ESTSOFT WASSUP

AI 서비스 기획

맹광국 강사

ggmaeng@gmail.com

EST

데이터 전처리

Numpy

Pandas

OpenCV

Numpy

배열의 특성과 생성

다차원 배열

배열 활용

Numpy

NumPy("넘파이"라 읽는다)는 행렬이나 일반적으로 대규모 다차원 배열을 쉽게 처리할 수 있도록 지원하는 파이썬의 라이브러리이다. NumPy는 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공한다.

예제 [편집]

배열 생성

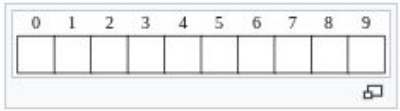
```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x
array([1, 2, 3])
>>> y = np.arange(10) # like Python's range, but returns an array
>>> y
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- Numeric + Python = Numpy
- 수학 및 과학 연산을 위한 파이썬 패키지
- 배열이나 행렬 계산에 용이한 메서드를 제공
- 한글로 넘파이로 주로 통칭, 넘피/늄파이라고 부르기도 함
- http://www.numpy.org

배열(array)

EST

다른 뜻에 대해서는 배열 (동음이의) 문서를 참고하십시오.

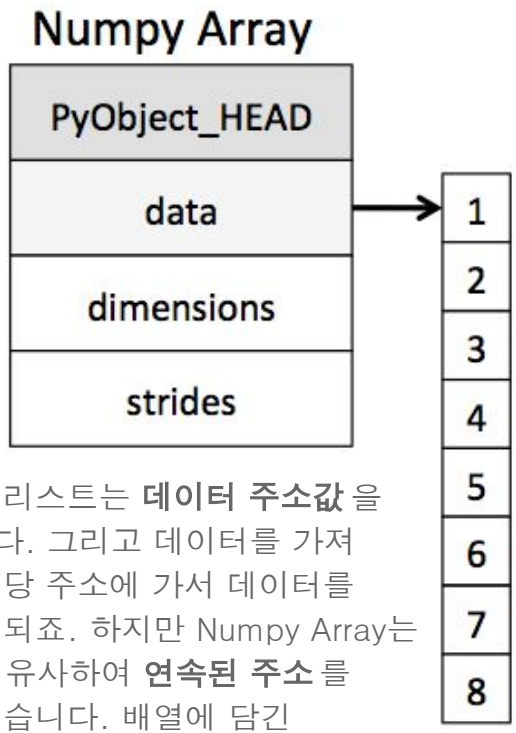


