리를 사용해서 엑셀 파일을 처리할 수 있다.

▶ 실습 : 엑셀파일 읽기

```
1  import xlrd
2
3  input_file = 'csv/sales_2013.xlsx'
4
5  workbook = xlrd.open_workbook(input_file)
6
7  sheetCount = workbook.nsheets # 속성
8
9  for   worksheet   in workbook.sheets() :
10       sheetName = worksheet.name
11       sRow = worksheet.nrows
12       sCol = worksheet.ncols
13       print(sheetName, sRow, sCol)
```

▶ 실습 : 엑셀파일 쓰기

```
1  from xlrd import open_workbook
2  from xlwt import Workbook
3
4  inFile = "d:/pyData/sales_2013.xlsx"
5  outFile = "d:/pyData/output/20_2.xls"
6
7  outWorkbook = Workbook()
8  outWorksheet = outWorkbook.add_sheet('2013_1')
9
10 with open_workbook(inFile) as workbook :
11       worksheet = workbook.sheet_by_name('january_2013')
12       for rowNum in range(worksheet.nrows) :
13            for colNum in range(worksheet.ncols) :
14                 cValue = worksheet.cell_value(rowNum, colNum)
15                 outWorksheet.write(rowNum, colNum, cValue)
16                 print(cValue, end='₩t')
17            print()
18       outWorkbook.save(outFile)
```

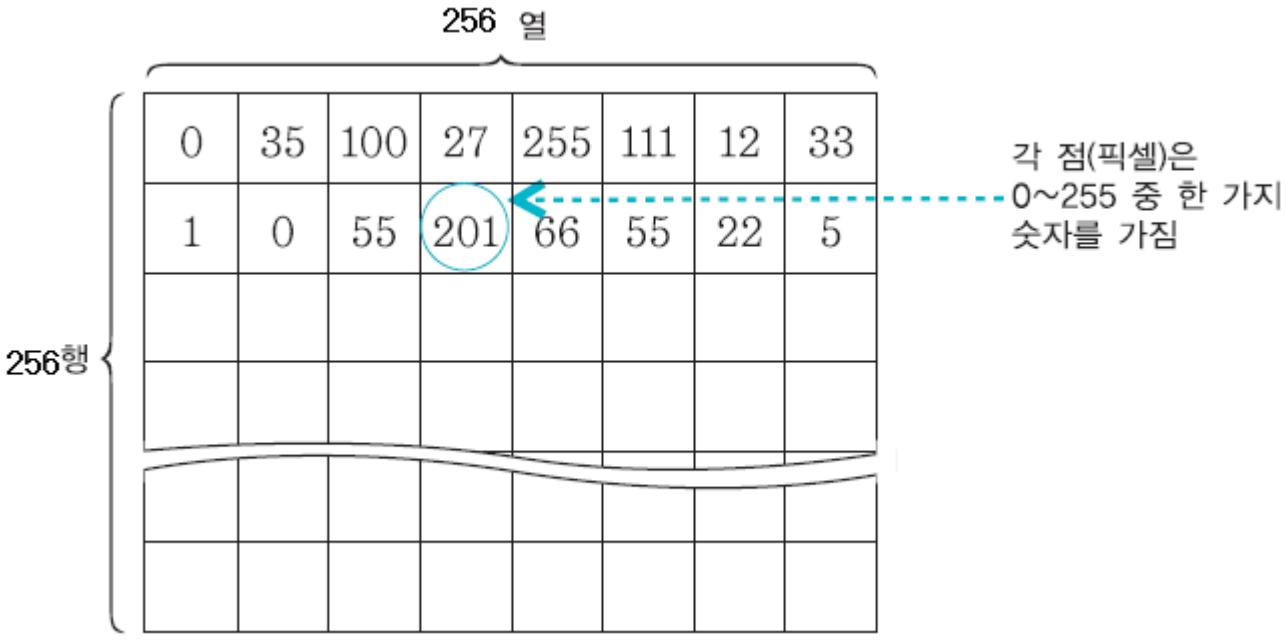
# 이미지 데이터 분석

## RAW 사진 파일의 구성 이해

처리할 이미지 파일이 GIF, JPG, BMP 등의 칼라 사진이면 좋겠지만, 이 파일들의 내부 구조가 상당히 복잡하고 어렵다.

가장 단순한 사진 파일 형태인 **RAW 사진 파일**을 사용하겠다. 예를 들어서 사용할 *.raw 파일은 256x256 픽셀 크기의 흑백 이미지 이다. 정확히는 0과 1로만 나뉘는 흑백(Black-White) 이미지가 아니라, 0부터 255까지 밝기로 구분되는 그레이(Gray) 이미지이다.

RAW 사진 파일의 구조를 들여다 보면 다음과 같다.




[그림 11-1] raw 사진 파일 구조

정방형 크기의 사진 파일에서 각각의 점(픽셀)은 0~255의 값을 갖다. 픽셀의 값이 0에 가까울수록 검정색에 가까운 회색이 되고, 255에 가까울수록 흰색에 가까운 회색이 된다. 예로 모든 값이 0으로 채워지면 이 사진은 완전히 까만 이미지가 되고, 모든 값이 255로 채워지면 아무것도 없는 흰색 사진으로 보인다.

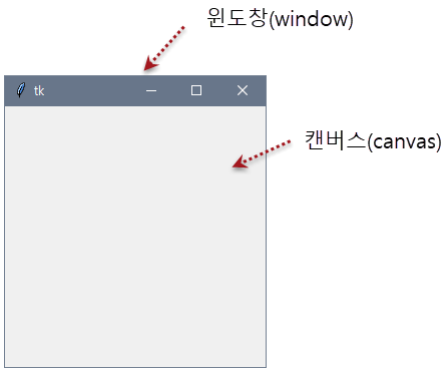
### 윈도 창의 작성

기본적인 윈도창을 띄우는 간단한 코드를 살펴 보자.

▶ 실습 : 윈도창 기본

```
1  from tkinter import *
2
3  ## 변수 ##
4  window = None
5  canvas = None
6  XSIZE, YSIZE=256,256
7
8  ## 메인 코드 ##
9  window=Tk()
10 canvas = Canvas(window, height=XSIZE, width=YSIZE)
11
12 canvas.pack()
13 window.mainloop()
```

*[출력 결과]*



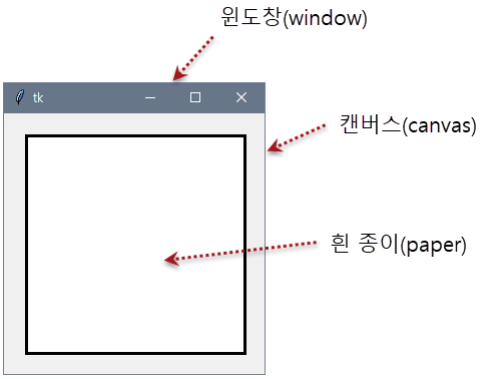

그런데, 캔버스에는 직접 선을 그리는 작업은 할 수 있지만, RAW이미지를 출력할 수는 없다. RAW이미지는 [256x256개의 점을 각각 찍어주는 방식으로 그려야 한다.

그래서, 11행에 다음의 두 행을 추가로 코딩할 것이다. paper 변수는 캔버스 위에 흰색 종이를 한 장 붙인 것으로

생각하면 된다. 그리고, 잠시 후에 이 종이(paper)에 256x256개의 점을 찍어서 사진을 화면에 출력할 것이다..

```
paper=PhotoImage(width=XSIZE, height=YSIZE)
canvas.create_image ( (XSIZE/2, YSIZE/2), image=paper, state="normal")
```

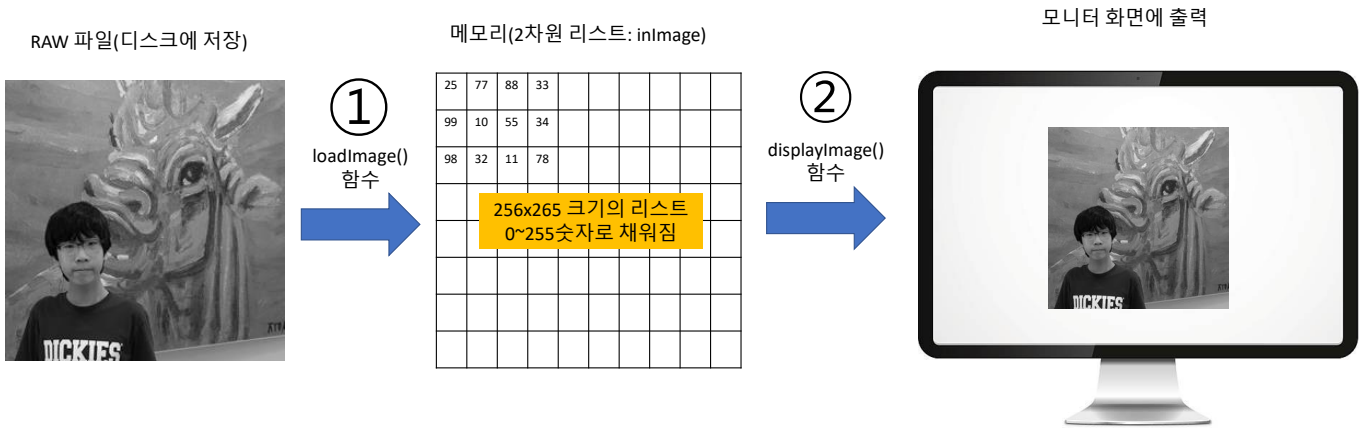
위 코드를 추가하면 다음 그림과 같은 개념이 같이 된다. 물론 아직은 점을 찍은 적이 없어서 위코드의 결과와 겉모양은 동일하게 보인다




[그림] 흰 종이를 붙인 개념의 윈도창

### RAW 파일의 화면 출력

RAW 파일을 읽어서 메모리에 저장해 보겠다. 메모리 변수는 2차원 리스트로 구현하면 된다. 개념적으로 다음과 같이 된다.




[그림] RAW 파일을 메모리로 읽은 후 화면에 출력하는 개념

▶ 실습 : 윈도창 기본 + 메뉴, 대화상자

```
1  from tkinter import *
2  from   tkinter.filedialog import *
3  from   tkinter.simpledialog import *
4  from   tkinter.messagebox import *
5
6  def openFile() :
7      filename = askopenfilename(parent=window,
8                                 filetypes=(("RAW 파일", "*.raw"), ("모든파일", "*.*")))
9      messagebox.showinfo('',filename)
10
11 ## 변수 선언 부분 ##
12 window = None
13 canvas = None
14 XSIZE, YSIZE = 256, 256
15
16 ## 메인 코드 부분 ##
17 window = Tk()
18 window.title('CJ 파이썬 라이브러리')
19 canvas = Canvas(window, height = XSIZE, width = YSIZE)
20
21 mainMenu = Menu(window);window.config(menu=mainMenu)
22 fileMenu = Menu(mainMenu);mainMenu.add_cascade(label='파일', menu=fileMenu)
23 fileMenu.add_command(label='열기', command=openFile)
24 fileMenu.add_command(label='저장', command=None)
25 fileMenu.add_separator()
26 fileMenu.add_command(label='종료', command=None)
27
28 canvas.pack()
29 window.mainloop()
```

▶ 실습 : RAW 데이터 뷰어

```
1  from tkinter import *
2  from   tkinter.filedialog import *
3  from   tkinter.simpledialog import *
4  from   tkinter.messagebox import *
5  import math
```

```
import numpy as np

### 메모리를 확보해서 반환하는 함수
def alloc2DMemory(height, width) :
    retMemory = [];    tmpList = []
    for i in range(height):  # 출력메모리 확보(0 으로 초기화)
        tmpList = []
        for k in range(width):
            tmpList.append(0)
        retMemory.append(tmpList)
    return retMemory

def loadImage(fname) :
    global window, canvas, paper, filename, inImage, outImage, inW, inH, outW, outH
    fsize = os.path.getsize(fname) # 파일 크기 확인
    inH = inW = int(math.sqrt(fsize))  # 입력메모리 크기 결정! (중요)
    inImage = []; tmpList = []
    # 메모리 할당
    inImage= None
    inImage = alloc2DMemory(inH, inW)

    print(len(inImage))

    # 파일 --> 메모리로 데이터 로딩
    fp = open(fname, 'rb') # 파일 열기(바이너리 모드)
    for  i  in range(inH) :
        for  k  in  range(inW) :
            inImage[i][k] =  int(ord(fp.read(1)))
    fp.close()

def openFile() :
    global window, canvas, paper, filename,inImage, outImage,inW, inH, outW, outH
    filename = askopenfilename(parent=window,
                               filetypes=(("RAW 파일", "*.raw"), ("모든파일", "*.*")))
    loadImage(filename) # 파일 --> 입력메모리
    equal() # 입력메모리--> 출력메모리

def display() :
```

```
    global window, canvas, paper, filename, inImage, outImage, inW, inH, outW, outH,
VIEW_X, VIEW_Y
    # 기존에 캐버스 있으면 뜯어내기.
    if canvas != None:
        canvas.destroy()
    # 화면 준비 (고정됨)
    VIEW_X, VIEW_Y = 512, 512
    if VIEW_X >= outW or VIEW_Y >= outH:  # 영상이 256 미만이면
        VIEW_X = outW
        VIEW_Y = outH
        step = 1  # 건너뛸숫자
    else:
        step = outW / VIEW_X  # step 을 실수도 인정. 128, 256, 512 단위가 아닌 것 고려.

    window.geometry(str(int(VIEW_X * 1.2)) + 'x' + str(int(VIEW_Y * 1.2)))
    canvas = Canvas(window, width=VIEW_X, height=VIEW_Y)
    paper = PhotoImage(width=VIEW_X, height=VIEW_Y)
    canvas.create_image((VIEW_X / 2, VIEW_X / 2), image=paper, state='normal')

    # 화면에 출력. 실수 step 을 위해서 numpy 사용
    rgbString = ''  # 여기에 전체 픽셀 문자열을 저장할 계획
    for i in np.arange(0, outH, step):
        tmpString = ''
        for k in np.arange(0, outW, step):
            i = int(i);                k = int(k)  # 첨자이므로 정수화
            r = g = b = outImage[i][k];
            tmpString += ' #%02x%02x%02x' % (r, g, b)
        rgbString += '{' + tmpString + '} '
    paper.put(rgbString)

    canvas.pack(expand=1, anchor=CENTER)
    status.configure(text='이미지 정보:' + str(outW) + 'x' + str(outH))

def equal() :  # 동일 영상 알고리즘
    global window, canvas, paper, filename, inImage, outImage, inW, inH, outW, outH
    # 중요! 출력메모리의 크기를 결정
    outW = inW;  outH = inH;
    # 메모리 할당
```

```
    outImage= None
    outImage = alloc2DMemory(outH, outW)

    ##############################
    # 진짜 영상처리 알고리즘을 구현
    ##############################
    for  i  in  range(inH) :
        for  k  in  range(inW) :
            outImage[i][k] = inImage[i][k]

    display()

import time
def addImage() :  # 밝게하기 알고리즘
    global window, canvas, paper, filename, inImage, outImage, inW, inH, outW, outH
    # 중요! 출력메모리의 크기를 결정
    outW = inW;  outH = inH;
    # 메모리 할당
    outImage= None
    outImage = alloc2DMemory(outH, outW)
    ##############################
    # 진짜 영상처리 알고리즘을 구현
    ##############################
    value = askinteger('밝게/어둡게하기', '밝게/어둡게 할 값-->', minvalue=-255,
maxvalue=255)
    startTime = time.time()
    for  i  in  range(inH) :
        for  k  in  range(inW) :
            if inImage[i][k] + value > 255 :
                outImage[i][k] = 255
            else :
                outImage[i][k] = inImage[i][k] + value

    seconds = time.time() - startTime  # 초단위
    display()
    status.configure(text=status.cget("text") + '₩t₩t 걸린시간(초):' + "{0:.2f}".format(seconds))


```

```
118 ## 변수 선언 부분 ##
119 window = None
120 canvas = None
121 XSIZE, YSIZE = 256, 256
122
123 ## 메인 코드 부분 ##
124 window = Tk()
125 window.title('CJ 파이썬 라이브러리')
126 window.geometry('500x500')
127 canvas = Canvas(window, height = XSIZE, width = YSIZE)
128
129 status = Label(window, text='이미지 정보:', bd=1, relief=SUNKEN, anchor=W)
130 status.pack(side=BOTTOM, fill=X)
131
132 mainMenu = Menu(window);window.config(menu=mainMenu)
133 fileMenu = Menu(mainMenu);mainMenu.add_cascade(label='파일', menu=fileMenu)
134 fileMenu.add_command(label='열기', command=openFile)
135 fileMenu.add_command(label='저장', command=None)
136 fileMenu.add_separator()
137 fileMenu.add_command(label='종료', command=None)
138
139 pixelMenu = Menu(mainMenu);mainMenu.add_cascade(label='화소점처리', menu=pixelMenu)
140 pixelMenu.add_command(label='동일영상', command=equal)
141 pixelMenu.add_command(label='밝게/어둡게', command=addImage)
142
143 canvas.pack()
144 window.mainloop()
```

▶ 실습 : RAW → CSV  1 (Histogram )

```
1 # RAW image --> CSV File
2 import os
3 import math
4
5 input_file = "DATA/RAW/64/LENA64.RAW"
6 output_file = "DATA/output/LENA64.CSV"
7 header = "Column, Row, Value"
8
```

```
stat = [0]*255

with open(input_file, 'rb') as filereader:
    with open(output_file, 'w', newline='') as filewriter:
        header_list = header.split(',')
        print(header_list)
        filewriter.write(','.join(map(str,header_list))+'₩n')
        fsize = os.path.getsize(input_file)
        XSIZE = YSIZE = int(math.sqrt(fsize))
        for row in range(XSIZE):
            for col in range(YSIZE):
                data = int(ord(filereader.read(1)))
                stat[data] += 1
                row_list = [row, col, data]
                filewriter.write(','.join(map(str,row_list))+'₩n')
        print(stat)
```

▶ 실습 : RAW → CSV  2 (순서 섞기)

```
# RAW image --> CSV File
# 순서 섞기

import os
import math
import random

input_file = "data/RAW/64/LENA64.RAW"
output_file = "data/output/LENA64_2.CSV"
header = "Column, Row, Value"

rawFileList = []
with open(input_file, 'rb') as filereader:
    with open(output_file, 'w') as filewriter:
        header_list = header.split(',')
        print(header_list)
        filewriter.write(','.join(map(str,header_list))+'₩n')
        fsize = os.path.getsize(input_file)
        XSIZE = YSIZE = int(math.sqrt(fsize))
```

```
20          for row in range(XSIZE):
21              for col in range(YSIZE):
22                  data = int(ord(filereader.read(1)))
23                  row_list = [row, col, data]
24                  rawFileList.append(row_list)
25
26          # rawFileList 섞기
27          random.shuffle(rawFileList)
28
29          # File 쓰기
30          for row in rawFileList :
31              filewriter.write(','.join(map(str,row))+'₩n')
```

▶ 실습 : 순서섞인 CSV → RAW

```
1  # Random 순서의 CSV --> Raw
2  # 메모리에 로딩 후 Write
3  import csv
4  import math
5
6  input_file = "data/output/LENA64_2.CSV"
7  output_file = "data/output/LENA64_2.raw"
8
9  with open(input_file, 'r', newline='') as csv_in_file:
10     with open(output_file, 'wb') as filewriter:
11         filereader = csv.reader(csv_in_file, delimiter=',')
12         header = next(filereader)
13
14         # 행 개수 카운트
15         num_lines = sum(1 for line in filereader)
16         print(num_lines)
17         csv_in_file.seek (0,0)
18         XSIZE = YSIZE = int(math.sqrt(num_lines))
19         header = next(filereader)
20
21         # 메모리 할당
22         rawList = []
23         print(XSIZE, YSIZE)
```

```
24          for _ in range(XSIZE) :
25              tmpList = []
26              for _ in range(YSIZE) :
27                  tmpList.append(0)
28              rawList.append(tmpList)
29
30          # 해당 행,열에 값 대입
31          for row_list in filereader:
32              row = int(row_list[0])
33              col = int(row_list[1])
34              rawList[row][col] = int(row_list[2])
35
36          # 파일에 쓰기
37          for row in range(XSIZE) :
38              for col in range(YSIZE) :
39                  byteVal = bytes([rawList[row][col]])
40                  filewriter.write(byteVal)
```

▶ 실습 : RAW → Excel (숫자)

```
1  from xlwt import Workbook
2  import os
3  import math
4
5  input_file = "data/RAW/64/LENA64.RAW"
6  output_file = "data/output/LENA64.xls"
7
8  output_workbook = Workbook()
9  output_worksheet = output_workbook.add_sheet('LENA64')
10
11 with open(input_file, 'rb') as filereader:
12     fsize = os.path.getsize(input_file)
13     XSIZE = YSIZE = int(math.sqrt(fsize))
14
15     for row in range(XSIZE):
16         for col in range(YSIZE):
17             data = int(ord(filereader.read(1)))
18             output_worksheet.write(row, col, data)
```

```
19
20 output_workbook.save(output_file)
```

▶ 실습 : RAW → Excel (셀 서식)

```
 1 from xlsxwriter import Workbook
 2 import os
 3 import math
 4
 5 input_file = "data/RAW/256/LENA256.RAW"
 6 output_workbook = Workbook("data/output/LENA256#2.xlsx")
 7 output_worksheet = output_workbook.add_worksheet('LENA256')
 8
 9 with open(input_file, 'rb') as filereader:
10     fsize = os.path.getsize(input_file)
11     XSIZE = YSIZE = int(math.sqrt(fsize))
12
13     output_worksheet.set_column(0, XSIZE, 1.0 ) # 열 너비 지정 (약 0.34)
14     # 행 높이 지정
15     for row in range(YSIZE) :
16         output_worksheet.set_row(row, 9.5) # 행 높이 지정(약 0.35)
17
18     for row in range(XSIZE):
19         for col in range(YSIZE):
20             data = int(ord(filereader.read(1)))
21             if data <= 15 :  #'#FFFFFF' 형태로 만듬
22                 hexStr = '#' + ('0'+hex(data)[2:]) * 3
23             else :
24                 hexStr = '#' + hex(data)[2:]*3
25
26             cell_format = output_workbook.add_format() # 포맷
27             cell_format.set_bg_color(hexStr)
28             output_worksheet.write(row, col, '' , cell_format)
29
30 output_workbook.close()
```

# 데이터베이스 처리 및 활용

MySQL 데이터베이스와 Python의 연동을 다룬다. pymysql을 설치해야 한다.