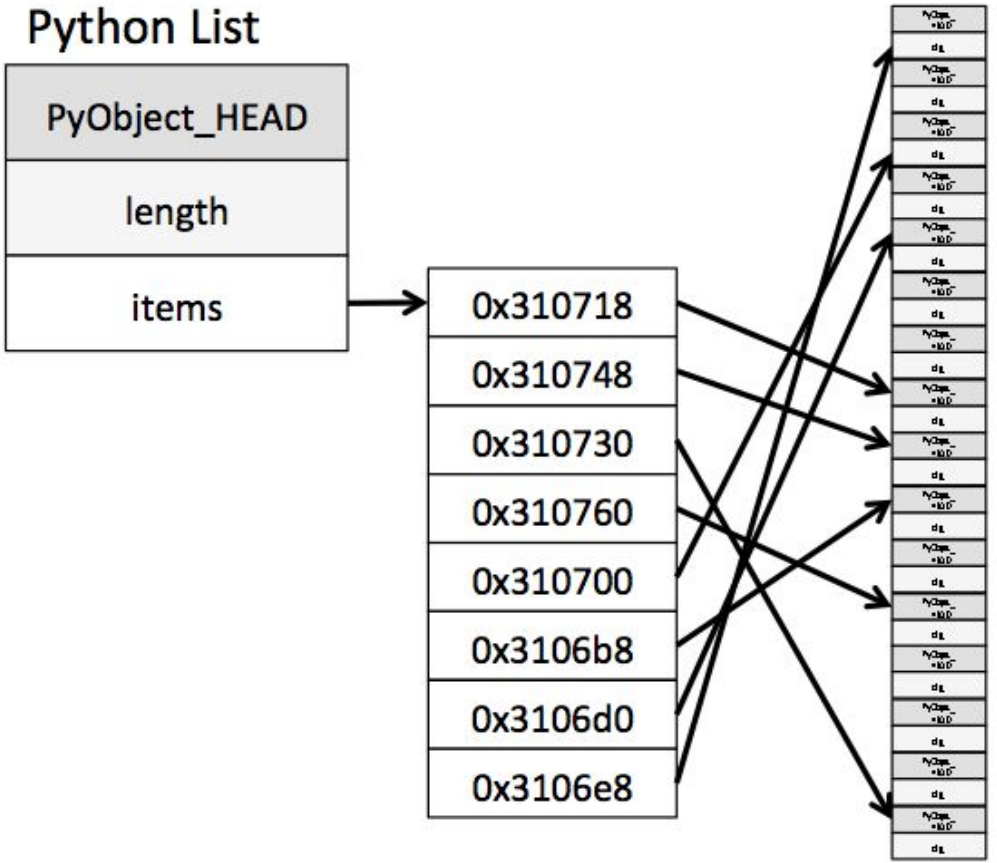
컴퓨터 과학에서 **배열**(영어: array, 配列·排列, 문화어: 배렬)은 번호(인덱스)와 번호에 대응하는 데이터들로 이루어진 자료 구조를 나타낸다. 일반적으로 배열에는 같은 종류의 데이터들이 순차적으로 저장되어, 값의 번호가 곧 배열의 시작점으로부터 값이 저장되어 있는 상대적인 위치가 된다. 대부분의 프로그래밍 언어에서 사용할 수 있는 가장 기초적인 자료 구조로, 기본적인 용도 외에 다른 복잡한 자료 구조들을 표현하기 위해서 또는 행렬, 벡터 등을 컴퓨터에서 표현하는 용도 등으로도 사용된다.

배열의 첫 번째 요소의 메모리 주소를 첫 번째 주소, 기본 주소 또는 기본 주소라고 한다.

- 리스트와 외부적으로는 유사해 보이지만 내부적으로 많은 차이 존재
- 일반 List에 비해 빠르고, 메모리 효율적
- 메모리 영역을 미리 계산하여 공간을 할당
- 반복문 없이 데이터 배열에 대한 처리를 지원함
- 선형대수와 관련된 다양한 기능을 제공함
- 항목의 타입을 미리 지정해야 생성가능
- 항목의 모형(개수) 가 정해져야 생성 가능
- 한번 정해진 모형(개수)는 변경불가 but 값은 변경 가능
    - append()는 arr에 직접 값을 추가하지 않고, 복사된 배열에 값을 추가하고 리턴합니다.

np.array의 구조(list와의 차이점)

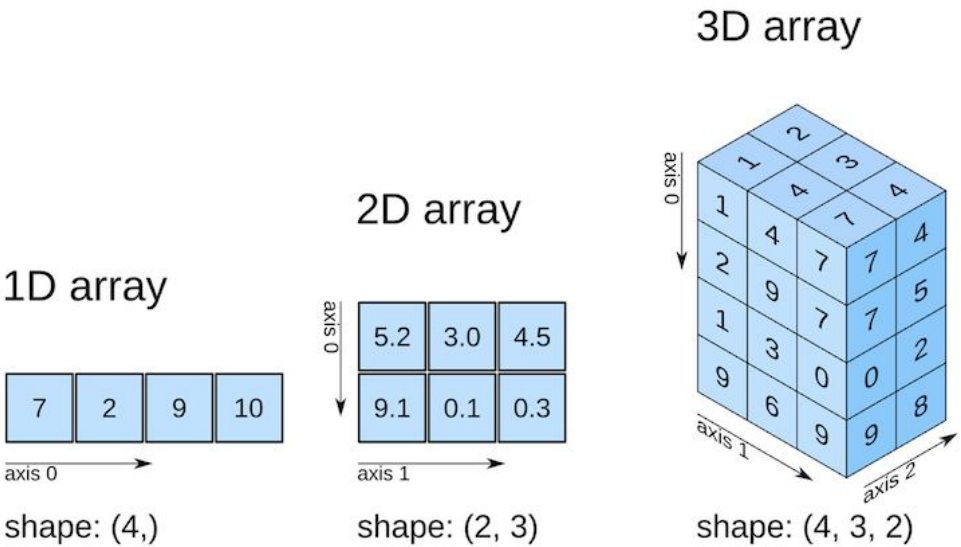