# CSV File --> MySQL Database

# DB Server : 192.168.111.200. MySQL 5.x. pyUser/1234, pyDB

# root 로 접속 후,

#  GRANT ALL PRIVILEGES ON pyDB.* TO pyUser@'%' IDENTIFIED BY '1234';

#  CREATE DATABASE pyDB;

#  EXIT

#

▶ 실습 : RAW → MySQL

```
1  import os
2  import math
3  import pymysql
4  import csv
5
6  # MySQL Connection 연결
7  conn = pymysql.connect(host='192.168.111.200', user='pyUser', password='1234',  db='pyDB', charset='utf8')
8  # Connection 으로부터 Cursor 생성
9  curs = conn.cursor()
10 # SQL 문 실행 (Table 삭제 후, 재생성)
11 sql = "DROP TABLE IF EXISTS lena64;"
12 curs.execute(sql)
13 sql = "CREATE TABLE lena64 ( row INT, col INT, data INT)"
14 curs.execute(sql)
15
16 input_file = "F:/output/LENA64.CSV"
17
18 with open(input_file, 'r', newline='') as csv_in_file:
19     filereader = csv.reader(csv_in_file)
```

```
20      header = next(filereader)
21      for row_list in filereader:
22          rowVal, colVal, dataVal = row_list
23          #sql = "INSERT INTO lena64 VALUES (" + str(rowVal) + "," + str(colVal) + "," +
    str(dataVal) + ")"
24          #curs.execute(sql)
25          sql = "INSERT INTO lena64 VALUES (?,?,?)"
26          curs.execute(sql, row_list)
27
28  # Connection 닫기
29  conn.close()
```

▶ 실습 : MySQL (Histogram)

```
 1  import pymysql
 2  from tkinter import *
 3  import os
 4  import math
 5
 6  # (1) 데이터베이스 연결
 7  con = pymysql.connect(host='192.168.111.200', user='pyUser', password='1234' ₩
 8                                , db='pyDB',   charset='utf8')
 9  # (2) 커서 생성
10  cur = con.cursor()
11
12  # 테이블의 파일명 목록을 가져와서 선택하도록 하기.
13  sql = "SELECT DISTINCT fname, resol FROM rawTBL"
14  cur.execute(sql)
15
16  fnameAndResList = []
17  while True :
18      row = cur.fetchone()
19      if row == None :
20          break
21      fnameAndResList.append(row)
22
23  ######################
24  # <팝업 띄우기>
```

```
selectIndexTuple = None
def btnClick() :
    global selectIndexTuple
    selectIndexTuple = listbox.curselection()
    popWindow.quit()
    popWindow.destroy()

popWindow = Tk()
listbox = Listbox(popWindow)
listbox.pack()
for item in fnameAndResList:
    listbox.insert(END, item)
btn = Button(popWindow, text="선택",
             command= btnClick )
btn.pack()
popWindow.mainloop()

fname, resol = fnameAndResList[selectIndexTuple[0]] # 선택된 튜플의 첫번째 인덱스
# </팝업띄우기>
#######################

sql = "SELECT x, y, value FROM rawTBL WHERE fname = '" ₩
      + fname + "' and resol=" + str(resol)
cur.execute(sql)

# 히스토그램은 무조건 256x256 윈도 크기임.
resol = 256
window = Tk()
window.geometry(str(resol) + "x" + str(resol))
canvas = Canvas(window, height=resol, width=resol)
paper = PhotoImage( height=resol, width=resol)
canvas.create_image( (resol/2, resol/2), image=paper, state='normal')

width = height = resol

histogram=[0]*256
while True :
    row = cur.fetchone()
```

```
63        if row == None :
64            break
65        x, y, value = row
66        histogram[value] += 1
67        #print(x, y, value)
68
69  # 정규화
70  maxValue = max(histogram)
71  minValue = min(histogram)
72
73  diff = maxValue - minValue
74
75  normal_hist = []
76  for i in range(256) :
77        normal_hist.append( int( (histogram[i] - minValue)   * 255.0 / diff) )
78
79  for i in range(256) :
80        for k in range(normal_hist[i]) :
81            paper.put("#%02x%02x%02x" % (0, 0, 0), (i,255-k))
82
83  canvas.pack()
84  con.commit()
85  con.close()
86  window.mainloop()
```

▶ 실습 : MySQL (RAW Viewer)

```
1   import pymysql
2   from tkinter import *
3   import os
4   import math
5
6   # (1) 데이터베이스 연결
7   con = pymysql.connect(host='192.168.111.200', user='pyUser', password='1234' ₩
8                                 , db='pyDB',   charset='utf8')
9   # (2) 커서 생성
10  cur = con.cursor()
11
```

```
# 테이블의 파일명 목록을 가져와서 선택하도록 하기.
sql = "SELECT DISTINCT fname, resol FROM rawTBL"
cur.execute(sql)

fnameAndResList = []
while True :
    row = cur.fetchone()
    if row == None :
        break
    fnameAndResList.append(row)

######################
# <팝업 띄우기>
selectIndexTuple = None
def btnClick() :
    global selectIndexTuple
    selectIndexTuple = listbox.curselection()
    popWindow.quit()
    popWindow.destroy()

popWindow = Tk()
listbox = Listbox(popWindow)
listbox.pack()
for item in fnameAndResList:
    listbox.insert(END, item)
btn = Button(popWindow, text="선택",
             command= btnClick )
btn.pack()
popWindow.mainloop()

fname, resol = fnameAndResList[selectIndexTuple[0]] # 선택된 튜플의 첫번째 인덱스
# </팝업띄우기>
######################

sql = "SELECT x, y, value FROM rawTBL WHERE fname = '" + fname + "' and resol="
+ str(resol)
cur.execute(sql)

```

```
window = Tk()
canvas = Canvas(window, height=resol, width=resol)
paper = PhotoImage( height=resol, width=resol)
canvas.create_image( (resol/2, resol/2), image=paper, state='normal')

width = height = resol

while True :
    row = cur.fetchone()
    if row == None :
        break
    x, y, value = row
    paper.put("#%02x%02x%02x" % (value, value, value), (y, x))

con.commit()
con.close()

canvas.pack()
window.mainloop()
```

# Numpy 기본

Numpy 는 C 언어로 구현된 파이썬 라이브러리로써, 고성능의 수치계산을 위해 제작되었다. Numerical Python 의 줄임말이기도 한 Numpy 는 벡터 및 행렬 연산에 있어서 매우 편리한 기능을 제공한다.

또한 이는 데이터분석을 할 때 사용되는 라이브러리인 pandas 와 matplotlib 의 기반으로 사용되기도 한다.

numpy 에서는 기본적으로 array 라는 단위로 데이터를 관리하며 이에 대해 연산을 수행한다. array 는 말그대로 행렬이라는 개념으로 생각하시면 된다.

먼저 numpy 를 사용하기 위해서는 아래와 같은 코드로 numpy 를 import 해야 한다.

***import numpy as np***

사실상, numpy 를 설치하고 단순히 import numpy 만 해도 되지만, 이를 코드에서 보다 편하게 사용하기 위해 as np 를 붙임으로써 np 라는 이름으로 numpy 를 사용한다.

## 1. Numpy 란?¶

In [1]:

```
# numpy 사용하기
import numpy as np
```

## 2. Array 정의 및 사용하기¶

In [2]:

```
data1 = [1,2,3,4,5]
data1
```

Out[2]:

```
[1, 2, 3, 4, 5]
```

In [3]:

```
data2 = [1,2,3,3.5,4]
data2
```

Out[3]:

```
[1, 2, 3, 3.5, 4]
```

In [4]:

```
# numpy 를 이용하여 array 정의하기
# 1. 위에서 만든 python list 를 이용
arr1 = np.array(data1)
arr1
```

Out[4]:

```
array([1, 2, 3, 4, 5])
```

In [5]:

```
# array 의 형태(크기)를 확인할 수 있다.
arr1.shape
```

Out[5]:

```
(5,)
```

In [6]:

```
# 2. 바로 python list 를 넣어 줌으로써 만들기
arr2 = np.array([1,2,3,4,5])
arr2
```

Out[6]:

```
array([1, 2, 3, 4, 5])
```

In [7]:

```
arr2.shape
```

Out[7]:

```
(5,)
```

In [8]:

```
# array 의 자료형을 확인할 수 있다.
arr2.dtype
```

Out[8]:

```
dtype('int64')
```

In [9]:

```
arr3 = np.array(data2)
arr3
```

Out[9]:

```
array([1. , 2. , 3. , 3.5, 4. ])
```

In [11]:

```
arr3.shape
```

Out[11]:

```
(5,)
```

In [12]:

```
arr3.dtype
```

Out[12]:

```
dtype('float64')
```

In [21]:

```
arr4 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
arr4
```

Out[21]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [22]:

```
arr4.shape
```

Out[22]:

```
(4, 3)
```

**numpy shape**

numpy 에서는 해당 array 의 크기를 알 수 있다.

shape 을 확인함으로써 몇개의 데이터가 있는지, 몇 차원으로 존재하는지 등을 확인할 수 있다.

위에서 arr1.shape 의 결과는 (5,) 으로써, 1 차원의 데이터이며 총 5 라는 크기를 갖고 있음을 알 수 있다.

arr4.shape 의 결과는 (4,3) 으로써, 2 차원의 데이터이며 4 * 3 크기를 갖고 있는 array 이다.

**numpy 자료형**

arr1 이나 arr2 는 int64 라는 자료형을 갖는 것에 반해 arr3 는 float64 라는 자료형을 갖는다.

이는 arr3 내부 데이터를 살펴보면 3.5 라는 실수형 데이터를 갖기 때문임을 알 수 있다.

numpy 에서 사용되는 자료형은 아래와 같다. (자료형 뒤에 붙는 숫자는 몇 비트 크기인지를 의미한다.)

- 부호가 있는 정수 int(8, 16, 32, 64)
- 부호가 없는 정수 uint(8 ,16, 32, 54)
- 실수 float(16, 32, 64, 128)
- 복소수 complex(64, 128, 256)
- 불리언 bool
- 문자열 string_
- 파이썬 오프젝트 object
- 유니코드 unicode_

## 2-1. np.zeros(), np.ones(), np.arange() 함수¶

numpy 에서 array 를 정의할 때 사용되는 함수들이다.

아래 실습을 통해 각각이 어떻게 다른지 살펴본다.

In [23]:

```
np.zeros(10)
```

Out[23]:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [25]:

```
np.zeros((3,5))
```

Out[25]:

```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [24]:

```
# np.zeros() 함수는 인자로 받는 크기만큼, 모든요소가 0 인 array 를 만든다.
```

In [26]:

```
np.ones(9)
```

Out[26]:

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [27]:

```
np.ones((2,10))
```

Out[27]:

```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])
```

In [28]:

```
# np.ones() 함수는 인자로 받는 크기만큼, 모든요소가 1 인 array 를 만든다.
```

In [29]:

```
np.arange(10)
```

Out[29]:

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

In [32]:

```
np.arange(3,10)
```

Out[32]:

```
array([3, 4, 5, 6, 7, 8, 9])
```

In [33]:

```
# np.arange() 함수는 인자로 받는 값 만큼 1 씩 증가하는 1 차원 array 를 만든다.
# 이때 하나의 인자만 입력하면 0 ~ 입력한 인자, 값 만큼의 크기를 가진다.
```

## 3. Array 연산¶

기본적으로 numpy 에서 연산을 할때는 크기가 서로 동일한 array 끼리 연산이 진행된다.

이때 같은 위치에 있는 요소들 끼리 연산이 진행된다.

In [40]:

```
arr1 = np.array([[1,2,3],[4,5,6]])
arr1
```

Out[40]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [41]:

```
arr1.shape
```

Out[41]:

```
(2, 3)
```

In [42]:

```
arr2 = np.array([[10,11,12],[13,14,15]])
arr2
```

Out[42]:

```
array([[10, 11, 12],
       [13, 14, 15]])
```

In [43]:

```
arr2.shape
```

Out[43]:

```
(2, 3)
```

## 3-1. array 덧셈¶

In [44]:

```
arr1 + arr2
```

Out[44]:

```
array([[11, 13, 15],
       [17, 19, 21]])
```

## 3-2. array 뺄셈¶

In [45]:

```
arr1 - arr2
```

Out[45]:

```
array([[-9, -9, -9],
       [-9, -9, -9]])
```

## 3-3. array 곱셈¶

주의하자.

행렬의 곱처럼 곱셈이 진행되는 것이 아니라 각 요소별로 곱셈이 진행된다.

In [47]:

```
arr1 * arr2
```

Out[47]:

```
array([[10, 22, 36],
```

```
       [52, 70, 90]])
```

## 3-4. array 나눗셈¶

In [48]:

```
arr1 / arr2
```

Out[48]:

```
array([[0.1        , 0.18181818, 0.25       ],
       [0.30769231, 0.35714286, 0.4        ]])
```

## 3-5. array 의 브로드 캐스트¶

위에서는 array 가 같은 크기를 가져야 서로 연산이 가능하다고 했지만,

numpy 에서는 브로드캐스트라는 기능을 제공한다.

브로드캐스트란, 서로 크기가 다른 array 가 연산이 가능하게끔 하는 것이다.

In [49]:

```
arr1
```

Out[49]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [50]:

```
arr1.shape
```

Out[50]:

```
(2, 3)
```

In [52]:

```
arr3 = np.array([10,11,12])
arr3
```

Out[52]:

```
array([10, 11, 12])
```

In [53]:

```
arr3.shape
```

Out[53]:

```
(3,)
```

In [54]:

```
arr1 + arr3
```

Out[54]:

```
array([[11, 13, 15],
       [14, 16, 18]])
```

In [55]:

```
arr1 * arr3
```

Out[55]:

```
array([[10, 22, 36],
       [40, 55, 72]])
```

위와 같이 서로 크기가 다른 arr1 과 arr3 의 연산이 가능하다.

연산결과를 살펴보면 arr3 이 [10,11,12] 에서 [[10,11,12],[10,11,12]]로 확장되어 계산되었음을 확인할 수 있다.

동일한 방식으로 하나의 array 에 스칼라 연산도 가능하다.

In [56]:

```
arr1 * 10
```

Out[56]:

```
array([[10, 20, 30],
       [40, 50, 60]])
```

In [57]:

```
# 요소에 대해 제곱처리
arr1 ** 2
```

Out[57]:

```
array([[ 1,  4,  9],
       [16, 25, 36]])
```

# 4. Array 인덱싱¶

numpy 에서 사용되는 인덱싱은 기본적으로 python 인덱싱과 동일하다.

이때, python 에서와 같이 1 번째로 시작하는 것이 아니라 0 번째로 시작하는 것에 주의한다.

In [64]:

```
arr1 = np.arange(10)
arr1
```

Out[64]:

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

In [65]:

```
# 0 번째 요소
arr1[0]
```

Out[65]:

```
0
```

In [66]:

```
# 3 번째 요소
arr1[3]
```

Out[66]:

```
3
```

In [72]:

```
# 3 번째 요소부터 8 번째 요소
arr1[3:9]
```

Out[72]:

```
array([3, 4, 5, 6, 7, 8])
```

In [73]:

```
arr1[:]
```

Out[73]:

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

물론 1 차원이 아닌 그 이상의 차원에서도 인덱싱이 가능하다.

In [69]:

```
arr2 = np.array([[1,2,3,4],
                 [5,6,7,8],
                 [9,10,11,12]])
arr2
```

Out[69]:

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [71]:

```
# 2 차원의 array 에서 인덱싱을 하기 위해선 2 개의 인자를 입력해야 한다.
arr2[0,0]
```

Out[71]:

```
1
```

In [77]:

```
# 2 행의 모든 요소 꺼내기
arr2[2,:]
```

Out[77]:

```
array([ 9, 10, 11, 12])
```

In [78]:

```
# 2 행의 3 번째 요소 꺼내기
```

```
arr2[2,3]
```

Out[78]:

```
12
```

In [80]:

```
# 모든 열의 3 번째 요소 꺼내기

arr2[:,3]
```

Out[80]:

```
array([ 4,  8, 12])
```

# 5. Array boolean 인덱싱(마스크)¶

위에서 이용한 다차원의 인덱싱을 응용하여 boolean 인덱싱을 할 수 있다.

해당 기능은 주로 마스크라고 이야기하는데, boolean 인덱싱을 통해 만들어낸 array 를 통해

우리가 원하는 행 또는 열의 값만 뽑아낼 수 있다.

즉, 마스크처럼 우리가 가리고 싶은 부분은 가리고, 원하는 요소만 꺼낼 수 있다.

In [81]:

```
names = np.array(['Beomwoo','Beomwoo','Kim','Joan','Lee','Beomwoo','Park','Beomwoo'])

names
```

Out[81]:

```
array(['Beomwoo', 'Beomwoo', 'Kim', 'Joan', 'Lee', 'Beomwoo', 'Park',

       'Beomwoo'], dtype='<U7')
```

In [82]:

```
names.shape
```

Out[82]:

```
(8,)
```

In [87]:

```
# 아래에서 사용되는 np.random.randn() 함수는 기대값이 0 이고, 표준편차가 1 인 가우시안 정규 분포를 따르는 난수를 발생시키는 함수이다.
```

```
# 이 외에도 0~1 의 난수를 발생시키는 np.random.rand() 함수도 존재한다.
data = np.random.randn(8,4)
data
```

Out[87]:

```
array([[-1.07099572, -0.85382063, -1.42474621, -0.05992846],
       [ 1.93097843, -1.8885167 ,  1.99767454,  0.31524498],
       [-0.22633642, -0.76385264,  0.16368804,  0.91204438],
       [ 1.34321923,  1.54532121,  0.28814921,  0.50688776],
       [ 0.4126606 , -0.52356522,  0.27124037, -0.66383264],
       [ 0.88575452, -1.39205929,  0.91019739, -1.04676349],
       [-0.05673648, -1.63408607, -2.29844338, -0.3662913 ],
       [ 0.45963024,  0.35662128,  0.18307525,  1.46992167]])
```

In [88]:

```
data.shape
```

Out[88]:

```
(8, 4)
```

위와 같은 names 와 data 라는 array 가 있다.

이때, names 의 각 요소가 data 의 각 행과 연결된다고 가정해보자.

그리고 이 때, names 가 Beomwoo 인 행의 data 만 보고 싶을 때 다음과 같이 마스크를 사용한다.

In [91]:

```
# 요소가 Beomwoo 인 항목에 대한 mask 생성
names_Beomwoo_mask = (names == 'Beomwoo')
names_Beomwoo_mask
```

Out[91]:

```
array([ True,  True, False, False, False,  True, False,  True])
```

In [92]:

```
data[names_Beomwoo_mask,:]
```

Out[92]:

```
array([[-1.07099572, -0.85382063, -1.42474621, -0.05992846],
       [ 1.93097843, -1.8885167 ,  1.99767454,  0.31524498],
       [ 0.88575452, -1.39205929,  0.91019739, -1.04676349],
       [ 0.45963024,  0.35662128,  0.18307525,  1.46992167]])
```

위의 결과를 보면, 요소가 Beomwoo 인 것은 0 번째, 1 번째, 5 번째, 7 번째 이므로

data 에서 0,1,5,7 행의 모든 요소를 꺼내와야 한다.

이를 위해 요소가 Beomwoo 인 것에 대한 boolean 값을 가지는 mask 를 만들었고

마스크를 인덱싱에 응용하여 data 의 0,1,5,7 행을 꺼냈다.

In [93]:

```
# 요소가 Kim 인 행의 데이터만 꺼내기
data[names == 'Kim',:]
```

Out[93]:

```
array([[-0.22633642, -0.76385264,  0.16368804,  0.91204438]])
```

In [94]:

```
# 논리 연산을 응용하여, 요소가 Kim 또는 Park 인 행의 데이터만 꺼내기
data[(names == 'Kim') | (names == 'Park'),:]
```

Out[94]:

```
array([[-0.22633642, -0.76385264,  0.16368804,  0.91204438],
       [-0.05673648, -1.63408607, -2.29844338, -0.3662913 ]])
```

물론 data array 자체적으로도 마스크를 만들고, 이를 응용하여 인덱싱이 가능하다.

data array 에서 0 번째 열의 값이 0 보다 작은 행을 구해보자.

In [97]:

```
# 먼저 마스크를 만든다.
# data array 에서 0 번째 열이 0 보다 작은 요소의 boolean 값은 다음과 같다.
data[:,0] < 0
```

Out[97]:

```
array([ True, False,  True, False, False, False,  True, False])
```

In [98]:

```
# 위에서 만든 마스크를 이용하여 0 번째 열의 값이 0 보다 작은 행을 구한다.
data[data[:,0]<0,:]
```

Out[98]:

```
array([[-1.07099572, -0.85382063, -1.42474621, -0.05992846],
       [-0.22633642, -0.76385264,  0.16368804,  0.91204438],
       [-0.05673648, -1.63408607, -2.29844338, -0.3662913 ]])
```

이를 통해 특정 위치에만 우리가 원하는 값을 대입할 수 있다.

위에서 얻은, 0 번째 열의 값이 0 보다 작은 행의 2,3 번째 열값에 0 을 대입해보자.

In [100]:

```
# 0 번째 열의 값이 0 보다 작은 행의 2,3 번째 열 값
data[data[:,0]<0,2:4]
```

Out[100]:

```
array([[-1.42474621, -0.05992846],
       [ 0.16368804,  0.91204438],
       [-2.29844338, -0.3662913 ]])
```

In [102]:

```
data[data[:,0]<0,2:4] = 0
data
```

Out[102]:

```
array([[-1.07099572, -0.85382063,  0.        ,  0.        ],
       [ 1.93097843, -1.8885167 ,  1.99767454,  0.31524498],
       [-0.22633642, -0.76385264,  0.        ,  0.        ],
       [ 1.34321923,  1.54532121,  0.28814921,  0.50688776],
       [ 0.4126606 , -0.52356522,  0.27124037, -0.66383264],
       [ 0.88575452, -1.39205929,  0.91019739, -1.04676349],
```

```
       [-0.05673648, -1.63408607,  0.        ,  0.        ],
       [ 0.45963024,  0.35662128,  0.18307525,  1.46992167]])
```

# 6. Numpy 함수¶

numpy 에서는 array 에 적용되는 다양한 함수가 있다.

## 6-1. 하나의 array 에 적용되는 함수¶

In [112]:

```
arr1 = np.random.randn(5,3)
arr1
```

Out[112]:

```
array([[-1.28394941, -1.38235479,  0.3676742 ],
       [ 0.91707237,  0.45364032,  0.00683315],
       [ 0.51191795,  0.39014894, -0.15396686],
       [ 0.75541648, -3.0457677 ,  0.83785171],
       [ 0.36609986,  1.2300834 ,  0.51764117]])
```

In [113]:

```
# 각 성분의 절대값 계산하기
np.abs(arr1)
```

Out[113]:

```
array([[1.28394941, 1.38235479, 0.3676742 ],
       [0.91707237, 0.45364032, 0.00683315],
       [0.51191795, 0.39014894, 0.15396686],
       [0.75541648, 3.0457677 , 0.83785171],
       [0.36609986, 1.2300834 , 0.51764117]])
```

In [114]:

```
# 각 성분의 제곱근 계산하기 ( == array ** 0.5)
np.sqrt(arr1)
```

```
/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/__main__.py:2: RuntimeWarning: invalid value encountered in sqrt
  from ipykernel import kernelapp as app
```

Out[114]:

```
array([[        nan,         nan, 0.60636144],
       [0.95763896, 0.67352826, 0.08266285],
       [0.71548442, 0.62461903,         nan],
       [0.86914699,         nan, 0.9153424 ],
       [0.60506187, 1.10909125, 0.71947284]])
```

In [115]:

```
# 각 성분의 제곱 계산하기
np.square(arr1)
```

Out[115]:

```
array([[1.64852609e+00, 1.91090476e+00, 1.35184315e-01],
       [8.41021739e-01, 2.05789543e-01, 4.66919060e-05],
       [2.62059989e-01, 1.52216194e-01, 2.37057951e-02],
       [5.70654061e-01, 9.27670091e+00, 7.01995494e-01],
       [1.34029109e-01, 1.51310516e+00, 2.67952378e-01]])
```

In [116]:

```
# 각 성분을 무리수 e 의 지수로 삼은 값을 계산하기
np.exp(arr1)
```

Out[116]:

```
array([[0.27694138, 0.25098684, 1.44437138],
       [2.50195487, 1.57403175, 1.00685655],
       [1.66848821, 1.47720079, 0.85730043],
       [2.12849782, 0.04755979, 2.3113961 ],
       [1.44209925, 3.42151487, 1.6780647 ]])
```

In [117]:

```
# 각 성분을 자연로그, 상용로그, 밑이 2 인 로그를 씌운 값을 계산하기

np.log(arr1)
```

```
/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/__main__.py:2: RuntimeWarning: invalid value encountered in log

  from ipykernel import kernelapp as app
```

Out[117]:

```
array([[        nan,         nan, -1.00055807],
       [-0.08656889, -0.79045064, -4.98596986],
       [-0.66959092, -0.94122672,         nan],
       [-0.28048605,         nan, -0.17691415],
       [-1.00484914,  0.20708197, -0.65847301]])
```

In [118]:

```
np.log10(arr1)
```

```
/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/__main__.py:1: RuntimeWarning: invalid value encountered in log10

  if __name__ == '__main__':
```

Out[118]:

```
array([[        nan,         nan, -0.43453685],
       [-0.03759639, -0.34328835, -2.1653792 ],
       [-0.29079964, -0.40876957,         nan],
       [-0.12181354,         nan, -0.07683284],
       [-0.43640043,  0.08993456, -0.28597119]])
```

In [119]:

```
np.log2(arr1)
```

```
/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/__main__.py:1: RuntimeWarning: invalid value encountered in log2

  if __name__ == '__main__':
```

Out[119]:

```
array([[        nan,         nan, -1.44350016],
       [-0.1248925 , -1.14037921, -7.193234  ],
       [-0.9660155 , -1.35790312,         nan],
       [-0.40465583,         nan, -0.25523316],
       [-1.44969086,  0.29875613, -0.94997574]])
```

In [120]:

```
# 각 성분의 부호 계산하기(+인 경우 1, -인 경우 -1, 0 인 경우 0)
np.sign(arr1)
```

Out[120]:

```
array([[-1., -1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1., -1.],
       [ 1., -1.,  1.],
       [ 1.,  1.,  1.]])
```

In [121]:

```
# 각 성분의 소수 첫 번째 자리에서 올림한 값을 계산하기
np.ceil(arr1)
```

Out[121]:

```
array([[-1., -1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1., -0.],
       [ 1., -3.,  1.],
       [ 1.,  2.,  1.]])
```

In [122]:

```
# 각 성분의 소수 첫 번째 자리에서 내림한 값을 계산하기
np.floor(arr1)
```

Out[122]:

```
array([[-2., -2.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0., -1.],
       [ 0., -4.,  0.],
       [ 0.,  1.,  0.]])
```

In [124]:

```
# 각 성분이 NaN 인 경우 True 를, 아닌 경우 False 를 반환하기
np.isnan(arr1)
```

Out[124]:

```
array([[False, False, False],
       [False, False, False],
       [False, False, False],
       [False, False, False],
       [False, False, False]])
```

In [125]:

```
np.isnan(np.log(arr1))
```

```
/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/__main__.py:1: RuntimeWarning: invalid value encountered in log
  if __name__ == '__main__':
```

Out[125]:

```
array([[ True,  True, False],
       [False, False, False],
       [False, False,  True],
       [False,  True, False],
       [False, False, False]])
```

In [126]:

```
# 각 성분이 무한대인 경우 True 를, 아닌 경우 False 를 반환하기
```

```
np.isinf(arr1)
```

Out[126]:

```
array([[False, False, False],
       [False, False, False],
       [False, False, False],
       [False, False, False],
       [False, False, False]])
```

In [127]:

```
# 각 성분에 대해 삼각함수 값을 계산하기(cos, cosh, sin, sinh, tan, tanh)
np.cos(arr1)
```

Out[127]:

```
array([[ 0.28292939,  0.18732825,  0.93316587],
       [ 0.60814678,  0.89885772,  0.99997665],
       [ 0.8718066 ,  0.92485242,  0.9881705 ],
       [ 0.72798607, -0.9954123 ,  0.66906099],
       [ 0.9337306 ,  0.33415913,  0.86898882]])
```

In [128]:

```
np.tanh(arr1)
```

Out[128]:

```
array([[-0.85753362, -0.88147747,  0.35195568],
       [ 0.72450949,  0.42488675,  0.00683304],
       [ 0.47143823,  0.37148862, -0.15276165],
       [ 0.63836924, -0.99548634,  0.68466951],
       [ 0.3505756 ,  0.84260351,  0.47587741]])
```

## 6-2. 두 개의 array 에 적용되는 함수¶

In [129]:

```
arr1
```

Out[129]:

```
array([[-1.28394941, -1.38235479,  0.3676742 ],
       [ 0.91707237,  0.45364032,  0.00683315],
       [ 0.51191795,  0.39014894, -0.15396686],
       [ 0.75541648, -3.0457677 ,  0.83785171],
       [ 0.36609986,  1.2300834 ,  0.51764117]])
```

In [131]:

```
arr2 = np.random.randn(5,3)
arr2
```

Out[131]:

```
array([[-1.08072661,  0.49305711, -1.2341793 ],
       [ 0.72539264, -0.17482108, -2.29144412],
       [ 0.33676285, -0.44206124,  0.68359426],
       [-1.3367298 , -0.62530265,  0.04842608],
       [ 0.46567612, -2.53008109,  0.80771562]])
```

In [132]:

```
# 두 개의 array 에 대해 동일한 위치의 성분끼리 연산 값을 계산하기(add, subtract, multiply, divide)
np.multiply(arr1,arr2)
```

Out[132]:

```
array([[ 1.3875983 , -0.68157986, -0.45377588],
       [ 0.66523755, -0.07930589, -0.01565778],
       [ 0.17239495, -0.17246972, -0.10525086],
       [-1.00978772,  1.90452663,  0.04057388],
       [ 0.17048396, -3.11221075,  0.41810686]])
```

In [133]:

```
# 두 개의 array 에 대해 동일한 위치의 성분끼리 비교하여 최대값 또는 최소값 계산하기(maximum, minimum)
np.maximum(arr1,arr2)
```

Out[133]:

```
array([[-1.08072661,  0.49305711,  0.3676742 ],
       [ 0.91707237,  0.45364032,  0.00683315],
       [ 0.51191795,  0.39014894,  0.68359426],
       [ 0.75541648, -0.62530265,  0.83785171],
       [ 0.46567612,  1.2300834 ,  0.80771562]])
```

## 6-3. 통계 함수¶

통계 함수를 통해 array 의 합이나 평균등을 구할 때,

추가로 axis 라는 인자에 대한 값을 지정하여 열 또는 행의 합 또는 평균등을 구할 수 있다.

In [134]:

```
arr1
```

Out[134]:

```
array([[-1.28394941, -1.38235479,  0.3676742 ],
       [ 0.91707237,  0.45364032,  0.00683315],
       [ 0.51191795,  0.39014894, -0.15396686],
       [ 0.75541648, -3.0457677 ,  0.83785171],
       [ 0.36609986,  1.2300834 ,  0.51764117]])
```

In [136]:

```
# 전체 성분의 합을 계산
np.sum(arr1)
```

Out[136]:

```
0.4883407866652476
```

In [138]:

```
# 열 간의 합을 계산
np.sum(arr1, axis=1)
```

Out[138]:

```
array([-2.29863    ,  1.37754584,  0.74810003, -1.45249951,  2.11382443])
```

In [140]:

```
# 행 간의 합을 계산

np.sum(arr1, axis=0)
```

Out[140]:

```
array([ 1.26655726, -2.35424983,  1.57603336])
```

In [141]:

```
# 전체 성분의 평균을 계산

np.mean(arr1)
```

Out[141]:

```
0.032556052444349844
```

In [142]:

```
# 행 간의 평균을 계산

np.mean(arr1, axis=0)
```

Out[142]:

```
array([ 0.25331145, -0.47084997,  0.31520667])
```

In [143]:

```
# 전체 성분의 표준편차, 분산, 최소값, 최대값 계산(std, var, min, max)

np.std(arr1)
```

Out[143]:

```
1.0840662273348296
```

In [145]:

```
np.min(arr1, axis=1)
```

Out[145]:

```
array([-1.38235479,  0.00683315, -0.15396686, -3.0457677 ,  0.36609986])
```

In [147]:

```
# 전체 성분의 최소값, 최대값이 위치한 인덱스를 반환(argmin, argmax)

np.argmin(arr1)
```

Out[147]:

```
10
```

In [148]:

```
np.argmax(arr1,axis=0)
```

Out[148]:

```
array([1, 4, 3])
```

In [149]:

```
# 맨 처음 성분부터 각 성분까지의 누적합 또는 누적곱을 계산(cumsum, cumprod)

np.cumsum(arr1)
```

Out[149]:

```
array([-1.28394941, -2.6663042 , -2.29863   , -1.38155763, -0.92791731,
       -0.92108416, -0.40916621, -0.01901727, -0.17298413,  0.58243235,
       -2.46333535, -1.62548364, -1.25938378, -0.02930038,  0.48834079])
```

In [152]:

```
np.cumsum(arr1,axis=1)
```

Out[152]:

```
array([[-1.28394941, -2.6663042 , -2.29863   ],
       [ 0.91707237,  1.3707127 ,  1.37754584],
       [ 0.51191795,  0.90206689,  0.74810003],
       [ 0.75541648, -2.29035122, -1.45249951],
       [ 0.36609986,  1.59618326,  2.11382443]])
```

In [153]:

```
np.cumprod(arr1)
```

Out[153]:

```
array([-1.28394941e+00,  1.77487362e+00,  6.52575230e-01,  5.98458715e-01,
        2.71485005e-01,  1.85509711e-03,  9.49657512e-04,  3.70507870e-04,
       -5.70459347e-05, -4.30934393e-05,  1.31252606e-04,  1.09970221e-04,
        4.02600826e-05,  4.95232591e-05,  2.56352776e-05])
```

## 6-4. 기타 함수¶

In [154]:

```
arr1
```

Out[154]:

```
array([[-1.28394941, -1.38235479,  0.3676742 ],
       [ 0.91707237,  0.45364032,  0.00683315],
       [ 0.51191795,  0.39014894, -0.15396686],
       [ 0.75541648, -3.0457677 ,  0.83785171],
       [ 0.36609986,  1.2300834 ,  0.51764117]])
```

In [155]:

```
# 전체 성분에 대해서 오름차순으로 정렬
np.sort(arr1)
```

Out[155]:

```
array([[-1.38235479, -1.28394941,  0.3676742 ],
       [ 0.00683315,  0.45364032,  0.91707237],
       [-0.15396686,  0.39014894,  0.51191795],
       [-3.0457677 ,  0.75541648,  0.83785171],
       [ 0.36609986,  0.51764117,  1.2300834 ]])
```

In [156]:

```
# 전체 성분에 대해서 내림차순으로 정렬
np.sort(arr1)[::-1]
```

Out[156]:

```
array([[ 0.36609986,  0.51764117,  1.2300834 ],
       [-3.0457677 ,  0.75541648,  0.83785171],
       [-0.15396686,  0.39014894,  0.51191795],
       [ 0.00683315,  0.45364032,  0.91707237],
       [-1.38235479, -1.28394941,  0.3676742 ]])
```

In [157]:

```
# 행 방향으로 오름차순으로 정렬
np.sort(arr1,axis=0)
```

Out[157]:

```
array([[-1.28394941, -3.0457677 , -0.15396686],
       [ 0.36609986, -1.38235479,  0.00683315],
       [ 0.51191795,  0.39014894,  0.3676742 ],
       [ 0.75541648,  0.45364032,  0.51764117],
       [ 0.91707237,  1.2300834 ,  0.83785171]])
```

# Pandas 기본¶

## ▶ Pandas 개요

Pandas 는 파이썬에서 사용하는 데이터분석 라이브러리로, 행과 열로 이루어진 데이터 객체를 만들어 다룰 수 있게 되며 보다 안정적으로 대용량의 데이터들을 처리하는데 매우 편리한 도구다.

먼저 pandas 를 사용하기 위해서는 pandas 를 설치한 이후에 import 를 해야 한다.

In [2]:

```
# pandas 사용하기
import numpy as np # numpy 도 함께 import
import pandas as pd
```

## ▶ Pandas 자료구조¶

Pandas 에서는 기본적으로 정의되는 자료구조인 Series 와 Data Frame 을 사용한다.

이 자료구조들은 빅 데이터 분석에 있어서 높은 수준의 성능을 보여줍니다.

### √ Series¶

먼저 Series 에 대해서 알아보도록 하겠습니다.

In [3]:

```
# Series 정의하기
obj = pd.Series([9, 3, -6, 5])
obj
```

Out[3]:

```
0    9
1    3
2   -6
3    5
```

```
dtype: int64
```

In [4]:

```
# Series 의 값만 확인하기
obj.values
```

Out[4]:

```
array([9, 3, -6, 5])
```

In [5]:

```
# Series 의 인덱스만 확인하기
obj.index
```

Out[5]:

```
RangeIndex(start=0, stop=4, step=1)
```

In [7]:

```
# Series 의 자료형 확인하기
obj.dtypes
```

Out[7]:

```
dtype('int64')
```

In [8]:

```
# 인덱스를 바꿀 수 있다.
obj2 = pd.Series([9, 3, -6, 5], index=['d', 'b', 'a', 'c'])
obj2
```

Out[8]:

```
d    9
b    3
a   -6
c    5
dtype: int64
```

In [10]:

```
# python 의 dictionary 자료형을 Series data 로 만들 수 있다.
# dictionary 의 key 가 Series 의 index 가 된다
sdata = {'Cass': 35000, 'Miller': 67000, 'Terra': 12000, 'Hite': 4000}
obj3 = pd.Series(sdata)
obj3
```

Out[10]:

```
Cass      35000
Miller    67000
Terra     12000
Hite       4000
dtype: int64
```

In [12]:

```
obj3.name = 'Salary'
obj3.index.name = "Names"
obj3
```

Out[12]:

```
Names
Cass      35000
Miller    67000
Terra     12000
Hite       4000
Name: Salary, dtype: int64
```

In [13]:

```
# index 변경
obj3.index = ['A', 'B', 'C', 'D']
obj3
```

Out[13]:

```
A      35000
B      67000
C      12000
D       4000
Name: Salary, dtype: int64
```

## √ Data Frame¶

이번에는 Data Frame 에 대해서 알아보도록 하겠습니다.

In [23]:

```
# Data Frame 정의하기
# 이전에 DataFrame 에 들어갈 데이터를 정의해주어야 하는데,
# 이는 python 의 dictionary 또는 numpy 의 array 로 정의할 수 있다.
data = {'name': ['Miller', 'Miller', 'Miller', 'Cass', 'Park'],
        'year': [2013, 2014, 2015, 2016, 2015],
        'points': [1.5, 1.7, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data)
df
```

Out[23]:

| | name | year | points |
|---|---|---|---|
| **0** | Miller | 2013 | 1.5 |
| **1** | Miller | 2014 | 1.7 |
| **2** | Miller | 2015 | 3.6 |
| **3** | Cass | 2016 | 2.4 |
| **4** | Park | 2015 | 2.9 |

In [24]:

```
# 행과 열의 구조를 가진 데이터가 생긴다.
```

In [25]:

```
# 행 방향의 index
```

```
df.index
```

Out[25]:

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

In [26]:

```
# 열 방향의 index

df.columns
```

Out[26]:

```
Index(['name', 'year', 'points'], dtype='object')
```

In [27]:

```
# 값 얻기

df.values
```

Out[27]:

```
array([['Miller', 2013, 1.5],

       ['Miller', 2014, 1.7],

       ['Miller', 2015, 3.6],

       ['Cass', 2016, 2.4],

       ['Park', 2015, 2.9]], dtype=object)
```

In [28]:

```
# 각 인덱스에 대한 이름 설정하기

df.index.name = 'Num'

df.columns.name = 'Info'

df
```

Out[28]:

| Info | name | year | points |
|---|---|---|---|
| **Num** | | | |
| **0** | Miller | 2013 | 1.5 |
| **1** | Miller | 2014 | 1.7 |
| **2** | Miller | 2015 | 3.6 |

| Info | name | year | points |
|---|---|---|---|
| **Num** | | | |
| **3** | Cass | 2016 | 2.4 |
| **4** | Park | 2015 | 2.9 |

In [29]:

```
# DataFrame 을 만들면서 columns 와 index 를 설정할 수 있다.
df2 = pd.DataFrame(data, columns=['year', 'name', 'points', 'penalty'],
                         index=['one', 'two', 'three', 'four', 'five'])
df2
```

Out[29]:

| | year | name | points | penalty |
|---|---|---|---|---|
| **one** | 2013 | Miller | 1.5 | NaN |
| **two** | 2014 | Miller | 1.7 | NaN |
| **three** | 2015 | Miller | 3.6 | NaN |
| **four** | 2016 | Cass | 2.4 | NaN |
| **five** | 2015 | Park | 2.9 | NaN |

DataFrame 을 정의하면서, data 로 들어가는 python dictionary 와 columns 의 순서가 달라도 알아서 맞춰서 정의된다.

하지만 data 에 포함되어 있지 않은 값은 NaN(Not a Number)으로 나타나게 되는데,

이는 null 과 같은 개념이다.

NaN 값은 추후에 어떠한 방법으로도 처리가 되지 않는 데이터이다.

따라서 올바른 데이터 처리를 위해 추가적으로 값을 넣어줘야 한다.

In [31]:

```
# describe() 함수는 DataFrame 의 계산 가능한 값들에 대한 다양한 계산 값을 보여준다.
df2.describe()
```

Out[31]:

| | year | points |
|---|---|---|
| **count** | 5.000000 | 5.000000 |
| **mean** | 2014.600000 | 2.420000 |
| **std** | 1.140175 | 0.864292 |
| **min** | 2013.000000 | 1.500000 |
| **25%** | 2014.000000 | 1.700000 |
| **50%** | 2015.000000 | 2.400000 |

| | year | points |
|---|---|---|
| **75%** | 2015.000000 | 2.900000 |
| **max** | 2016.000000 | 3.600000 |

# ▶DataFrame Indexing¶

In [33]:

```
data = {"names": ["Yaoong", "Yaoong", "Yaoong", "MungMung", "MungMung"],
        "year": [2014, 2015, 2016, 2015, 2016],
        "points": [1.5, 1.7, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data, columns=["year", "names", "points", "penalty"],
                  index=["one", "two", "three", "four", "five"])
df
```

Out[33]:

| | year | names | points | penalty |
|---|---|---|---|---|
| **one** | 2014 | Yaoong | 1.5 | NaN |
| **two** | 2015 | Yaoong | 1.7 | NaN |
| **three** | 2016 | Yaoong | 3.6 | NaN |
| **four** | 2015 | MungMung | 2.4 | NaN |
| **five** | 2016 | MungMung | 2.9 | NaN |

## √ DataFrame 에서 열을 선택하고 조작하기¶

In [34]:

```
df['year']
```

Out[34]:

```
one     2014
two     2015
three   2016
four    2015
```

```
five        2016
Name: year, dtype: int64
```

In [36]:

```
# 동일한 의미를 갖는, 다른 방법
df.year
```

Out[36]:

```
one          2014
two          2015
three       2016
four         2015
five        2016
Name: year, dtype: int64
```

In [37]:

```
df[['year','points']]
```

Out[37]:

| | year | points |
|---|---|---|
| **one** | 2014 | 1.5 |
| **two** | 2015 | 1.7 |
| **three** | 2016 | 3.6 |
| **four** | 2015 | 2.4 |
| **five** | 2016 | 2.9 |

In [38]:

```
# 특정 열에 대해 위와 같이 선택하고, 우리가 원하는 값을 대입할 수 있다.
df['penalty'] = 0.5
```

In [39]:

```
df
```

Out[39]:

| | year | names | points | penalty |
|---|---|---|---|---|
| **one** | 2014 | Yaoong | 1.5 | 0.5 |

| | year | names | points | penalty |
|---|---|---|---|---|
| **two** | 2015 | Yaoong | 1.7 | 0.5 |
| **three** | 2016 | Yaoong | 3.6 | 0.5 |
| **four** | 2015 | MungMung | 2.4 | 0.5 |
| **five** | 2016 | MungMung | 2.9 | 0.5 |

In [42]:

```
# 또는

df['penalty'] = [0.1, 0.2, 0.3, 0.4, 0.5] # python 의 List 나 numpy 의 array
```

In [43]:

```
df
```

Out[43]:

| | year | names | points | penalty |
|---|---|---|---|---|
| **one** | 2014 | Yaoong | 1.5 | 0.1 |
| **two** | 2015 | Yaoong | 1.7 | 0.2 |
| **three** | 2016 | Yaoong | 3.6 | 0.3 |
| **four** | 2015 | MungMung | 2.4 | 0.4 |
| **five** | 2016 | MungMung | 2.9 | 0.5 |

In [44]:

```
# 새로운 열을 추가하기

df['zeros'] = np.arange(5)
```

In [45]:

```
df
```

Out[45]:

| | year | names | points | penalty | zeros |
|---|---|---|---|---|---|
| **one** | 2014 | Yaoong | 1.5 | 0.1 | 0 |
| **two** | 2015 | Yaoong | 1.7 | 0.2 | 1 |
| **three** | 2016 | Yaoong | 3.6 | 0.3 | 2 |
| **four** | 2015 | MungMung | 2.4 | 0.4 | 3 |
| **five** | 2016 | MungMung | 2.9 | 0.5 | 4 |

In [49]:

```
# Series 를 추가할 수도 있다.

val = pd.Series([-1.2, -1.5, -1.7], index=['two','four','five'])
```

In [50]:

```
df['debt'] = val
```

In [51]:

```
df
```

Out[51]:

| | year | names | points | penalty | zeros | debt |
|---|---|---|---|---|---|---|
| **one** | 2014 | Yaoong | 1.5 | 0.1 | 0 | NaN |
| **two** | 2015 | Yaoong | 1.7 | 0.2 | 1 | -1.2 |
| **three** | 2016 | Yaoong | 3.6 | 0.3 | 2 | NaN |
| **four** | 2015 | MungMung | 2.4 | 0.4 | 3 | -1.5 |
| **five** | 2016 | MungMung | 2.9 | 0.5 | 4 | -1.7 |

하지만 Series 로 넣을 때는 val 와 같이 넣으려는 data 의 index 에 맞춰서 데이터가 들어간다.

이점이 python list 나 numpy array 로 데이터를 넣을때와 가장 큰 차이점이다.

In [52]:

```
df['net_points'] = df['points'] - df['penalty']
```

In [53]:

```
df['high_points'] = df['net_points'] > 2.0
```

In [54]:

```
df
```

Out[54]:

| | year | names | points | penalty | zeros | debt | net_points | high_points |
|---|---|---|---|---|---|---|---|---|
| **one** | 2014 | Yaoong | 1.5 | 0.1 | 0 | NaN | 1.4 | False |
| **two** | 2015 | Yaoong | 1.7 | 0.2 | 1 | -1.2 | 1.5 | False |
| **three** | 2016 | Yaoong | 3.6 | 0.3 | 2 | NaN | 3.3 | True |
| **four** | 2015 | MungMung | 2.4 | 0.4 | 3 | -1.5 | 2.0 | False |
| **five** | 2016 | MungMung | 2.9 | 0.5 | 4 | -1.7 | 2.4 | True |

In [55]:

```
# 열 삭제하기

del df['high_points']
```

In [56]:

```
del df['net_points']

del df['zeros']
```

In [57]:

```
df
```

Out[57]:

| | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| **one** | 2014 | Yaoong | 1.5 | 0.1 | NaN |
| **two** | 2015 | Yaoong | 1.7 | 0.2 | -1.2 |
| **three** | 2016 | Yaoong | 3.6 | 0.3 | NaN |
| **four** | 2015 | MungMung | 2.4 | 0.4 | -1.5 |
| **five** | 2016 | MungMung | 2.9 | 0.5 | -1.7 |

In [58]:

```
df.columns
```

Out[58]:

```
Index(['year', 'names', 'points', 'penalty', 'debt'], dtype='object')
```

In [59]:

```
df.index.name = 'Order'

df.columns.name = 'Info'
```

In [60]:

```
df
```

Out[60]:

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| **Order** | | | | | |
| **one** | 2014 | Yaoong | 1.5 | 0.1 | NaN |
| **two** | 2015 | Yaoong | 1.7 | 0.2 | -1.2 |
| **three** | 2016 | Yaoong | 3.6 | 0.3 | NaN |
| **four** | 2015 | MungMung | 2.4 | 0.4 | -1.5 |
| **five** | 2016 | MungMung | 2.9 | 0.5 | -1.7 |

## √ DataFrame 에서 행을 선택하고 조작하기¶

pandas 에서는 DataFrame 에서 행을 인덱싱하는 방법이 무수히 많다.

물론 위에서 소개했던 열을 선택하는 방법도 수많은 방법중에 하나에 불과하다.

In [61]:

```
# 0 번째 부터 2(3-1) 번째까지 가져온다.

# 뒤에 써준 숫자번째의 행은 뺀다.

df[0:3]
```

Out[61]:

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| **Order** | | | | | |
| **one** | 2014 | Yaoong | 1.5 | 0.1 | NaN |
| **two** | 2015 | Yaoong | 1.7 | 0.2 | -1.2 |
| **three** | 2016 | Yaoong | 3.6 | 0.3 | NaN |

In [63]:

```
# tow 라는 행부터 four 라는 행까지 가져온다.

# 뒤에 써준 이름의 행을 빼지 않는다.

df['two':'four'] # 하지만 비추천!
```

Out[63]:

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| **Order** | | | | | |
| **two** | 2015 | Yaoong | 1.7 | 0.2 | -1.2 |
| **three** | 2016 | Yaoong | 3.6 | 0.3 | NaN |
| **four** | 2015 | MungMung | 2.4 | 0.4 | -1.5 |

In [65]:

```
# 아래 방법을 권장한다.

# .loc 또는 .iloc 함수를 사용하는 방법.

df.loc['two'] # 반환 형태는 Series
```

Out[65]:

```
Info

year          2015

names         Yaoong

points         1.7
```

```
penalty          0.2
debt            -1.2
Name: two, dtype: object
```

In [67]:

```
df.loc['two':'four']
```

Out[67]:

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| **Order** | | | | | |
| **two** | 2015 | Yaoong | 1.7 | 0.2 | -1.2 |
| **three** | 2016 | Yaoong | 3.6 | 0.3 | NaN |
| **four** | 2015 | MungMung | 2.4 | 0.4 | -1.5 |

In [69]:

```
df.loc['two':'four', 'points']
```

Out[69]:

```
Order
two          1.7
three        3.6
four         2.4
Name: points, dtype: float64
```

In [70]:

```
df.loc[:,'year'] # == df['year']
```

Out[70]:

```
Order
one          2014
two          2015
three        2016
four         2015
five         2016
Name: year, dtype: int64
```

In [71]:

```
df.loc[:,['year','names']]
```

Out[71]:

| Info | year | names |
|---|---|---|
| **Order** | | |
| **one** | 2014 | Yaoong |
| **two** | 2015 | Yaoong |
| **three** | 2016 | Yaoong |
| **four** | 2015 | MungMung |
| **five** | 2016 | MungMung |

In [72]:

```
df.loc['three':'five','year':'penalty']
```

Out[72]:

| Info | year | names | points | penalty |
|---|---|---|---|---|
| **Order** | | | | |
| **three** | 2016 | Yaoong | 3.6 | 0.3 |
| **four** | 2015 | MungMung | 2.4 | 0.4 |
| **five** | 2016 | MungMung | 2.9 | 0.5 |

In [73]:

```
# 새로운 행 삽입하기

df.loc['six',:] = [2013,'Jun',4.0,0.1,2.1]
```

In [74]:

```
df
```

Out[74]:

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| **Order** | | | | | |
| **one** | 2014.0 | Yaoong | 1.5 | 0.1 | NaN |
| **two** | 2015.0 | Yaoong | 1.7 | 0.2 | -1.2 |
| **three** | 2016.0 | Yaoong | 3.6 | 0.3 | NaN |
| **four** | 2015.0 | MungMung | 2.4 | 0.4 | -1.5 |
| **five** | 2016.0 | MungMung | 2.9 | 0.5 | -1.7 |
| **six** | 2013.0 | Jun | 4.0 | 0.1 | 2.1 |

In [79]:

```
# .iloc 사용:: index 번호를 사용한다.

df.iloc[3] # 3 번째 행을 가져온다.
```

Out[79]:

```
Info

year              2015

names          MungMung

points             2.4

penalty            0.4

debt              -1.5

Name: four, dtype: object
```

In [80]:

```
df.iloc[3:5, 0:2]
```

Out[80]:

| Info | year | names |
|---|---|---|
| **Order** | | |
| **four** | 2015.0 | MungMung |
| **five** | 2016.0 | MungMung |

In [81]:

```
df.iloc[[0,1,3], [1,2]]
```

Out[81]:

| Info | names | points |
|---|---|---|
| **Order** | | |
| **one** | Yaoong | 1.5 |
| **two** | Yaoong | 1.7 |
| **four** | MungMung | 2.4 |

In [83]:

```
df.iloc[:,1:4]
```

Out[83]:

| Info | names | points | penalty |
|---|---|---|---|
| Order | | | |
| one | Yaoong | 1.5 | 0.1 |
| two | Yaoong | 1.7 | 0.2 |
| three | Yaoong | 3.6 | 0.3 |
| four | MungMung | 2.4 | 0.4 |
| five | MungMung | 2.9 | 0.5 |
| six | Jun | 4.0 | 0.1 |

In [84]:

```
df.iloc[1,1]
```

Out[84]:

```
'Yaoong'
```

## ▶ DataFrame 에서의 boolean Indexing¶

In [85]:

```
df
```

Out[85]:

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| Order | | | | | |
| one | 2014.0 | Yaoong | 1.5 | 0.1 | NaN |
| two | 2015.0 | Yaoong | 1.7 | 0.2 | -1.2 |
| three | 2016.0 | Yaoong | 3.6 | 0.3 | NaN |
| four | 2015.0 | MungMung | 2.4 | 0.4 | -1.5 |
| five | 2016.0 | MungMung | 2.9 | 0.5 | -1.7 |
| six | 2013.0 | Jun | 4.0 | 0.1 | 2.1 |

In [87]:

```
# year 가 2014 보다 큰 boolean data

df['year'] > 2014
```

Out[87]:

```
Order
```

```
one      False
two       True
three     True
four      True
five      True
six      False
Name: year, dtype: bool
```

In [89]:

```
# year 가 2014 보다 큰 모든 행의 값
df.loc[df['year']>2014,:]
```

Out[89]:

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| **Order** | | | | | |
| **two** | 2015.0 | Yaoong | 1.7 | 0.2 | -1.2 |
| **three** | 2016.0 | Yaoong | 3.6 | 0.3 | NaN |
| **four** | 2015.0 | MungMung | 2.4 | 0.4 | -1.5 |
| **five** | 2016.0 | MungMung | 2.9 | 0.5 | -1.7 |

In [90]:

```
df.loc[df['names'] == 'Yaoong',['names','points']]
```

Out[90]:

| Info | names | points |
|---|---|---|
| **Order** | | |
| **one** | Yaoong | 1.5 |
| **two** | Yaoong | 1.7 |
| **three** | Yaoong | 3.6 |

In [92]:

```
# numpy 에서와 같이 논리연산을 응용할 수 있다.
df.loc[(df['points']>2)&(df['points']<3),:]
```

Out[92]:

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| **Order** | | | | | |
| **four** | 2015.0 | MungMung | 2.4 | 0.4 | -1.5 |

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| Order | | | | | |
| five | 2016.0 | MungMung | 2.9 | 0.5 | -1.7 |

In [93]:

```
# 새로운 값을 대입할 수도 있다.

df.loc[df['points'] > 3, 'penalty'] = 0
```

In [94]:

```
df
```

Out[94]:

| Info | year | names | points | penalty | debt |
|---|---|---|---|---|---|
| Order | | | | | |
| one | 2014.0 | Yaoong | 1.5 | 0.1 | NaN |
| two | 2015.0 | Yaoong | 1.7 | 0.2 | -1.2 |
| three | 2016.0 | Yaoong | 3.6 | 0.0 | NaN |
| four | 2015.0 | MungMung | 2.4 | 0.4 | -1.5 |
| five | 2016.0 | MungMung | 2.9 | 0.5 | -1.7 |
| six | 2013.0 | Jun | 4.0 | 0.0 | 2.1 |

# ▶Data¶

In [95]:

```
# DataFrame 을 만들때 index, column 을 설정하지 않으면 기본값으로 0 부터 시작하는 정수형 숫자로 입력된다.

df = pd.DataFrame(np.random.randn(6,4))

df
```

Out[95]:

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0.682000 | -0.570393 | -1.602829 | -1.316469 |
| 1 | -1.176203 | 0.171527 | 0.387018 | 1.027615 |
| 2 | -0.263178 | -0.212049 | 1.006856 | 0.096111 |
| 3 | 2.679378 | 0.347145 | 0.548144 | 0.826258 |
| 4 | -0.249877 | 0.018721 | -0.393715 | 0.302361 |
| 5 | 0.851420 | -0.440360 | -0.345895 | 1.055936 |

In [98]:

```
df.columns = ['A', 'B', 'C', 'D']

df.index = pd.date_range('20160701', periods=6)

#pandas 에서 제공하는 date range 함수는 datetime 자료형으로 구성된, 날짜 시각등을 알 수 있는 자료형을 만드는 함수

df.index
```

Out[98]:

```
DatetimeIndex(['2016-07-01', '2016-07-02', '2016-07-03', '2016-07-04',

               '2016-07-05', '2016-07-06'],

              dtype='datetime64[ns]', freq='D')
```

In [99]:

```
df
```

Out[99]:

| | A | B | C | D |
|---|---|---|---|---|
| **2016-07-01** | 0.682000 | -0.570393 | -1.602829 | -1.316469 |
| **2016-07-02** | -1.176203 | 0.171527 | 0.387018 | 1.027615 |
| **2016-07-03** | -0.263178 | -0.212049 | 1.006856 | 0.096111 |
| **2016-07-04** | 2.679378 | 0.347145 | 0.548144 | 0.826258 |
| **2016-07-05** | -0.249877 | 0.018721 | -0.393715 | 0.302361 |
| **2016-07-06** | 0.851420 | -0.440360 | -0.345895 | 1.055936 |

In [101]:

```
# np.nan 은 NaN 값을 의미한다.

df['F'] = [1.0, np.nan, 3.5, 6.1, np.nan, 7.0]

df
```

Out[101]:

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2016-07-01** | 0.682000 | -0.570393 | -1.602829 | -1.316469 | 1.0 |
| **2016-07-02** | -1.176203 | 0.171527 | 0.387018 | 1.027615 | NaN |
| **2016-07-03** | -0.263178 | -0.212049 | 1.006856 | 0.096111 | 3.5 |
| **2016-07-04** | 2.679378 | 0.347145 | 0.548144 | 0.826258 | 6.1 |
| **2016-07-05** | -0.249877 | 0.018721 | -0.393715 | 0.302361 | NaN |
| **2016-07-06** | 0.851420 | -0.440360 | -0.345895 | 1.055936 | 7.0 |

**NaN 없애기**

In [102]:

```
# 행의 값중 하나라도 nan 인 경우 그 행을 없앤다.

df.dropna(how='any')
```

Out[102]:

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2016-07-01** | 0.682000 | -0.570393 | -1.602829 | -1.316469 | 1.0 |
| **2016-07-03** | -0.263178 | -0.212049 | 1.006856 | 0.096111 | 3.5 |
| **2016-07-04** | 2.679378 | 0.347145 | 0.548144 | 0.826258 | 6.1 |
| **2016-07-06** | 0.851420 | -0.440360 | -0.345895 | 1.055936 | 7.0 |

In [103]:

```
# 행의 값의 모든 값이 nan 인 경우 그 행으 없앤다.

df.dropna(how='all')
```

Out[103]:

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2016-07-01** | 0.682000 | -0.570393 | -1.602829 | -1.316469 | 1.0 |
| **2016-07-02** | -1.176203 | 0.171527 | 0.387018 | 1.027615 | NaN |
| **2016-07-03** | -0.263178 | -0.212049 | 1.006856 | 0.096111 | 3.5 |
| **2016-07-04** | 2.679378 | 0.347145 | 0.548144 | 0.826258 | 6.1 |
| **2016-07-05** | -0.249877 | 0.018721 | -0.393715 | 0.302361 | NaN |
| **2016-07-06** | 0.851420 | -0.440360 | -0.345895 | 1.055936 | 7.0 |

**주의** drop 함수는 특정 행 또는 열을 drop 하고난 DataFrame 을 반환한다.

즉, 반환을 받지 않으면 기존의 DataFrame 은 그대로이다.

아니면, inplace=True 라는 인자를 추가하여, 반환을 받지 않고서도

기존의 DataFrame 이 변경되도록 한다.

In [104]:

```
# nan 값에 값 넣기

df.fillna(value=0.5)
```

Out[104]:

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2016-07-01** | 0.682000 | -0.570393 | -1.602829 | -1.316469 | 1.0 |
| **2016-07-02** | -1.176203 | 0.171527 | 0.387018 | 1.027615 | 0.5 |

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2016-07-03** | -0.263178 | -0.212049 | 1.006856 | 0.096111 | 3.5 |
| **2016-07-04** | 2.679378 | 0.347145 | 0.548144 | 0.826258 | 6.1 |
| **2016-07-05** | -0.249877 | 0.018721 | -0.393715 | 0.302361 | 0.5 |
| **2016-07-06** | 0.851420 | -0.440360 | -0.345895 | 1.055936 | 7.0 |

In [107]:

```
# nan 값인지 확인하기

df.isnull()
```

Out[107]:

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2016-07-01** | False | False | False | False | False |
| **2016-07-02** | False | False | False | False | True |
| **2016-07-03** | False | False | False | False | False |
| **2016-07-04** | False | False | False | False | False |
| **2016-07-05** | False | False | False | False | True |
| **2016-07-06** | False | False | False | False | False |

In [109]:

```
# F 열에서 nan 값을 포함하는 행만 추출하기

df.loc[df.isnull()['F'],:]
```

Out[109]:

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2016-07-02** | -1.176203 | 0.171527 | 0.387018 | 1.027615 | NaN |
| **2016-07-05** | -0.249877 | 0.018721 | -0.393715 | 0.302361 | NaN |

In [110]:

```
pd.to_datetime('20160701')
```

Out[110]:

```
Timestamp('2016-07-01 00:00:00')
```

In [111]:

```
# 특정 행 drop 하기

df.drop(pd.to_datetime('20160701'))
```

Out[111]:

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2016-07-02** | -1.176203 | 0.171527 | 0.387018 | 1.027615 | NaN |
| **2016-07-03** | -0.263178 | -0.212049 | 1.006856 | 0.096111 | 3.5 |
| **2016-07-04** | 2.679378 | 0.347145 | 0.548144 | 0.826258 | 6.1 |
| **2016-07-05** | -0.249877 | 0.018721 | -0.393715 | 0.302361 | NaN |
| **2016-07-06** | 0.851420 | -0.440360 | -0.345895 | 1.055936 | 7.0 |

In [112]:

```
# 2 개 이상도 가능

df.drop([pd.to_datetime('20160702'),pd.to_datetime('20160704')])
```

Out[112]:

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2016-07-01** | 0.682000 | -0.570393 | -1.602829 | -1.316469 | 1.0 |
| **2016-07-03** | -0.263178 | -0.212049 | 1.006856 | 0.096111 | 3.5 |
| **2016-07-05** | -0.249877 | 0.018721 | -0.393715 | 0.302361 | NaN |
| **2016-07-06** | 0.851420 | -0.440360 | -0.345895 | 1.055936 | 7.0 |

In [113]:

```
# 특정 열 삭제하기

df.drop('F', axis = 1)
```

Out[113]:

| | A | B | C | D |
|---|---|---|---|---|
| **2016-07-01** | 0.682000 | -0.570393 | -1.602829 | -1.316469 |
| **2016-07-02** | -1.176203 | 0.171527 | 0.387018 | 1.027615 |
| **2016-07-03** | -0.263178 | -0.212049 | 1.006856 | 0.096111 |
| **2016-07-04** | 2.679378 | 0.347145 | 0.548144 | 0.826258 |
| **2016-07-05** | -0.249877 | 0.018721 | -0.393715 | 0.302361 |
| **2016-07-06** | 0.851420 | -0.440360 | -0.345895 | 1.055936 |

In [114]:

```
# 2 개 이상의 열도 가능

df.drop(['B','D'], axis = 1)
```

Out[114]:

| | A | C | F |
|---|---|---|---|
| **2016-07-01** | 0.682000 | -1.602829 | 1.0 |
| **2016-07-02** | -1.176203 | 0.387018 | NaN |
| **2016-07-03** | -0.263178 | 1.006856 | 3.5 |

| | A | C | F |
|---|---|---|---|
| **2016-07-04** | 2.679378 | 0.548144 | 6.1 |
| **2016-07-05** | -0.249877 | -0.393715 | NaN |
| **2016-07-06** | 0.851420 | -0.345895 | 7.0 |

## ▶ Data 분석용 함수들¶

In [115]:

```
data = [[1.4, np.nan],
        [7.1, -4.5],
    [np.nan, np.nan],
    [0.75, -1.3]]
df = pd.DataFrame(data, columns=["one", "two"], index=["a", "b", "c", "d"])
```

In [116]:

```
df
```

Out[116]:

| | one | two |
|---|---|---|
| **a** | 1.40 | NaN |
| **b** | 7.10 | -4.5 |
| **c** | NaN | NaN |
| **d** | 0.75 | -1.3 |

In [118]:

```
# 행방향으로의 합(즉, 각 열의 합)
df.sum(axis=0)
```

Out[118]:

```
one    9.25
two   -5.80
dtype: float64
```

In [119]:

```
# 열방향으로의 합(즉, 각 행의 합)
df.sum(axis=1)
```

Out[119]:

```
a    1.40
b    2.60
c    0.00
d   -0.55
dtype: float64
```

이때, 위에서 볼 수 있듯이 NaN 값은 배제하고 계산한다.

NaN 값을 배제하지 않고 계산하려면 아래와 같이 skipna 에 대해 false 를 지정해준다.

In [123]:

```
df.sum(axis=1, skipna=False)
```

Out[123]:

```
a     NaN
b    2.60
c     NaN
d   -0.55
dtype: float64
```

In [120]:

```
# 특정 행 또는 특정 열에서만 계산하기
df['one'].sum()
```

Out[120]:

```
9.25
```

In [121]:

```
df.loc['b'].sum()
```

Out[121]:

```
2.5999999999999996
```

## √ pandas 에서 DataFrame 에 적용되는 함수들

sum() 함수 이외에도 pandas 에서 DataFrame 에 적용되는 함수는 다음의 것들이 있다.

count 전체 성분의 (NaN 이 아닌) 값의 갯수를 계산

min, max 전체 성분의 최솟, 최댓값을 계산

argmin, argmax 전체 성분의 최솟값, 최댓값이 위치한 (정수)인덱스를 반환

idxmin, idxmax 전체 인덱스 중 최솟값, 최댓값을 반환

quantile 전체 성분의 특정 사분위수에 해당하는 값을 반환 (0~1 사이)

sum 전체 성분의 합을 계산

mean 전체 성분의 평균을 계산

median 전체 성분의 중간값을 반환

mad 전체 성분의 평균값으로부터의 절대 편차(absolute deviation)의 평균을 계산

std, var 전체 성분의 표준편차, 분산을 계산

cumsum 맨 첫 번째 성분부터 각 성분까지의 누적합을 계산 (0 에서부터 계속 더해짐)

cumprod 맨 첫번째 성분부터 각 성분까지의 누적곱을 계산 (1 에서부터 계속 곱해짐)

In [125]:

```
df2 = pd.DataFrame(np.random.randn(6, 4),
                   columns=["A", "B", "C", "D"],
                   index=pd.date_range("20160701", periods=6))
df2
```

Out[125]:

| | A | B | C | D |
|---|---|---|---|---|
| **2016-07-01** | 1.024359 | 0.213159 | 0.277114 | -0.304599 |
| **2016-07-02** | -1.693934 | 0.619885 | -1.620676 | 0.067814 |
| **2016-07-03** | -0.216422 | 0.659557 | 0.342833 | -0.141637 |
| **2016-07-04** | -0.873492 | 0.669932 | -0.515944 | 0.280424 |
| **2016-07-05** | -0.923094 | -1.174652 | 2.678657 | -0.663322 |
| **2016-07-06** | -1.580576 | 0.539906 | -1.900090 | -0.428551 |

In [127]:

```
# A 열과 B 열의 상관계수 구하기

df2['A'].corr(df2['B'])
```

Out[127]:

```
-0.06715327766901227
```

In [128]:

```
# B 열과 C 열의 공분산 구하기

df2['B'].cov(df2['C'])
```

Out[128]:

```
-1.0099019967454226
```

## √ 정렬함수 및 기타함수¶

In [130]:

```
dates = df2.index

random_dates = np.random.permutation(dates)

df2 = df2.reindex(index=random_dates, columns=["D", "B", "C", "A"])

df2
```

Out[130]:

| | D | B | C | A |
|---|---|---|---|---|
| **2016-07-02** | 0.067814 | 0.619885 | -1.620676 | -1.693934 |
| **2016-07-06** | -0.428551 | 0.539906 | -1.900090 | -1.580576 |
| **2016-07-03** | -0.141637 | 0.659557 | 0.342833 | -0.216422 |
| **2016-07-01** | -0.304599 | 0.213159 | 0.277114 | 1.024359 |
| **2016-07-05** | -0.663322 | -1.174652 | 2.678657 | -0.923094 |
| **2016-07-04** | 0.280424 | 0.669932 | -0.515944 | -0.873492 |

In [131]:

```
# index 와 column 의 순서가 섞여있다.

# 이때 index 가 오름차순이 되도록 정렬해보자
```

```
df2.sort_index(axis=0)
```

Out[131]:

| | D | B | C | A |
|---|---|---|---|---|
| **2016-07-01** | -0.304599 | 0.213159 | 0.277114 | 1.024359 |
| **2016-07-02** | 0.067814 | 0.619885 | -1.620676 | -1.693934 |
| **2016-07-03** | -0.141637 | 0.659557 | 0.342833 | -0.216422 |
| **2016-07-04** | 0.280424 | 0.669932 | -0.515944 | -0.873492 |
| **2016-07-05** | -0.663322 | -1.174652 | 2.678657 | -0.923094 |
| **2016-07-06** | -0.428551 | 0.539906 | -1.900090 | -1.580576 |

In [132]:

```
# column 을 기준으로?

df2.sort_index(axis=1)
```

Out[132]:

| | A | B | C | D |
|---|---|---|---|---|
| **2016-07-02** | -1.693934 | 0.619885 | -1.620676 | 0.067814 |
| **2016-07-06** | -1.580576 | 0.539906 | -1.900090 | -0.428551 |
| **2016-07-03** | -0.216422 | 0.659557 | 0.342833 | -0.141637 |
| **2016-07-01** | 1.024359 | 0.213159 | 0.277114 | -0.304599 |
| **2016-07-05** | -0.923094 | -1.174652 | 2.678657 | -0.663322 |
| **2016-07-04** | -0.873492 | 0.669932 | -0.515944 | 0.280424 |

In [134]:

```
# 내림차순으로는?

df2.sort_index(axis=1, ascending=False)
```

Out[134]:

| | D | C | B | A |
|---|---|---|---|---|
| **2016-07-02** | 0.067814 | -1.620676 | 0.619885 | -1.693934 |
| **2016-07-06** | -0.428551 | -1.900090 | 0.539906 | -1.580576 |
| **2016-07-03** | -0.141637 | 0.342833 | 0.659557 | -0.216422 |
| **2016-07-01** | -0.304599 | 0.277114 | 0.213159 | 1.024359 |
| **2016-07-05** | -0.663322 | 2.678657 | -1.174652 | -0.923094 |
| **2016-07-04** | 0.280424 | -0.515944 | 0.669932 | -0.873492 |

In [135]:

```
# 값 기준 정렬하기

# D 열의 값이 오름차순이 되도록 정렬하기
```

```
df2.sort_values(by='D')
```

Out[135]:

| | D | B | C | A |
|---|---|---|---|---|
| **2016-07-05** | -0.663322 | -1.174652 | 2.678657 | -0.923094 |
| **2016-07-06** | -0.428551 | 0.539906 | -1.900090 | -1.580576 |
| **2016-07-01** | -0.304599 | 0.213159 | 0.277114 | 1.024359 |
| **2016-07-03** | -0.141637 | 0.659557 | 0.342833 | -0.216422 |
| **2016-07-02** | 0.067814 | 0.619885 | -1.620676 | -1.693934 |
| **2016-07-04** | 0.280424 | 0.669932 | -0.515944 | -0.873492 |

In [136]:

```
# B 열의 값이 내림차순이 되도록 정렬하기

df2.sort_values(by='B', ascending=False)
```

Out[136]:

| | D | B | C | A |
|---|---|---|---|---|
| **2016-07-04** | 0.280424 | 0.669932 | -0.515944 | -0.873492 |
| **2016-07-03** | -0.141637 | 0.659557 | 0.342833 | -0.216422 |
| **2016-07-02** | 0.067814 | 0.619885 | -1.620676 | -1.693934 |
| **2016-07-06** | -0.428551 | 0.539906 | -1.900090 | -1.580576 |
| **2016-07-01** | -0.304599 | 0.213159 | 0.277114 | 1.024359 |
| **2016-07-05** | -0.663322 | -1.174652 | 2.678657 | -0.923094 |

In [138]:

```
df2["E"] = np.random.randint(0, 6, size=6)

df2["F"] = ["alpha", "beta", "gamma", "gamma", "alpha", "gamma"]

df2
```

Out[138]:

| | D | B | C | A | E | F |
|---|---|---|---|---|---|---|
| **2016-07-02** | 0.067814 | 0.619885 | -1.620676 | -1.693934 | 1 | alpha |
| **2016-07-06** | -0.428551 | 0.539906 | -1.900090 | -1.580576 | 1 | beta |
| **2016-07-03** | -0.141637 | 0.659557 | 0.342833 | -0.216422 | 3 | gamma |
| **2016-07-01** | -0.304599 | 0.213159 | 0.277114 | 1.024359 | 4 | gamma |
| **2016-07-05** | -0.663322 | -1.174652 | 2.678657 | -0.923094 | 5 | alpha |
| **2016-07-04** | 0.280424 | 0.669932 | -0.515944 | -0.873492 | 0 | gamma |

In [139]:

```
# E 열과 F 열을 동시에 고려하여, 오름차순으로 하려면?
```

```
df2.sort_values(by=['E','F'])
```

Out[139]:

| | D | B | C | A | E | F |
|---|---|---|---|---|---|---|
| **2016-07-04** | 0.280424 | 0.669932 | -0.515944 | -0.873492 | 0 | gamma |
| **2016-07-02** | 0.067814 | 0.619885 | -1.620676 | -1.693934 | 1 | alpha |
| **2016-07-06** | -0.428551 | 0.539906 | -1.900090 | -1.580576 | 1 | beta |
| **2016-07-03** | -0.141637 | 0.659557 | 0.342833 | -0.216422 | 3 | gamma |
| **2016-07-01** | -0.304599 | 0.213159 | 0.277114 | 1.024359 | 4 | gamma |
| **2016-07-05** | -0.663322 | -1.174652 | 2.678657 | -0.923094 | 5 | alpha |

In [140]:

```
# 지정한 행 또는 열에서 중복값을 제외한 유니크한 값만 얻기

df2['F'].unique()
```

Out[140]:

```
array(['alpha', 'beta', 'gamma'], dtype=object)
```

In [142]:

```
# 지정한 행 또는 열에서 값에 따른 개수 얻기

df2['F'].value_counts()
```

Out[142]:

```
gamma    3

alpha    2

beta     1

Name: F, dtype: int64
```

In [144]:

```
# 지정한 행 또는 열에서 입력한 값이 있는지 확인하기

df2['F'].isin(['alpha','beta'])

# 아래와 같이 응용할 수 있다.
```

Out[144]:

```
2016-07-02     True

2016-07-06     True
```

```
2016-07-03    False
2016-07-01    False
2016-07-05     True
2016-07-04    False
Name: F, dtype: bool
```

In [146]:

```
# F 열의 값이 alpha 나 beta 인 모든 행 구하기
df2.loc[df2['F'].isin(['alpha','beta']),:]
```

Out[146]:

| | D | B | C | A | E | F |
|---|---|---|---|---|---|---|
| **2016-07-02** | 0.067814 | 0.619885 | -1.620676 | -1.693934 | 1 | alpha |
| **2016-07-06** | -0.428551 | 0.539906 | -1.900090 | -1.580576 | 1 | beta |
| **2016-07-05** | -0.663322 | -1.174652 | 2.678657 | -0.923094 | 5 | alpha |

사용자가 직접 만든 함수를 적용하기

In [149]:

```
df3 = pd.DataFrame(np.random.randn(4, 3), columns=["b", "d", "e"],
                   index=["Seoul", "Incheon", "Busan", "Daegu"])
df3
```

Out[149]:

| | b | d | e |
|---|---|---|---|
| **Seoul** | 0.806401 | -0.101789 | -0.201956 |
| **Incheon** | 0.666876 | -2.144428 | -0.013760 |
| **Busan** | 1.811228 | 0.225317 | 1.445495 |
| **Daegu** | -0.608445 | 0.128199 | -0.325935 |

In [150]:

```
func = lambda x: x.max() - x.min()
```

In [151]:

```
df3.apply(func, axis=0)
```

Out[151]:

```
b    2.419673
```

```
d    2.369745
e    1.771431
dtype: float64
```

# Matplotlib 기본

matplotlib 은 다양한 데이터를 많은 방법으로 도식화 할 수 있도록 하는 파이썬 라이브러리로써, 우리는 matplotlib 의 pyplot 을 이용하게 된다.

이는 mathworks 에서 개발한 매트랩(MATLAB)과 비슷한 형태를 가지고 있다.

**matplotlib 을 이용하면 우리가 이전에 알아본 numpy 나 pandas 에서 사용되는 자료구조를 쉽게 시각화 할 수 있다.**

matplotlib 을 사용하기 위해서는 먼저 matplotlib 을 설치하고 아래와 같이 import 를 해주어야 한다.

***import matplotlib***

***import matplotlib.pyplot as plt***

또한 jupyter notebook 에서 그래프를 제대로 확인하기 위해서는 아래와 같은 매직 커맨드를 작성해줘야 한다.

***%matplotlib inline***

이러한 매직커맨드는 맨 뒤에 inline 이외에도 우리가 아래에서 사용하는 nbagg 등의 다양한 속성이 있다.

## ▶ Matplotlib 개요

In [11]:

```
import matplotlib

import matplotlib.pyplot as plt

%matplotlib nbagg
```

```
#위와 같은 것이 %로 시작하는 것을 jupyter notebook 의 magic command 라고 한다.
# %who 는 변수명의 리스트를 보여주고 %magic 은 모든 매직명령어를 보여준다.
import numpy as np
import pandas as pd
```

# ▶ Plot 의 종류¶

## √ Line plot 그리기¶

In [12]:

```
# Series 를 통한 line plot 그리기
s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))
s
```

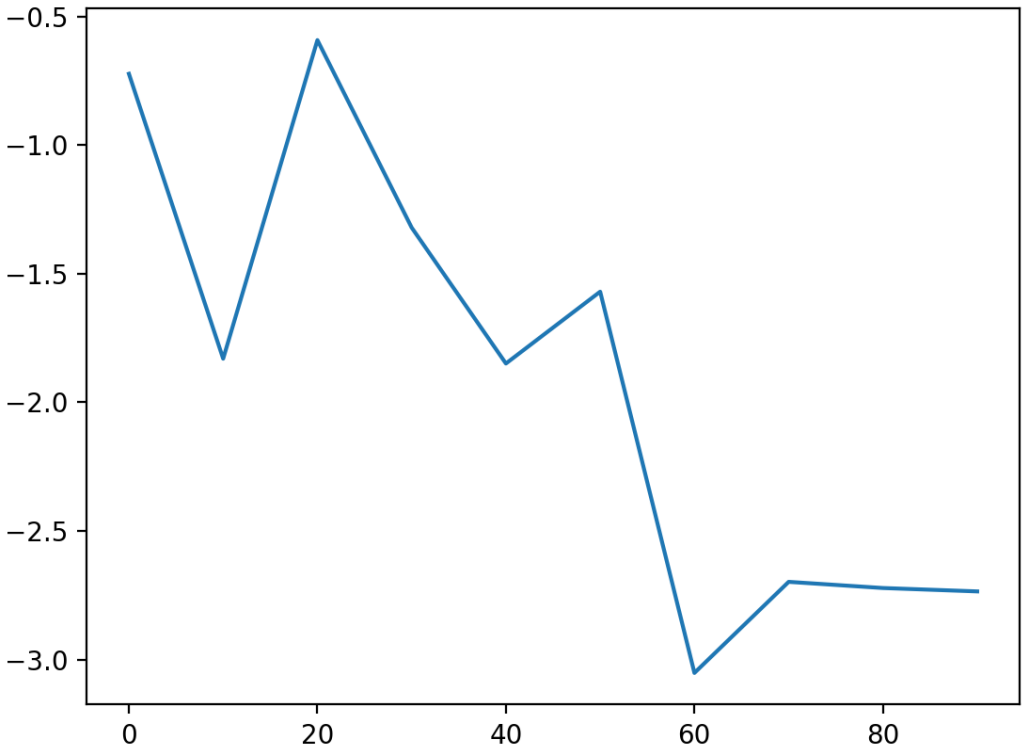
Out[12]:

```
0    -0.722984
10   -1.830547
20   -0.592063
30   -1.320894
40   -1.849281
50   -1.570039
60   -3.052119
70   -2.698092
80   -2.722053
90   -2.735229
dtype: float64
```

In [13]:

```
# 위에서 정의한 s 라는 시리즈에 대해서 line plot 을 그리고 싶다면?
```

```
s.plot()
```




Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x108ef0c88>
```

s 라는 Series 에서의 index 와 value 를 통해 그래프가 그려졌다.

그래프 우측상단의 전원버튼을 누르기 전까지 우리는 해당 그래프를 interactive 하게 조작할 수 있다.