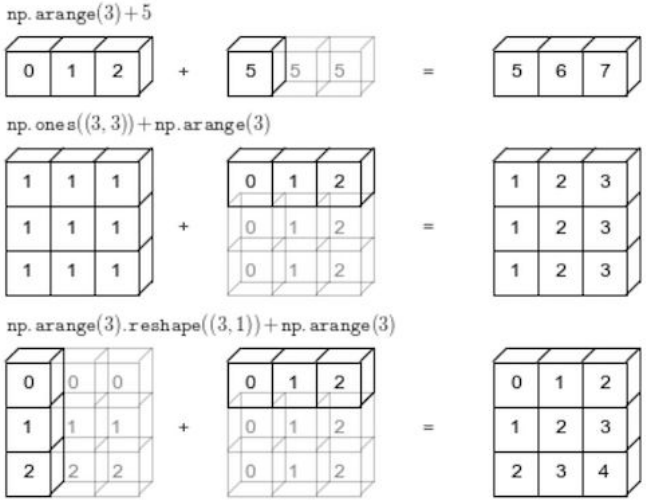


파이썬의 리스트는 **데이터 주소값**을 저장합니다. 그리고 데이터를 가져올때는 해당 주소에 가서 데이터를 가져오게 되죠. 하지만 Numpy Array는 C 배열과 유사하여 **연속된 주소**를 가지고 있습니다. 배열에 담긴 데이터를 가져온다면 순서대로 가져오면 되기 때문에 **메모리를 효율적**으로 사용합니다.

reference

다차원 배열

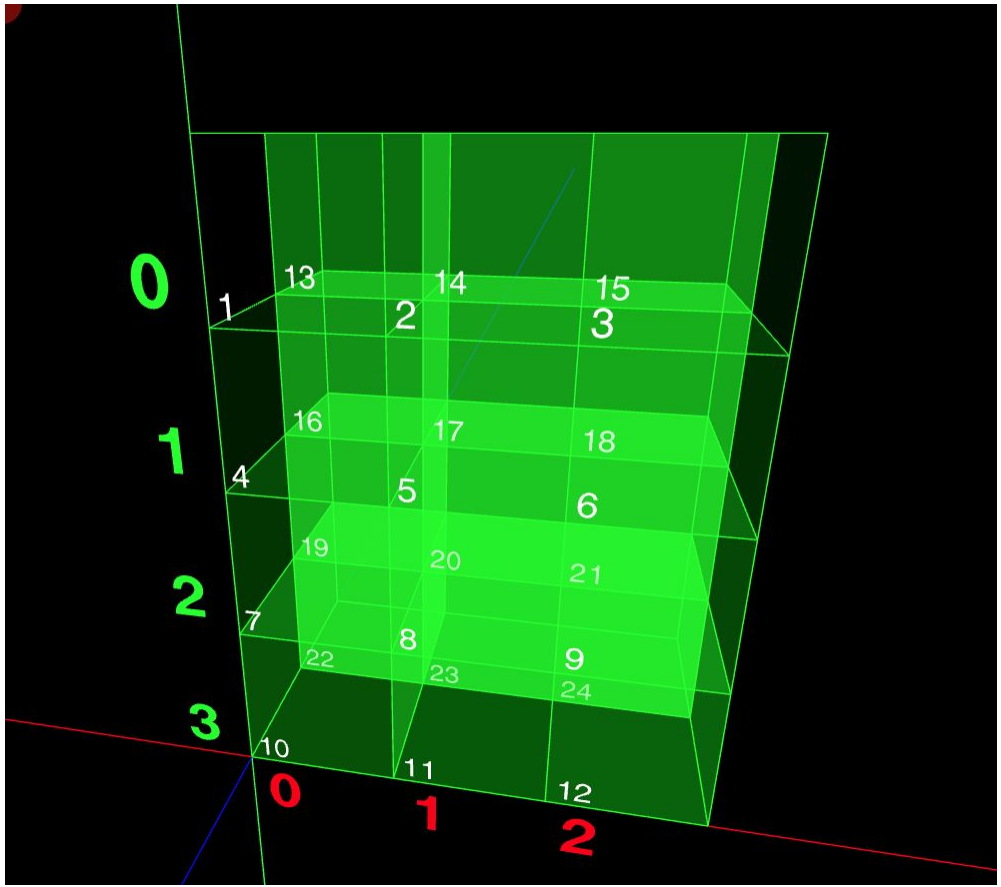

1D array
7
2
9
10
axis 0
shape: (4,)
2D array
axis 0
5.2
3.0
4.5
9.1
0.1
0.3
axis 1
shape: (2, 3)
3D array
axis 0
axis 1
axis 2
shape: (4, 3, 2)


np.arange(3)+5
np.ones((3,3))+np.arange(3)
np.arange(3).reshape((3,1))+np.arange(3)

Array Visualization

EST

https://array-3d-viz.vercel.app/



배열의 생성

- 구현된 이터러블을 활용해 asarray()로 생성: np.asarray(iterable)
- 지정 모형의 0으로 초기화 시키기: np.zeros((3,2))
- 지정 모형의 1으로 초기화 시키기: np.ones((3,2))
- 지정 모형의 0~1 랜덤 초기화 시키기: np.random.rand(3,2)