In [14]:

```
# DataFrame 을 통한 line plot 그리기

df = pd.DataFrame(np.random.randn(10, 4).cumsum(axis=0),

                  columns=["A", "B", "C", "D"],

                  index=np.arange(0, 100, 10))

df
```

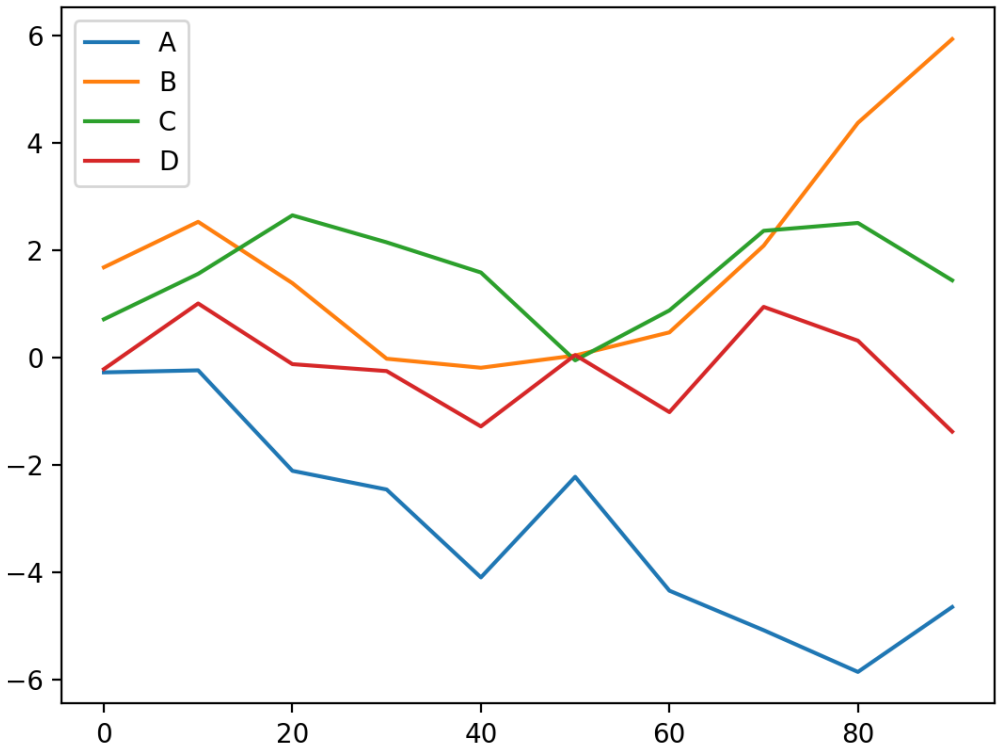
Out[14]:

| | A | B | C | D |
|---|---|---|---|---|
| **0** | -0.278464 | 1.680385 | 0.711803 | -0.216933 |
| **10** | -0.239848 | 2.527778 | 1.558551 | 1.006354 |

| | A | B | C | D |
|---|---|---|---|---|
| **20** | -2.112968 | 1.384759 | 2.648977 | -0.124528 |
| **30** | -2.461009 | -0.023573 | 2.145466 | -0.253936 |
| **40** | -4.098926 | -0.191797 | 1.583091 | -1.285248 |
| **50** | -2.224330 | 0.036316 | -0.053839 | 0.045480 |
| **60** | -4.346708 | 0.467878 | 0.877064 | -1.018642 |
| **70** | -5.083230 | 2.082973 | 2.360633 | 0.942955 |
| **80** | -5.860602 | 4.372568 | 2.506778 | 0.312459 |
| **90** | -4.651125 | 5.932881 | 1.437739 | -1.382153 |

In [16]:

```
df.plot()
```



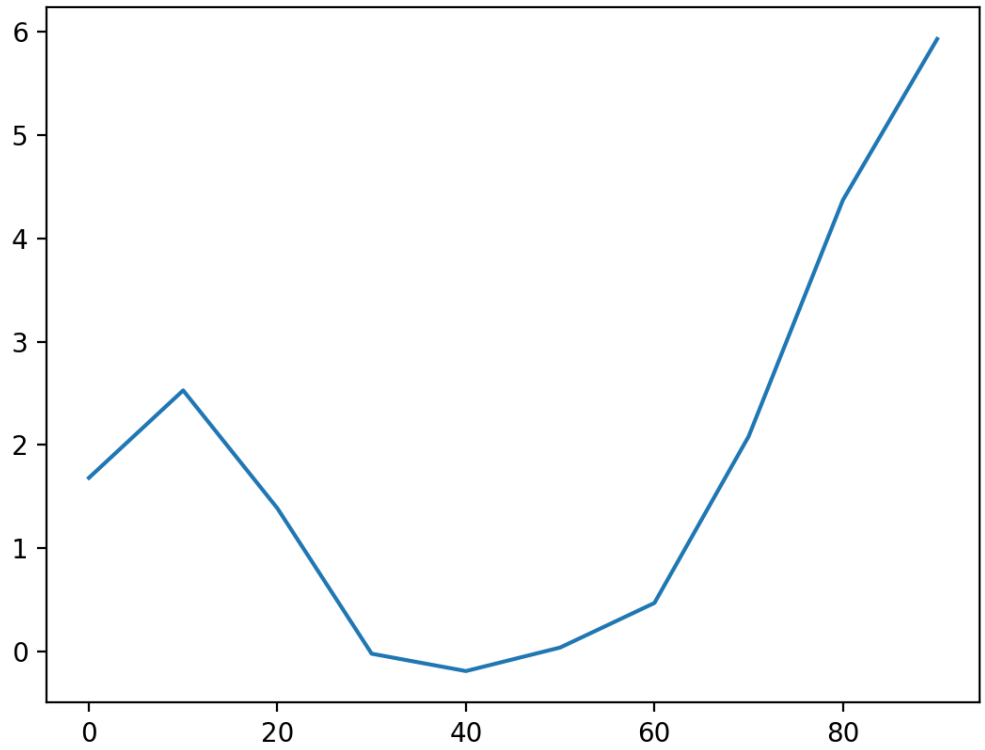

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x109df6710>
```

In [17]:

```
# 하나의 열에 대해서만 보고 싶다면?

df['B'].plot()
```




Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10a859b38>
```

## √ Bar plot 그리기¶

In [18]:

```
s2 = pd.Series(np.random.rand(16), index=list("abcdefghijklmnop"))
s2
```

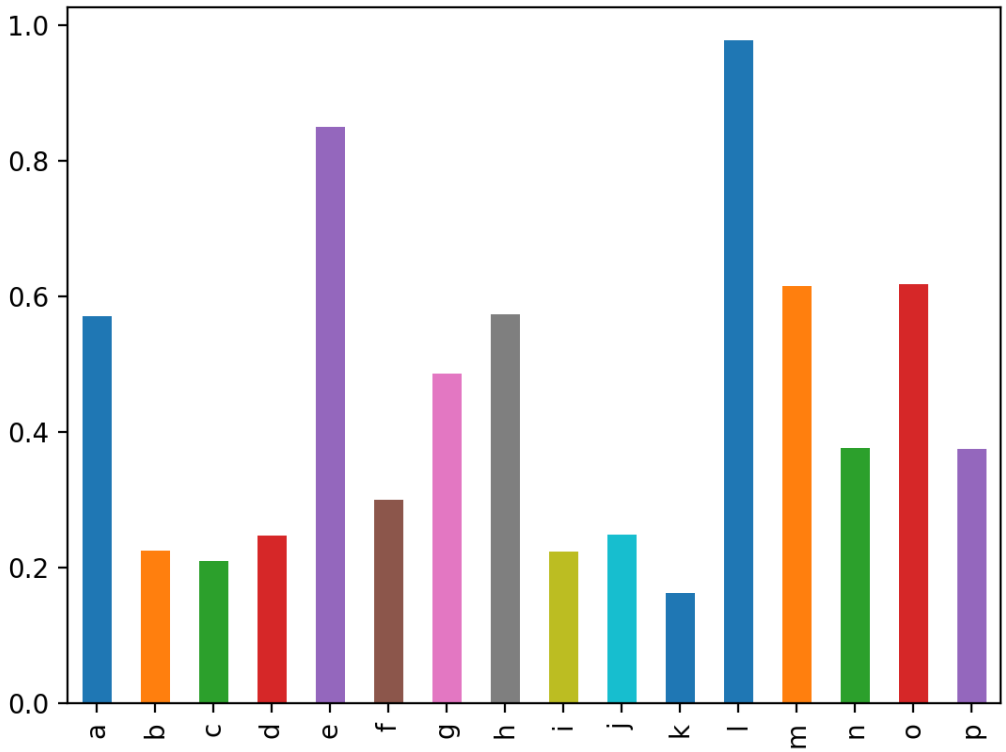
Out[18]:

```
a    0.571031
b    0.225421
c    0.210635
d    0.247162
e    0.850638
f    0.300911
```

```
g    0.485898
h    0.573721
i    0.223882
j    0.248352
k    0.163142
l    0.977120
m    0.615089
n    0.376035
o    0.618324
p    0.374877
dtype: float64
```

In [19]:

```
s2.plot(kind='bar')
```



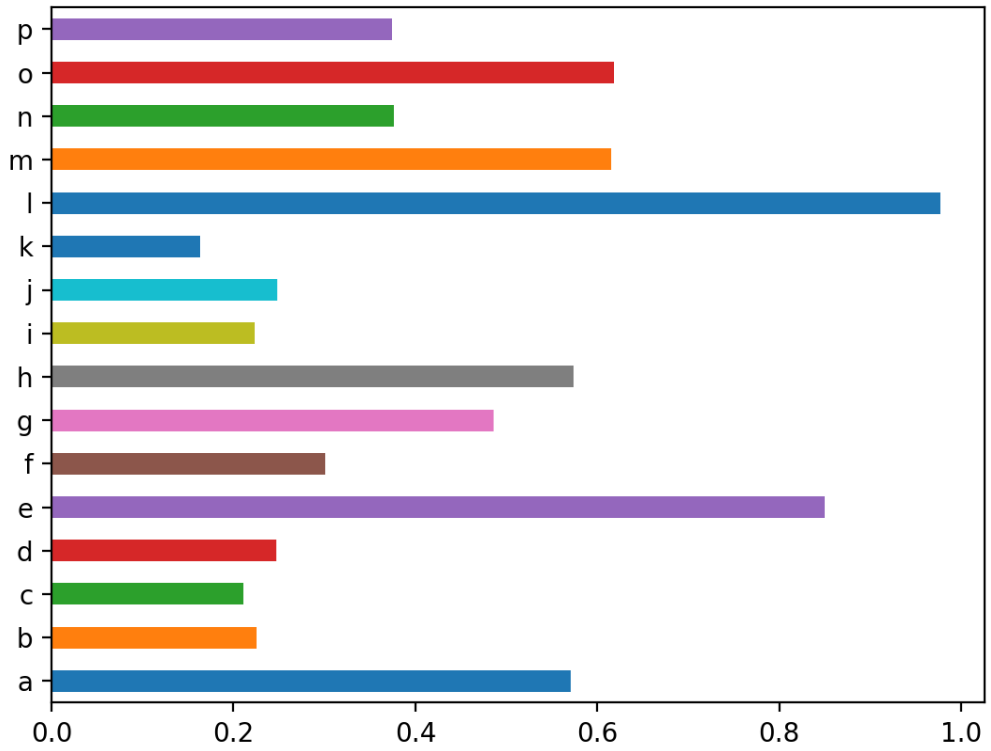

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10b26a208>
```

In [21]:

```
# 가로방향의 bar plot 그리기

s2.plot(kind='barh')
```




Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10c6f0908>
```

In [22]:

```
df2 = pd.DataFrame(np.random.rand(6, 4),

                   index=["one", "two", "three", "four", "five", "six"],

                   columns=pd.Index(["A", "B", "C", "D"], name="Genus"))

df2
```

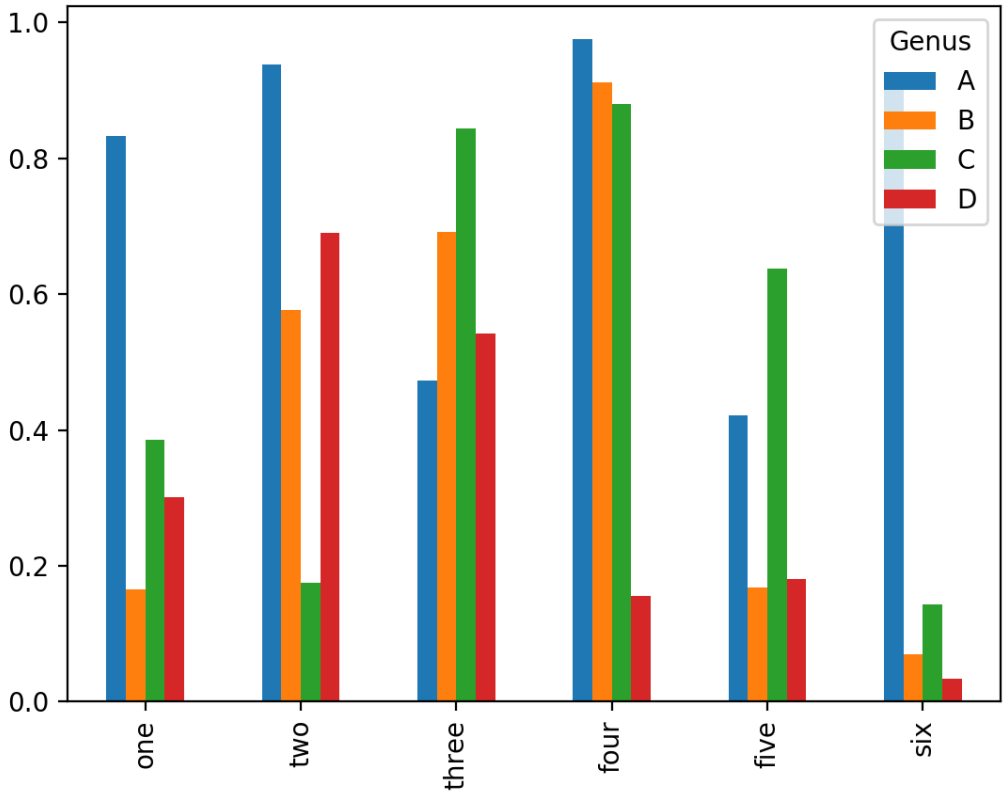
Out[22]:

| Genus | A | B | C | D |
|---|---|---|---|---|
| **one** | 0.832213 | 0.165459 | 0.385868 | 0.300776 |
| **two** | 0.937578 | 0.576798 | 0.175512 | 0.690425 |
| **three** | 0.473119 | 0.690937 | 0.844016 | 0.542061 |
| **four** | 0.974779 | 0.911599 | 0.880104 | 0.155459 |

| Genus | A | B | C | D |
|---|---|---|---|---|
| **five** | 0.421689 | 0.168038 | 0.637749 | 0.181037 |
| **six** | 0.921647 | 0.069688 | 0.143649 | 0.033414 |

In [23]:

```
df2.plot(kind='bar')
```

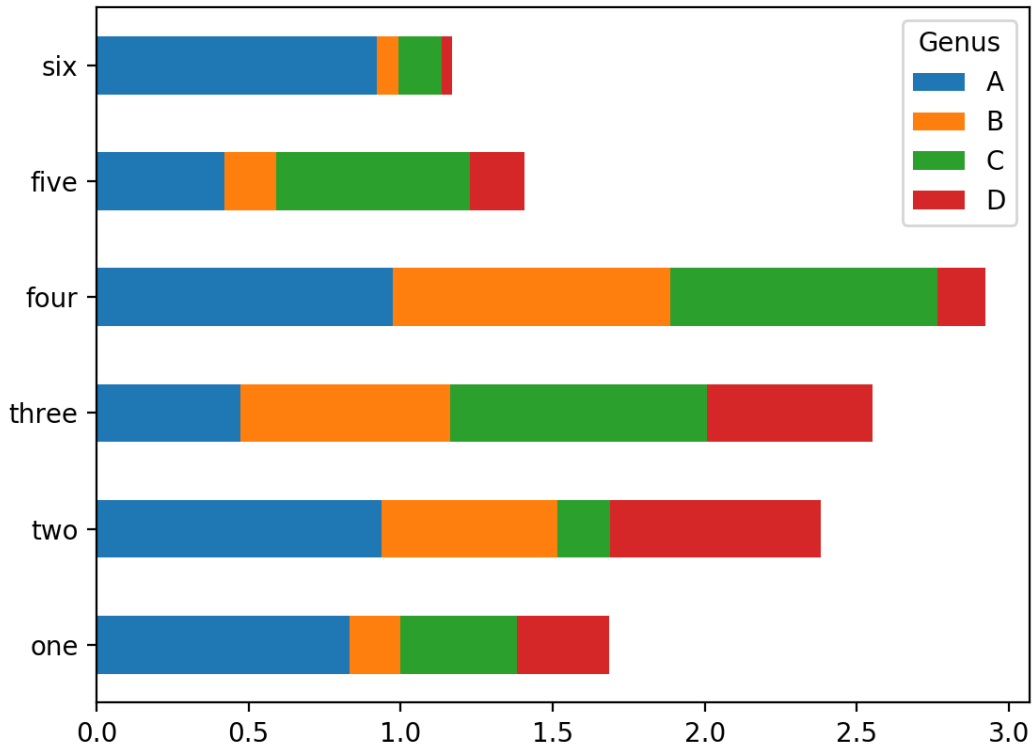



Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10d17e7f0>
```

In [28]:

```
df2.plot(kind='barh', stacked=True)
```




Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10feab8d0>
```

위와 같이 Stacked 속성을 True 로 설정하면, 하나의 인덱스에 대한 각 열의 값을 한줄로 쌓아서 나타내준다.

## √ Histogram 그리기¶

In [26]:

```
# histogram 은 index 가 필요없다.

s3 = pd.Series(np.random.normal(0, 1, size=200))

s3
```

Out[26]:

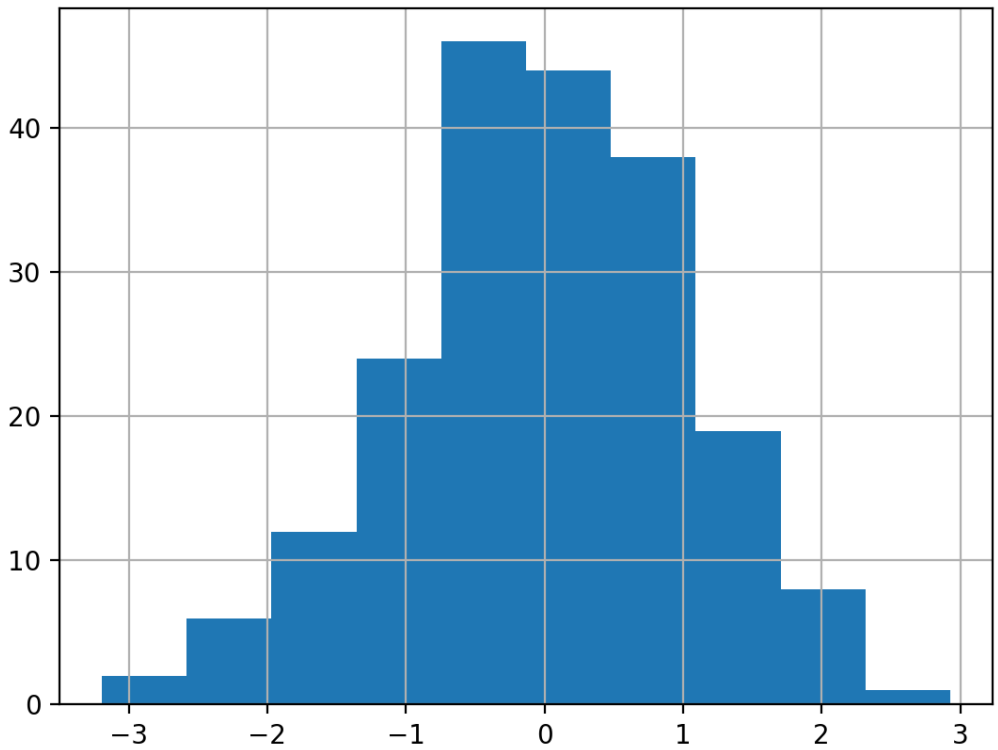
```
0      0.702157

1      1.868464
```

```
2         0.245772
3        -1.763628
.............................
197      -0.326276
198       1.579637
199       0.482489
Length: 200, dtype: float64
```

In [29]:

```
s3.hist()
```




Out[29]:

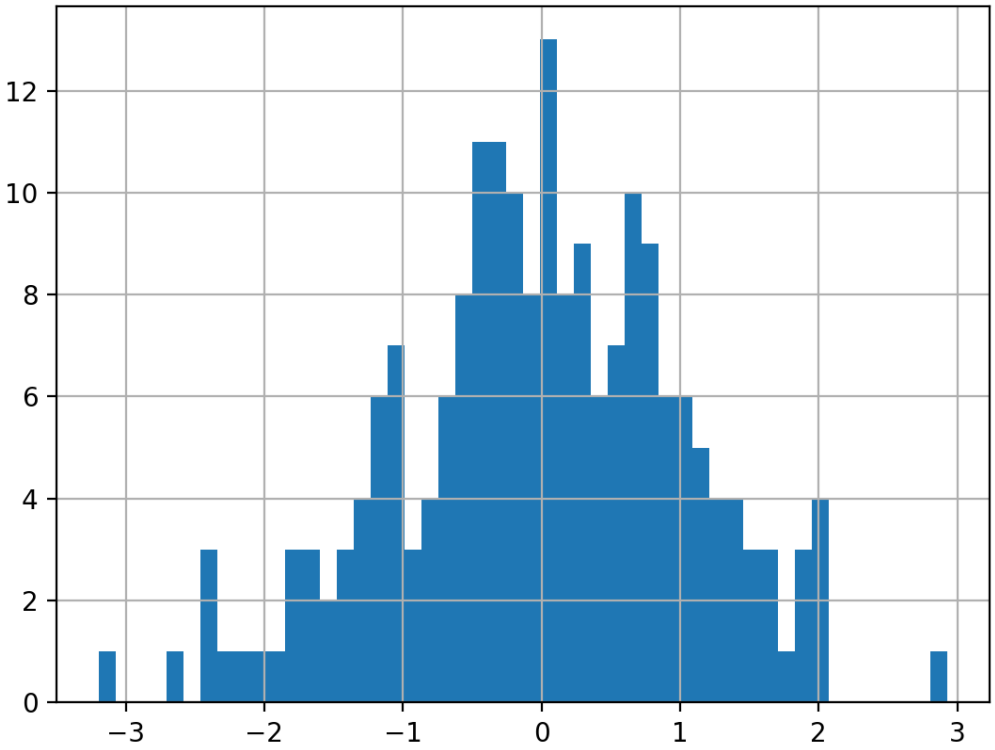
```
<matplotlib.axes._subplots.AxesSubplot at 0x10ffae2e8>
```

x 축의 구간 개수를 bin 이라고 한다.

이를 직접 설정할 수도 있다.