```
>>> import numpy as np
```

```
>>> a = np.array(0)
>>> a
array(0)
>>> print(a)
0
>>> type(a)
<class 'numpy.ndarray'>
```

```
>>> a = np.array([2, 4, 6])
>>> a
array([2, 4, 6])
>>> print(a)
[2 4 6]
>>> type(a)
<class 'numpy.ndarray'>
```

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> print(a)
[[1 2 3]
 [4 5 6]]
>>> type(a)
<class 'numpy.ndarray'>
```

배열의 모형

EST

- shape: 배열의 모형을 출력
- reshape(): 배열의 모형을 변형(절대모형유지)
- transpose(): 배열의 축을 반대로(행렬 전치)
- expand_dims(): 배열의 차원을 확장
- squeeze(): 배열의 차원을 축소
- concatenate(): 두개이상의 배열을 합침
- flatten(): 다차원 배열을 1차원으로
- split(): 배열 분할

```
a = np.asarray([[1,2,3],[4,5,6]])

a.shape
=> (2,3)
a.reshape((3,2))
=> [[1 2] [3 4] [5 6]]
```

```
a.shape
=> (2,3)

np.transpose(a)
=>[[1 4] [2 5] [3 6]]

e = np.expand_dims(a,axis=0)
e.shape
=> (1,2,3)
np.squeeze(e).shape
=>(2,3)

b.shape
=> (2,4)
np.concatenate((a,b),axis=1).shape
=> (2,7)
```

배열의 모형

ndim (dimention, rank)

```
>>> import numpy as np
```

```
>>> a = np.array(0)
>>> a.ndim
0
```

```
>>> a = np.array([2, 4, 6])
>>> a.ndim
1
```

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

```
>>> a = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
>>> a.ndim
3
```

shape

```
>>> import numpy as np
```

```
>>> a = np.array(3)
>>> print(a)
3
>>> a.shape
()
```

```
>>> a = np.array([1, 3, 5, 7, 9])
>>> print(a)
[1 3 5 7 9]
>>> a.shape
(5,)
```

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> print(a)
[[1 2 3]
 [4 5 6]]
>>> a.shape
(2, 3)
```

```
>>> a = np.array([[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]], [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]])
>>> print(a)
[[[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]

 [[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]]
>>> a.shape
(2, 3, 4)
```

배열의 모형

EST

Reshaping

```
>>> import numpy as np
```

```
>>> a = np.arange(12)
>>> print(a)
[ 0  1  2  3  4  5  6  7  8  9 10 11]
>>> a.shape
(12,)
```

```
>>> a.shape = 2, -1
>>> print(a)
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
>>> a.shape
(2, 6)
```

```
>>> a.shape = 3, -1
>>> print(a)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
>>> a.shape
(3, 4)
```

```
>>> a.shape = -1, 1
>>> print(a)
[[ 0]
 [ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]
 [11]]
>>> a.shape
(12, 1)
```

```
>>> a.shape = -1, 2
>>> print(a)
[[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]]
>>> a.shape
(6, 2)
```

```
>>> a.shape = -1, 3
>>> print(a)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>> a.shape
(4, 3)
```

```
>>> a.shape = -1
>>> print(a)
[ 0  1  2  3  4  5  6  7  8  9 10 11]
>>> a.shape
(12,)
```

size

```
>>> import numpy as np
```

```
>>> a = np.array([1, 3, 5, 7, 9])
>>> a.size
5
```

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

배열의 활용

EST

- 다양한 연산 함수 제공
  - len(): 배열의 개수 반환
  - max(), min(): 최댓값, 최솟값
  - mean(), median() : 평균, 중앙값
  - std(), var() : 표준편차, 분산
  - sum(), exp(), log(), power(): 각종 수학연산
  - sort(), argsort(): 정렬, 정렬 인덱스
  - argmax(), argmin(): 최대,최소 인덱스
  - where(): 쿼리 연산
  - astype(): 타입 변환

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

np.sum(amat, axis = 0)

=> [15 18 21 24]

np.sum(amat, axis = 1)
=> [10 26 42]
```

- 주의사항
  - 다차원일 경우 함수사용시 axis(축)을 지정해 줄것

Pandas

배열의 특성과 생성

다차원 배열

배열 활용

Pandas

EST

대부분의 데이터는 시계열(series)이나 표(table)의 형태로 나타낼 수 있다. 판다스(Pandas) 패키지는 이러한 데이터를 다루기 위한 시리즈(Series) 클래스와 데이터프레임(DataFrame) 클래스를 제공한다.

pandas는 데이터 조작 및 분석을 위한 파이썬 프로그래밍 언어 용으로 작성된 소프트웨어 라이브러리이다. 숫자 테이블과 시계열을 조작하기 위한 데이터 구조와 연산을 제공하며, 무료 소프트웨어 New BSD 라이센스이다. pandas란 이름은 한 개인에 대해 여러 기간동안 관찰을 한다는 데이터 세트에 대한 계량 경제학 용어인 "패널 데이터"라는 용어에서 파생되었다. 또한 "Python 데이터 분석"이라는 문구 자체에서 따온 것이기도 하다. Wes McKinney는 2007년부터 2010년까지 연구원으로 있을 때 AQR Capital에서 pandas를 만들기 시작했다.

https://pandas.pydata.org/

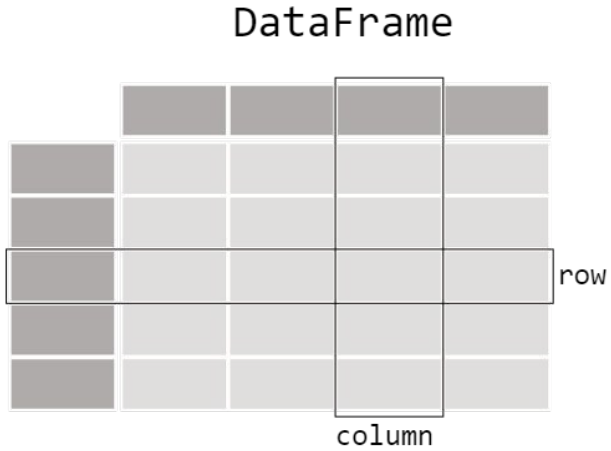
```
import pandas as pd
```

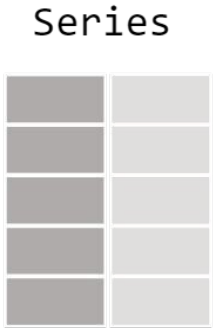
데이터 프레임 [편집]

Pandas는 주로 데이터 분석에 사용된다. Pandas를 사용하면 쉼표로 구분된 값, JSON, SQL 및 Microsoft Excel 과 같은 다양한 파일 형식에서 데이터를 가져올 수 있다. 병합 등의 각종 데이터 처리 동작을 허용, 재편, 선택 뿐만 아니라 청소 데이터 및 데이터 승강이 가능하다.

DataFrame

- DataFrame
  - 테이블형 구조로 이루어 져 있음
  - 데이터를 정형화 시킬수 있음
  - 특정 규칙을 따르는 대량의 데이터를 한눈에 파악 할 수 있음
  - 원하는 데이터를 쉽게 찾아올수 있음
  - 생성하기 위해선 테이블 형을 가질수 있는 데이터구조를 사전 구축해야함



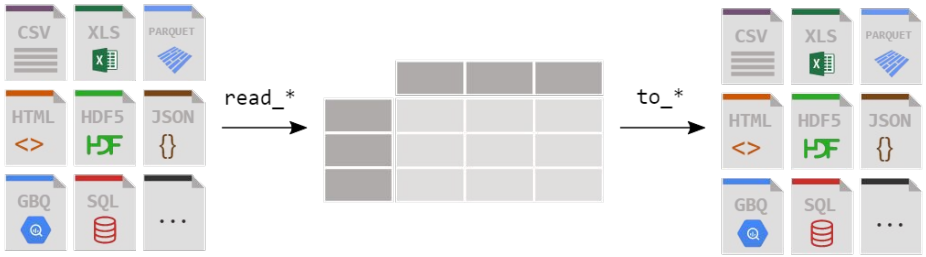


- Series
  - DataFrame의 단일열 또는 행
  - 1차원 구조를 가지고 있고 각 데이터의 인덱스를 가지고있음
  - DataFrame의 열 또는 행이름이 Series의 name으로 지정