In [30]:

```
s3.hist(bins=50)
```



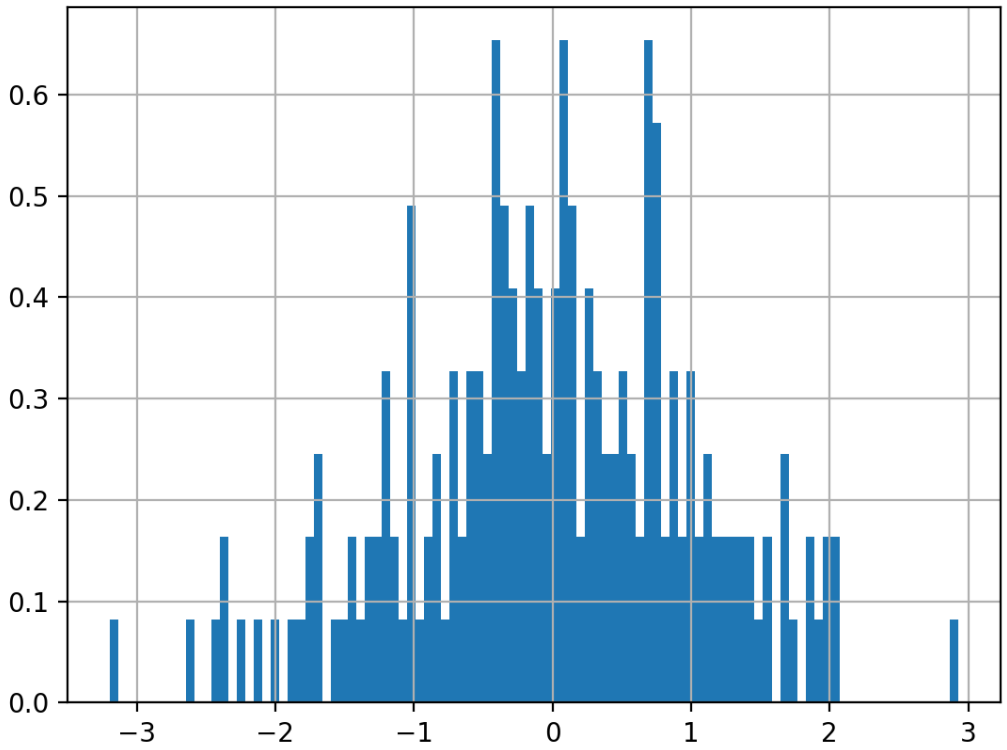

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1104cb978>
```

In [31]:

```
s3.hist(bins=100, normed=True)

# normed 속성을 True 로 설정하면, 각 bin 에 속하는 개수를 전체 개수로 나눈 비율, 즉 정규화 한 값을 bar 의 높이로 사용하게 된다.
```




Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x110f38780>
```

## √ 산점도(Scatter plot) 그리기¶

산점도의 경우에는 서로 다른 두 개의 독립변수에 대해 두 변수가 어떤 관계가 있는지 살펴보기 위해 사용된다.

In [33]:

```
x1 = np.random.normal(1, 1, size=(100, 1))

x2 = np.random.normal(-2, 4, size=(100, 1))

X = np.concatenate((x1, x2), axis=1)

X
```

Out[33]:

```
array([[ -0.3752975 ,   -4.69632177],
       [ -0.04698916,   -4.48810875],
       [  1.74717283,   -0.34868675],
```

```
       [  0.53977735,   5.98690306],
.....................
       [  1.34541854, -10.66228373],
       [  0.83378688,  -6.00832702],
       [  1.02315838,   4.00222819],
       [  0.33959563,   4.49335211]])
```

In [34]:

```
df3 = pd.DataFrame(X, columns=["x1", "x2"])
df3
```

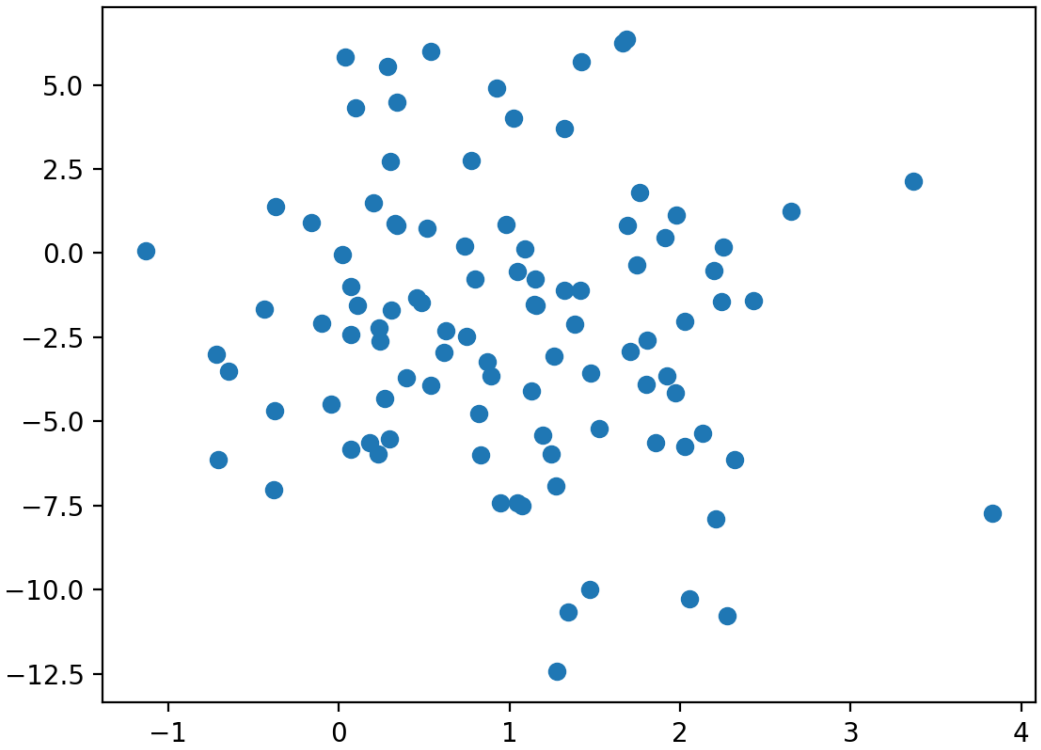
Out[34]:

| | x1 | x2 |
|---|---|---|
| **0** | -0.375297 | -4.696322 |
| **1** | -0.046989 | -4.488109 |
| **2** | 1.747173 | -0.348687 |
| **3** | 0.539777 | 5.986903 |
| **4** | 2.195603 | -0.532243 |
| **5** | 1.855601 | -5.631415 |
| | ..... | |
| **94** | 1.147161 | -1.513792 |
| **95** | 0.890595 | -3.657728 |
| **96** | 1.345419 | -10.662284 |
| **97** | 0.833787 | -6.008327 |
| **98** | 1.023158 | 4.002228 |
| **99** | 0.339596 | 4.493352 |

100 rows × 2 columns

In [36]:

```
plt.scatter(df3['x1'], df3['x2']) # x1 이 x 축, x2 가 y 축
```




Out[36]:

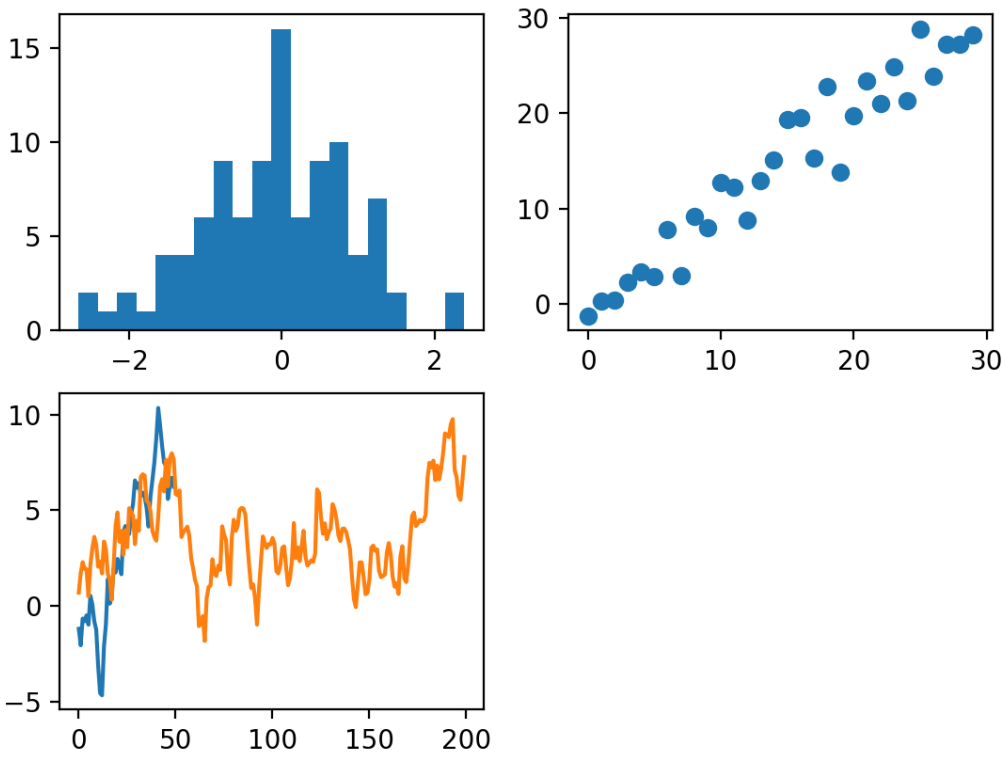
```
<matplotlib.collections.PathCollection at 0x11257b080>
```

## ▶ Plot 모양 변형하기¶

In [50]:

```
fig = plt.figure()

# 비어있는 figure 가 생성된다.
```




In [39]:

```
# subplot 추가하기, add_subplot 에는 총 3 가지 인자가 들어간다.
```

In [51]:

```
ax1 = fig.add_subplot(2, 2, 1)
```

In [52]:

```
# 첫번째 숫자와 두번째 숫자 : 우리가 figure 를 어떤 크기로 나눌지에 대한 값이다. 즉 위의 같은 경우는 2,2 이므로 우리의 figure 를 2x2 로 나눈다는 뜻.

# 세번째 숫자 : 첫번째, 두번째 숫자로 나눈 figure 에서 좌측상단으로 우측방향으로 숫자가 붙는다. 이때 우리가 add 하고자 하는 subplot 이 몇번째에 들어가는지를 나타낸다.

# 즉, 위와 같은 경우 figure 는 다음과 같이 나누어진다.

# 1  2

# 3  4

# 이때 우리는 1 위치에 subplot 을 추가하고 해당 subplot 을 ax1 이라는 변수로 반환받는다.
```

In [53]:

```
ax2 = fig.add_subplot(2,2,2)
```

In [54]:

```
ax3 = fig.add_subplot(2,2,3)
```

In [55]:

```
plt.plot(np.random.randn(50).cumsum())

# 위치를 지정하지 않고 plot 을 그리니 맨마지막에 그림이 그려진다.

# figure 에 추가된 subplot 상 맨 마지막에 위치한 곳에 그려지는 것이 아니라, 제일 마지막에 추가한 subplot 에 그려진다.

# 2 -> 3 -> 1 순으로 subplot 을 추가하여 테스트 해보면 1 번 요소에 그려진다.
```

Out[55]:

```
[<matplotlib.lines.Line2D at 0x116378160>]
```

In [56]:

```
plt.plot(np.random.randn(200).cumsum())

# 강의에서는 한번더 위치 지정 없이 그리면 그 전의 요소에 그려진다고 했는데,

# 실제로 진행해보면 그냥 위의 것과 똑같이 제일 마지막에 추가한 subplot 에 중복되서 그려진다.
```

Out[56]:

```
[<matplotlib.lines.Line2D at 0x116384278>]
```

In [57]:

```
# 그럼 우리가 원하는 위치에 그림을 그리기 위해서는?

# 위에서 add_subplot 을 할때 변수명을 지정하여 반환값을 받았다.

# 해당 변수를 통해 plot 을 그리면 된다.

ax1.hist(np.random.randn(100), bins = 20) # bins 는 x 축 bar 의 개수
```

Out[57]:

```
(array([ 2.,  1.,  2.,  1.,  4.,  4.,  6.,  9.,  6.,  9., 16.,  6.,  9.,
        10.,  4.,  7.,  2.,  0.,  0.,  2.]),
 array([-2.65952998, -2.40690555, -2.15428112, -1.9016567 , -1.64903227,
        -1.39640784, -1.14378341, -0.89115898, -0.63853456, -0.38591013,
        -0.1332857 ,  0.11933873,  0.37196316,  0.62458759,  0.87721201,
```

```
        1.12983644,  1.38246087,  1.6350853 ,  1.88770973,  2.14033415,
        2.39295858]),
 <a list of 20 Patch objects>)
```

In [58]:

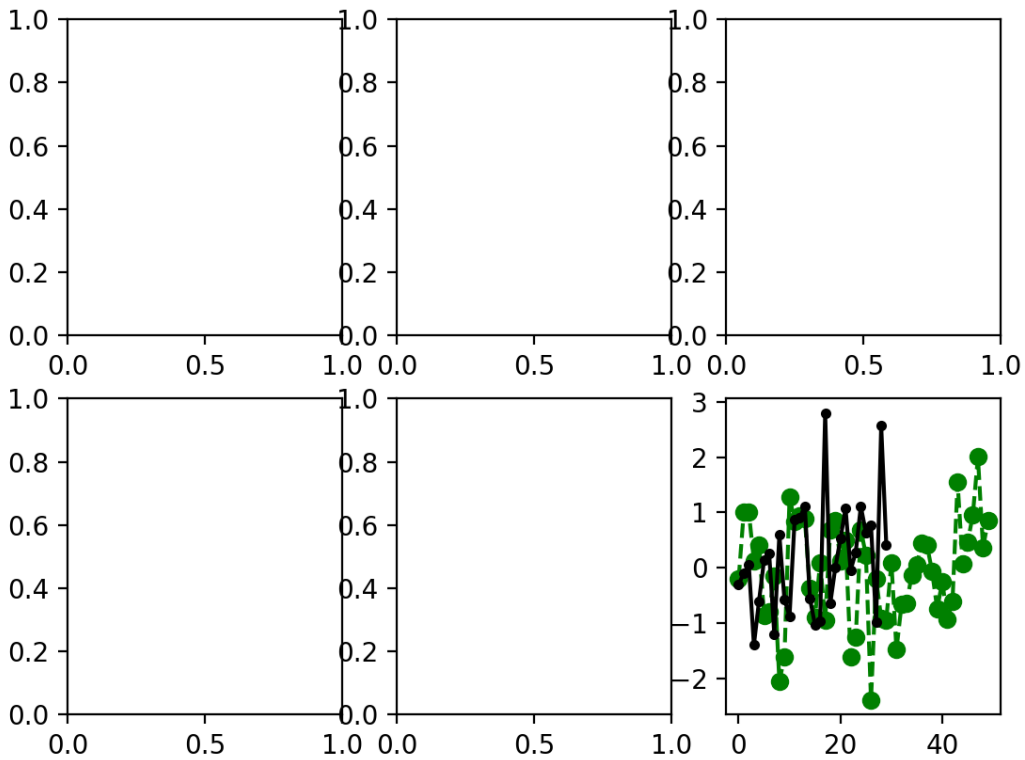
```
ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

Out[58]:

```
<matplotlib.collections.PathCollection at 0x1162d7c18>
```

In [59]:

```
# 또 다른 방법
fig, axes = plt.subplots(2,3)
# 위와 같이 만들면 2x3 subplot 들을 가지는 figure 를 만드는 것
# 이때 반환되는 값은 2 개로써, figrue 자체와, 축
```



In [49]:

```
# 반환받은 axes 에는 우리가 위에서 설정한 크기와 같은 shape 의 리스트로 각 요소에는 subplot 객체가 들어있다.
axes
```

Out[49]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x115212160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x115231748>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x115255cf8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1152873c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1152afa90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1152e2278>]],
      dtype=object)
```

### √ Plot 꾸미기¶

In [60]:

```
plt.plot(np.random.randn(50), color = 'g', marker='o', linestyle='--')
```

Out[60]:

```
[<matplotlib.lines.Line2D at 0x11654deb8>]
```

# color¶

값 색상

"b" blue

"g" green

"r" red

"c" cyan

"m" magenta

"y" yellow

"k" black

"w" white

# marker¶

값 마킹

"." point

"," pixel

"o" circle

"v" triangle_down

"^" triangle_up

"<" triangle_left

">" triangle_right

"8" octagon

"s" square

"p" pentagon

"*" star

"h" hexagon

"+" plus

"x" x

"D" diamond

# line style¶

값 라인 스타일

"-" solid line

"--" dashed line

"-." dash-dotted line

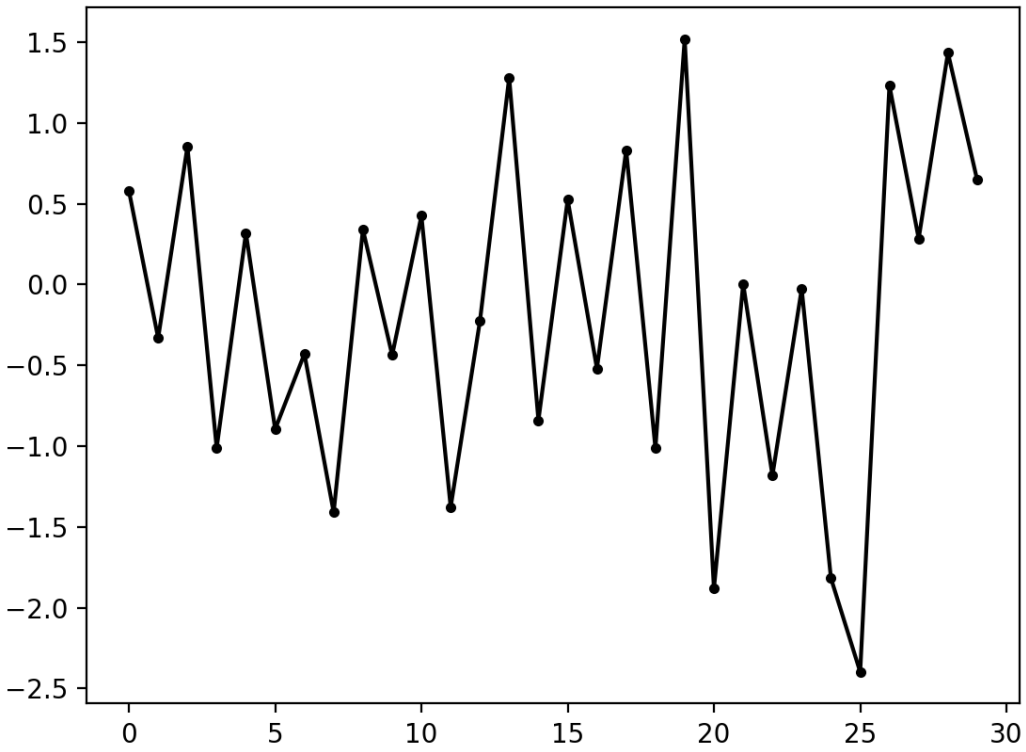
":" dotted line

"None" draw nothing

In [63]:

```
# 굳이 각각에 대해서 언급해주지 않고, 아래와 같이 연속적으로 나타내줘도 된다.

plt.plot(np.random.randn(30), 'k.-')
```

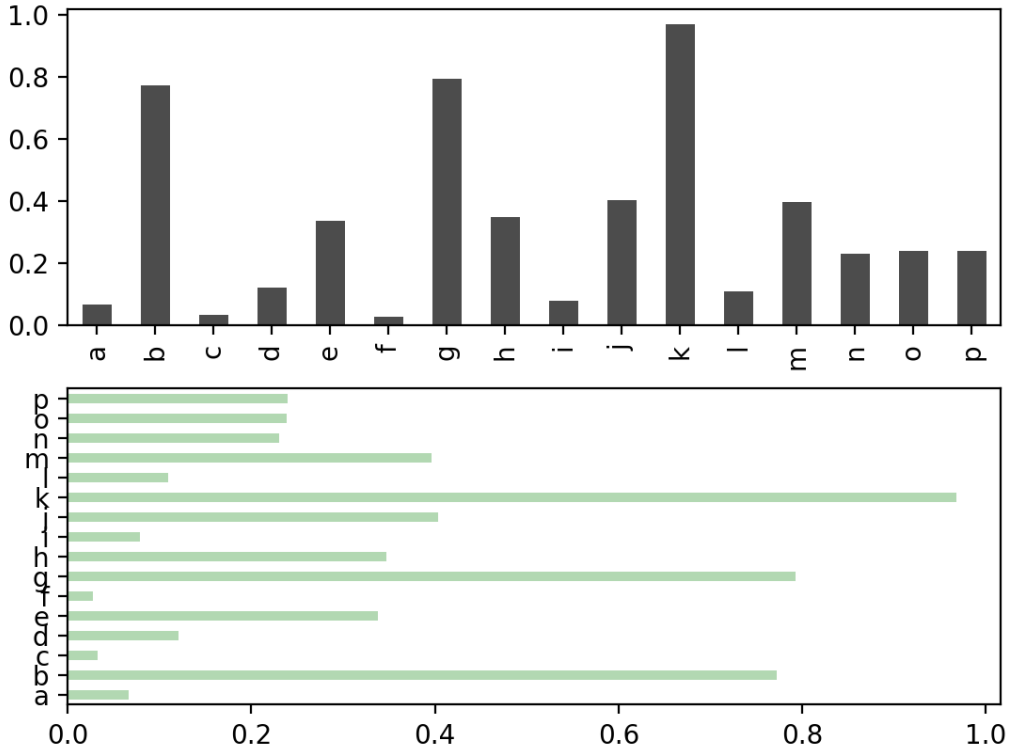



Out[63]:

```
[<matplotlib.lines.Line2D at 0x116aaecf8>]
```

In [69]:

```
fig, axes = plt.subplots(2,1)
```




In [70]:

```
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))
```

In [71]:

```
data.plot(kind='bar', ax=axes[0], color='k', alpha=0.7) #plot 함수를 그릴때, figure 에서 원하는 위치를 지정하기 위해 ax 속성을 사용
```