Read & Write

- 표(테이블) 형식의 데이터 읽기
  - read_csv(), read_excel() 등의 함수를 통해 테이블형 데이터 파일 읽기 가능

```
netflix =
pd.read_csv('data/netflix.csv')
```



- 표(테이블) 형식 데이터로 쓰기
  - to_csv(), to_excel() 등의 함수를 통해 테이블형 데이터파일로 쓰기 가능

```
netflix.to_excel('netflix.xlsx',
sheet_name='movie1', index=False)
```

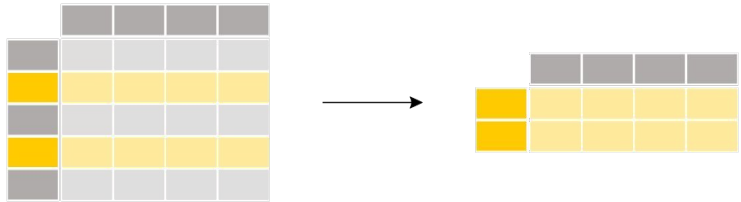
DataFrame 다루기

- 열(columns) 이름으로 하위 집합 생성
  - []내에 원하는 컬럼 이름 입력, 여러개의 열을 가져올시 리스트로

```
names = netflix['Name']
name_year = netflix[['Name', 'Year']]
```

- 행(row) 이름으로 하위 집합 생성
  - .loc[]내에 원하는 행의 이름을 입력, 여러개의 행을 가져올시 리스트로
  - .iloc[]내에 원하는 행의 인덱스 입력, 여러개의 행을 가져올시 리스트로
  - .loc[인덱스]['컬럼이름'] 으로 특정 아이템만 가져올수 있음

```
mv10= netflix.loc[10]  mv012 =
netflix.loc[[0,1,2]]
mv1_name = netflix.loc[1]['name']
mv10_5 = netflix.iloc[0:10, 0:5]
```



DataFrame 다루기

EST

- 조건을 기준으로 하위 집합 생성
  - []내에 비교연산자 조건을 입력시 bool 값의 Series 생성
  - & , | 통해 논리연산 가능
  - []내에 인덱스가 같은 bool 이터러블 입력시 true에 해당 하는 데이터만 반환

```
y_bools = netflix['Year'] == 2019  g_bools
= netflix['Category'] == Drama
netflix2019D = netflix[y_bools & g_bools]
```

```
g_bools = netflix['Category'] == Drama
neflix['is_drama'] = g_bools
neflix['ott'] = 'neflix'
```

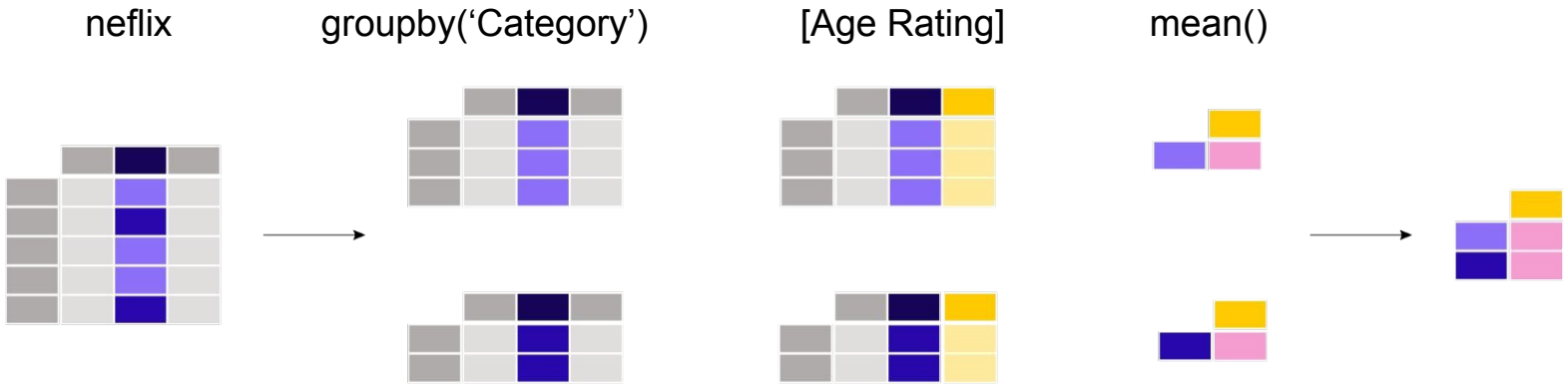
- 새로운 컬럼 입력(생성)하기
  - ['컬럼이름'] = data
  - data는 하나의 값일수도 있고 인덱스가 같은 이터러블 일수 있음
  - 컬럼이름 존재시 데이터 변환, 존재하지 않을시 마지막 축에 추가

DataFrame 다루기

- 데이터 통계 계산
  - mean(), median(), count(), sum(), std()등의 함수 사용가능
  - describe() 함수 사용하여 전반적인 통계값 확인 가능

- 그룹별 통계 계산
  - groupby()함수를 중간에 넣어 그룹별 통계계산 가능

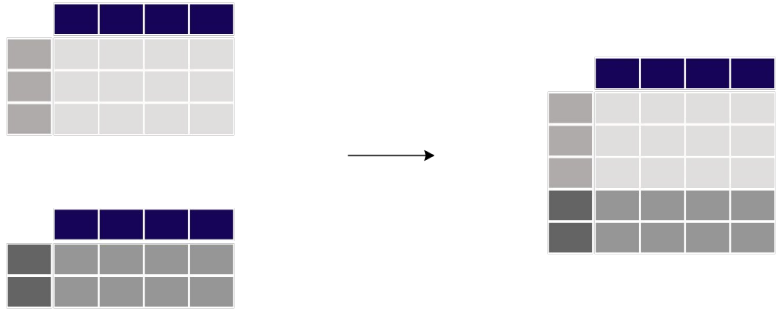
```
neflix[['Category', 'Age Rating']]
      .groupby('Category')
      .mean()
```



DataFrame 다루기

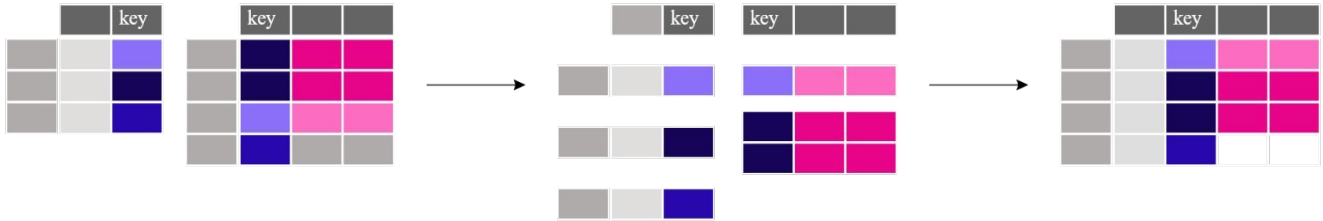
- 열을 기준으로 테이블 병합
  - concat() 함수를 사용하여 행병합
  - 열의 개수는 같아야함 (컬럼이 일치)

```
pd.concat([netflix1,netflix2])
```



- 행을 기준으로 테이블 병합
  - merge() 함수를 사용하여 열병합
  - 인덱스 또는 특정 컬럼의 값이 일치하는 항목끼리 병합

```
pd.merge(netflix1, netflix2, how="left",
on="Name")
```



OpenCV

OpenCV

OpenCV (Open Source Computer Vision)은 실시간 컴퓨터 비전을 목적으로 한 프로그래밍 라이브러리이다. 원래는 인텔이 개발하였다. **실시간** 이미지 프로세싱에 중점을 둔 라이브러리이다. 인텔 CPU에서 사용되는 경우 속도의 향상을 볼 수 있는 IPP(Intel Performance Primitives)를 지원한다. 이 라이브러리는 윈도우, 리눅스 등에서 사용 가능한 크로스 플랫폼이며 버전 4.5.0 부터는 오픈소스 아파치 라이선스 하에서, 그 이전 버전은 BSD 허가서 하에서 무료로 사용할 수 있다. OpenCV는 TensorFlow, Torch / PyTorch 및 Caffe의 딥러닝 프레임워크를 지원한다.