Out[71]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x117ab1c50>
```

In [72]:

```
data.plot(kind='barh', ax=axes[1], color='g', alpha=0.3)
```

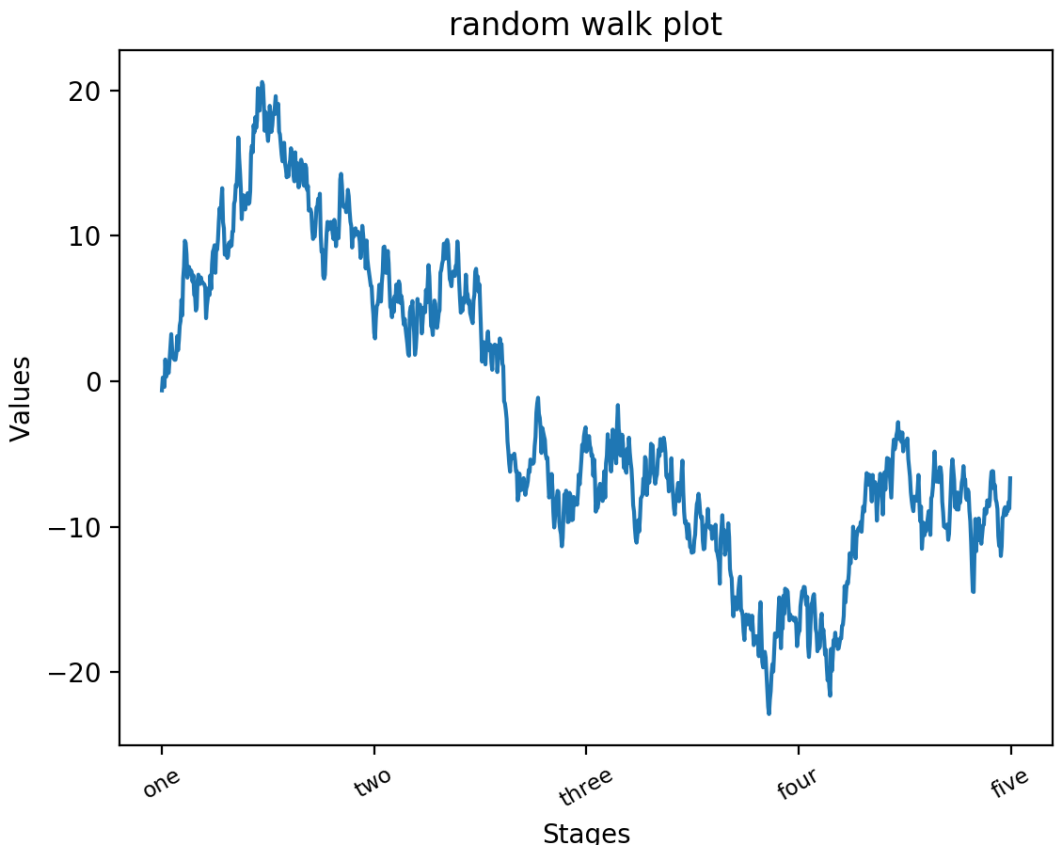
Out[72]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1184960b8>
```

**우리가 만들 plot 의 눈금, 레이블, 범위 등을 지정 및 수정할 수 있다.**

In [73]:

```
fig = plt.figure()
```




In [74]:

```
ax = fig.add_subplot(1,1,1)
```

In [75]:

```
ax.plot(np.random.randn(1000).cumsum())
```

Out[75]:

```
[<matplotlib.lines.Line2D at 0x119940320>]
```

In [76]:

```
# 이때 그래프에서 나타내는 눈금을 tick 이라고 한다.

# 즉, 위의 그래프의 x tick 은 200 이고 y tick 은 10 이다.
```

In [77]:

```
ax.set_xticks([0, 250, 500, 750, 1000])
```

Out[77]:

```
[<matplotlib.axis.XTick at 0x11940ac88>,

 <matplotlib.axis.XTick at 0x11940a5c0>,
```

```
 <matplotlib.axis.XTick at 0x117ab18d0>,
 <matplotlib.axis.XTick at 0x119926a90>,
 <matplotlib.axis.XTick at 0x1199321d0>]
```

In [78]:

```
# 눈금을 문자로 하기 위해서는?
labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'], rotation = 30, fontsize='small')
```

In [79]:

```
# 제목 입력하기
ax.set_title('random walk plot')
```

Out[79]:

```
Text(0.5,1,'random walk plot')
```

In [80]:

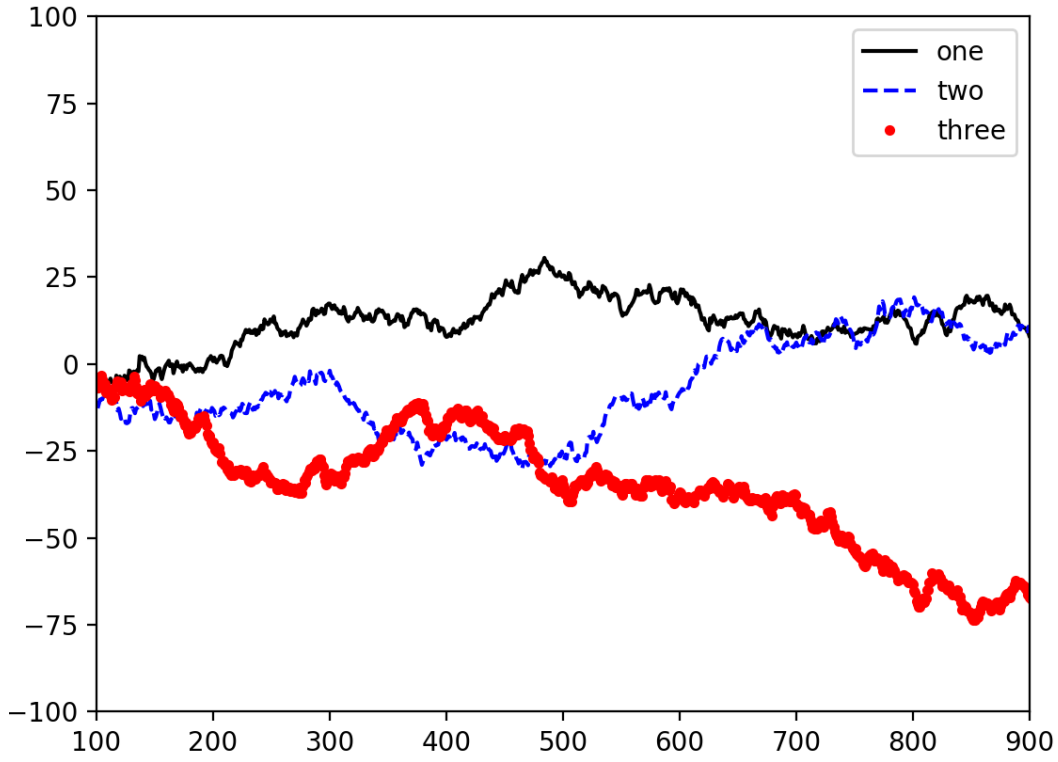
```
# 라벨 입력하기
ax.set_xlabel('Stages')
ax.set_ylabel('Values')
```

Out[80]:

```
Text(70.8194,0.5,'Values')
```

In [81]:

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
```




In [82]:

```
ax.plot(np.random.randn(1000).cumsum(), 'k', label='one')

ax.plot(np.random.randn(1000).cumsum(), 'b--', label='two')

ax.plot(np.random.randn(1000).cumsum(), 'r.', label='three')
```

Out[82]:

```
[<matplotlib.lines.Line2D at 0x119939da0>]
```

In [83]:

```
# 범례 표시하기

ax.legend(loc='best')

# loc 는 범례가 위치할 곳을 의미한다. best 를 주게 되면 현재 그래프에서 최적의 위치를 자동으로 찾는다.
```

Out[83]:

```
<matplotlib.legend.Legend at 0x119f4f550>
```

## √ x 축 범위 및 y 축 범위 수정하기

In [84]:

```
ax.get_xlim()

# 현재 그래프의 x 축 범위를 가져온다.
```

Out[84]:

```
(-49.95, 1048.95)
```

In [85]:

```
# 이를 변경하려면,

ax.set_xlim([100,900])
```

Out[85]:

```
(100, 900)
```

In [86]:

```
ax.set_ylim([-100,100])
```

Out[86]:

```
(-100, 100)
```

# 파이썬 라이브러리 응용 예제

## (1) 넘파이 응용 (고성능)

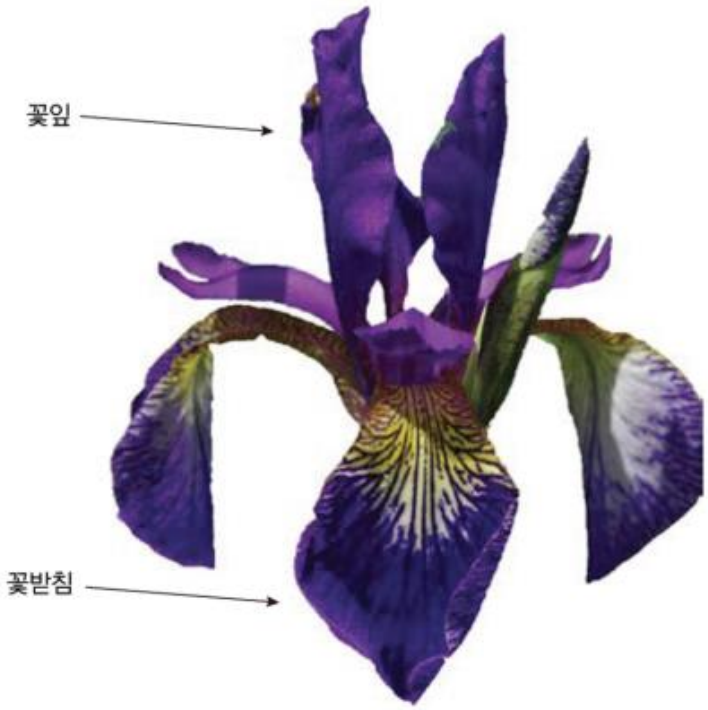
```
1  ## 영상 처리 및 데이터 분석 툴
2  from tkinter import *; import os.path ;import math
3  from   tkinter.filedialog import *
4  from   tkinter.simpledialog import *
5  import numpy as np
6
7  ## 함수 선언부
8
9  def loadImage(fname) :
10     global window, canvas, paper, filename, inImage, outImage, inW, inH, outW, outH
11     fsize = os.path.getsize(fname) # 파일 크기 확인
12     inH = inW = int(math.sqrt(fsize))  # 입력메모리 크기 결정! (중요)
13
14     # 파일 --> 메모리로 데이터 로딩
15     inImage = np.fromfile(fname, dtype='uint8') # 파일에서 바로 읽기 (1 차원) --> 성능이
   매우 우수함~~~~
16     inImage = inImage.reshape(inH, inW)  # 2 차원으로 Reshape
17
18 def openFile() :
19     global window, canvas, paper, filename,inImage, outImage,inW, inH, outW, outH
20     filename = askopenfilename(parent=window,
21                                filetypes=(("RAW 파일", "*.raw"), ("모든파일", "*.*")))
22     loadImage(filename) # 파일 --> 입력메모리
23     equal() # 입력메모리--> 출력메모리
24
25 import threading
26 def display() :
27     global window, canvas, paper, filename, inImage, outImage, inW, inH, outW, outH, VIEW_X,
   VIEW_Y
28     # 기존에 캐버스 있으면 뜯어내기.
29     if   canvas != None :
30          canvas.destroy()
```

```
31      # 화면 준비 (고정됨)
32      VIEW_X, VIEW_Y = 512, 512
33      if VIEW_X >= outW or VIEW_Y >= outH : # 영상이 256 미만이면
34          VIEW_X = outW
35          VIEW_Y = outH
36          step = 1   # 건너뛸숫자
37      else :
38          step = outW / VIEW_X # step 을 실수도 인정. 128, 256, 512 단위가 아닌 것 고려.
39
40      window.geometry(str(int(VIEW_X*1.2)) + 'x' + str(int(VIEW_Y*1.2)))
41      canvas = Canvas(window, width=VIEW_X, height=VIEW_Y)
42      paper = PhotoImage(width=VIEW_X, height=VIEW_Y)
43      canvas.create_image((VIEW_X/2, VIEW_X/2), image=paper, state='normal')
44
45      # 화면에 출력. 실수 step 을 위해서 numpy 사용
46      rgbString = ''   # 여기에 전체 픽셀 문자열을 저장할 계획
47      for i in np.arange(0, outH, step):
48          tmpString = ''
49          for k in np.arange(0, outW, step):
50              i = int(i);
51              k = int(k)   # 첨자이므로 정수화
52              r = g = b = int(outImage[i][k]);
53              tmpString += ' #%02x%02x%02x' % (r, g, b)
54          rgbString += '{' + tmpString + '} '
55      paper.put(rgbString)
56
57      canvas.pack(expand=1, anchor=CENTER)
58      status.configure(text='이미지 정보:' + str(outW) + 'x' + str(outH))
59
60  def equal() :   # 동일 영상 알고리즘
61      global window, canvas, paper, filename, inImage, outImage, inW, inH, outW, outH
62      # 중요! 출력메모리의 크기를 결정
63      outW = inW;   outH = inH;
64
65      ############################
66      # 진짜 영상처리 알고리즘을 구현
67      ############################
68      outImage = inImage[:]
```

```
69
70      display()
71
72  import time
73  def addImage() :   # 밝게하기 알고리즘
74      global window, canvas, paper, filename, inImage, outImage, inW, inH, outW, outH
75      # 중요! 출력메모리의 크기를 결정
76      outW = inW;   outH = inH;
77
78      ##############################
79      # 진짜 영상처리 알고리즘을 구현
80      ##############################
81      value = askinteger('밝게/어둡게하기', '밝게/어둡게 할 값-->', minvalue=-255,
    maxvalue=255)
82      startTime = time.time()
83      inImage = inImage.astype(np.int) # 범위를 넘을 수 있으므로, 형변환 필요
84      outImage = inImage + value
85
86      ## 조건으로 0~255 범위로 수렴
87      outImage = np.where(outImage > 255, 255, outImage)
88      outImage = np.where(outImage < 0, 0, outImage)
89
90      seconds = time.time() - startTime # 초단위
91      display()
92      status.configure(text=status.cget("text") + '₩t₩t 걸린시간(초):' + "{0:.2f}".format(seconds))
93
94  ## 변수 선언 부분 ##
95  window = None
96  canvas = None
97  XSIZE, YSIZE = 256, 256
98
99  ## 메인 코드 부분 ##
100 window = Tk()
101 window.title('CJ 파이썬 라이브러리 (Numpy)')
102 window.geometry('500x500')
103 canvas = Canvas(window, height = XSIZE, width = YSIZE)
104
105 status = Label(window, text='이미지 정보:', bd=1, relief=SUNKEN, anchor=W)
```

```
106 status.pack(side=BOTTOM, fill=X)
107
108 mainMenu = Menu(window);window.config(menu=mainMenu)
109 fileMenu = Menu(mainMenu);mainMenu.add_cascade(label='파일', menu=fileMenu)
110 fileMenu.add_command(label='열기', command=openFile)
111 fileMenu.add_command(label='저장', command=None)
112 fileMenu.add_separator()
113 fileMenu.add_command(label='종료', command=None)
114
115 pixelMenu = Menu(mainMenu);mainMenu.add_cascade(label='화소점처리', menu=pixelMenu)
116 pixelMenu.add_command(label='동일영상', command=equal)
117 pixelMenu.add_command(label='밝게/어둡게', command=addImage)
118
119 canvas.pack()
120 window.mainloop()
121 window.mainloop()
```

## (2) 판다스 응용 (머신러닝)




한 아마추어 식물학자가 들에서 발견한 붓꽃의 품종을 알고 싶다고 가정하겠습니다. 이 식물

학자는 붓꽃의 꽃잎petal과 꽃받침sepal의 폭과 길이를 센티미터 단위로 측정하였습니다.

또 전문 식물학자가 setosa, versicolor, virginica 종으로 분류한 붓꽃의 측정 데이터도 가지고 있습니다. 이 측정값을 이용해서 앞에서 채집한 붓꽃이 어떤 품종인지 구분하려고 합니다. 이 아마추어 식물학자가 야생에서 채집한 붓꽃은 이 세 종류뿐이라고 가정하겠습니다.

우리의 목표는 어떤 품종인지 구분해 놓은 측정 데이터를 이용해 새로 채집한 붓꽃의 품종을 예측하는 머신러닝 모델을 만드는 것입니다.

iris.csv 파일 (150개 조사) → 일부 예

| sepal.length | sepal.width | petal.length | petal.width | variety |
|---|---|---|---|---|
| 5.1 | 3.5 | 1.4 | 0.2 | Setosa |
| 4.9 | 3 | 1.4 | 0.2 | Setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | Setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | Setosa |
| 5.2 | 2.7 | 3.9 | 1.4 | Versicolor |
| 5 | 2 | 3.5 | 1 | Versicolor |
| 5.9 | 3 | 4.2 | 1.5 | Versicolor |
| 6 | 2.2 | 4 | 1 | Versicolor |
| 6.1 | 2.9 | 4.7 | 1.4 | Versicolor |
| 6.2 | 2.8 | 4.8 | 1.8 | Virginica |
| 6.1 | 3 | 4.9 | 1.8 | Virginica |
| 6.4 | 2.8 | 5.6 | 2.1 | Virginica |
| 7.2 | 3 | 5.8 | 1.6 | Virginica |
| 6.2 | 3.4 | 5.4 | 2.3 | Virginica |
| 5.9 | 3 | 5.1 | 1.8 | Virginica |

▶ 실습 : scikit-learn 기초 연습

```
1  from sklearn import svm, metrics
2
3  #1. Classifire 생성(선택) --> 머신러닝 알고리즘 선택
4  clf = svm.SVC(gamma='auto')
5
6  #2. 데이터로 학습 시키기
7  #clf.fit( [ 훈련데이터] , [정답] )
8  clf.fit (   [ [0,0],
```

```
                    [0,1],
                    [1,0],
                    [1,1]] ,
                  [ 0, 1, 1, 0])
#3. 정답률을 확인 (신뢰도)
results = clf.predict([[1,0],[0,0]])

score = metrics.accuracy_score(results, [1,0])
print("정답률 :", score*100, '%')

#4. 예측하기
# clf.predict( [예측할 데이터] )
result = clf.predict( [ [1,0] ])
print(result)
```

▶ 실습 : scikit-learn 기본 - 붓꽃

```
from sklearn import svm, metrics
import pandas as pd
'''
# 붓꽃 데이터 분류기 (머신러닝)
- 개요 : 150 개 붓꽃 정보(꽃받침 길이, 꽃받침 폭, 꽃잎 길이, 꽃잎 폭)
- 종류 : 3 개 (Iris-setosa, Iris-vesicolor, Iris-virginica)
-
'''
##0. 훈련데이터, 테스트데이터 준비
csv = pd.read_csv('data/iris.csv')
train_data = csv.iloc[0:120, 0:-1]
train_label = csv.iloc[0:120, [-1]]

test_data = csv.iloc[120:, 0:-1]
test_label = csv.iloc[120:, [-1]]


#1. Classifire 생성(선택) --> 머신러닝 알고리즘 선택
clf = svm.SVC(gamma='auto')

#2. 데이터로 학습 시키기
```

```
#clf.fit( [ 훈련데이터] , [정답] )
#clf.fit (   train_data ,train_label)
clf.fit (   train_data ,train_label.values.ravel())

# 3. 정답률 구하기
result = clf.predict(test_data)
score = metrics.accuracy_score(result, test_label)
print('정답률 : ', "{0:.2f}%".format(score * 100))

# 4. 내꺼 예측하기
#clf.predict( [예측할 데이터] )
result = clf.predict( [[4.1,3.3,1.5,0.2] ])
print(result)

```

▶ 실습 : scikit-learn 기본 3 - 학습/테스트 데이터 자동 분리

```
from sklearn import svm, metrics
import pandas as pd
from sklearn.model_selection import train_test_split

##0. 훈련데이터, 테스트데이터 준비
csv = pd.read_csv('c:/bigdata/iris.csv')
data = csv.iloc[:, 0:-1]
label = csv.iloc[:, [-1]]
## 학습용, 훈련용 분리
train_data, test_data, train_label, test_label = ₩
      train_test_split(data, label, train_size=0.3)

#1. Classifire 생성(선택) --> 머신러닝 알고리즘 선택
clf = svm.SVC(gamma='auto')

#2. 데이터로 학습 시키기
#clf.fit( [ 훈련데이터] , [정답] )
#clf.fit (   train_data ,train_label)
clf.fit (   train_data ,train_label.values.ravel())

# 3. 정답률 구하기
```

```
result = clf.predict(test_data)
score = metrics.accuracy_score(result, test_label)
print('정답률 : ', "{0:.2f}%".format(score * 100))

# 4. 내꺼 예측하기
#clf.predict( [예측할 데이터] )
result = clf.predict( [[4.1,3.3,1.5,0.2] ])
print(result)
```