OpenCV: OpenCV-Python Tutorials

OpenCV 설치

Python OpenCV는 다음과 같은 네 종류의 패키지를 제공합니다.

```
opencv-python
opencv-contrib-python
opencv-python-headless
opencv-contrib-python-headless
```

`contrib`가 포함된 패키지는 **확장 모듈**이 포함된 패키지이며, 추가 모듈이 포함된 OpenCV를 설치합니다

`headless`가 포함된 패키지는 GUI 라이브러리 종속성이 없어 **서버 환경(Docker, Cloud)**에서 사용할 수 있는 OpenCV를 설치합니다.

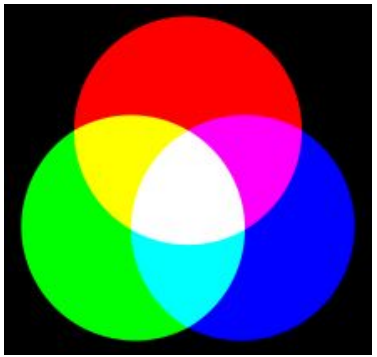
특별한 경우가 아니라면, 일반적으로 `opencv-python` 패키지를 사용합니다.

`OpenCV`는 `pip`를 통하여 설치할 수 있습니다.

명령 프롬프트나 터미널에서 `python -m pip install opencv-python` 명령어로 설치할 수 있습니다.

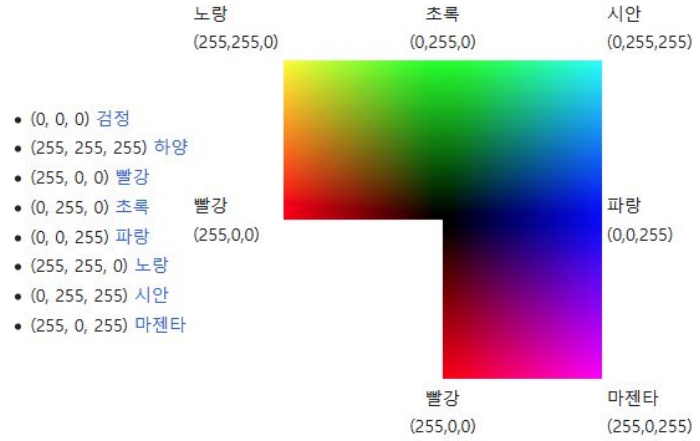
RGB

RGB 색 모형 은 빛의 삼원색을 이용하여 색을 표현하는 방식이다. 빨강(RED), 초록(GREEN), 파랑(BLUE) 세 종류의 광원(光源)을 이용하여 색을 혼합하며 색을 섞을수록 밝아지기 때문에 '가산 혼합'이라고 한다. 디지털 이미지에서 사용되는 RGB 가산혼합의 종류로는 sRGB, 어도비 RGB 등이 있다.



RGB의 분포도 [편집]

RGB 컬러의 분포도를 4각형으로 나타내면 다음과 같다.



채널별 값			혼합결과와 좌표	
빨강 (RED)	초록 (GREEN)	파랑 (BLUE)	혼합색	좌표
0	0	0	검정	(0,0,0)

각 광원의 최댓값을 255라고 한다면 빨강은 좌표(255,0,0)과 같이 표현되며 RED 광원만이 최댓값을 갖고 GREEN 광원과 BLUE광원은 빛이 없는 상태를 의미한다.

채널별 값			혼합 결과와 좌표	
빨강 (RED)	초록 (GREEN)	파랑 (BLUE)	혼합색	좌표
255	0	255	보라	(255,0,255)

좌표(80,0,0)과 같이 GREEN 광원과 BLUE 광원이 여전히 빛이 없는 상태에서 RED 광원이 약하게 비춰진다면 고동색이 될 것이다.

채널별 값			혼합결과와 좌표	
빨강 (RED)	초록 (GREEN)	파랑 (BLUE)	혼합색	좌표
90	0	0	연한 고동색	(90,0,0)

주황색은 좌표(255,127,0)으로 나타낼 수 있으며 RED 광원이 최댓값을 갖는 상태에서 GREEN 광원이 중간 정도로 섞인 상태라고 할 수 있다.

채널별 값			혼합결과와 좌표	
빨강 (RED)	초록 (GREEN)	파랑 (BLUE)	혼합색	좌표
255	127	0	주황색	(255,127,0)

세 광원이 모두 같은 값을 갖으면 무채색이 된다.

채널별 값			혼합결과와 좌표	
빨강 (RED)	초록 (GREEN)	파랑 (BLUE)	혼합색	좌표
192	192	192	은색	(192,192,192)