(Python 6강) numpy

생략되지 않은 PPT 이미지 출처는 프리코스 강좌입니다.

아래의 코드는 모두 프리코스 강의에서 발췌했습니다.

numpy part I

Numpy 정의

- Numerical Python
- 파이썬의 고성능 과학 계산용 패키지
- Matrix와 Vector와 같은 Array 연산의 사실상의 표준
- 한글로 넘파이로 주로 통칭, 넘피/늄파이라고 부르기도 함

Numpy 특징

- 일반 List에 비해 빠르고, 메모리 효율적
- 반복문 없이 데이터 배열에 대한 처리를 지원함
- 선형대수와 관련된 다양한 기능을 제공함
- C, C++, 포트란 등의 언어와 통합 가능

코드 표시

```
[1. 4. 5. 8.]
numpy.float64
```

코드 표시

```
[1. 4. 5. 8.]
numpy.float64
```

- numpy는 np.array 함수를 활용하여 배열을 생성함 -> ndarray
- numpy는 하나의 데이터 type만 배열에 넣을 수 있음
- List와 가장 큰 차이점, **Dynamic typing not supported**
- C의 Array를 사용하여 배열을 생성함

코드 표시

```
True
```

코드 표시

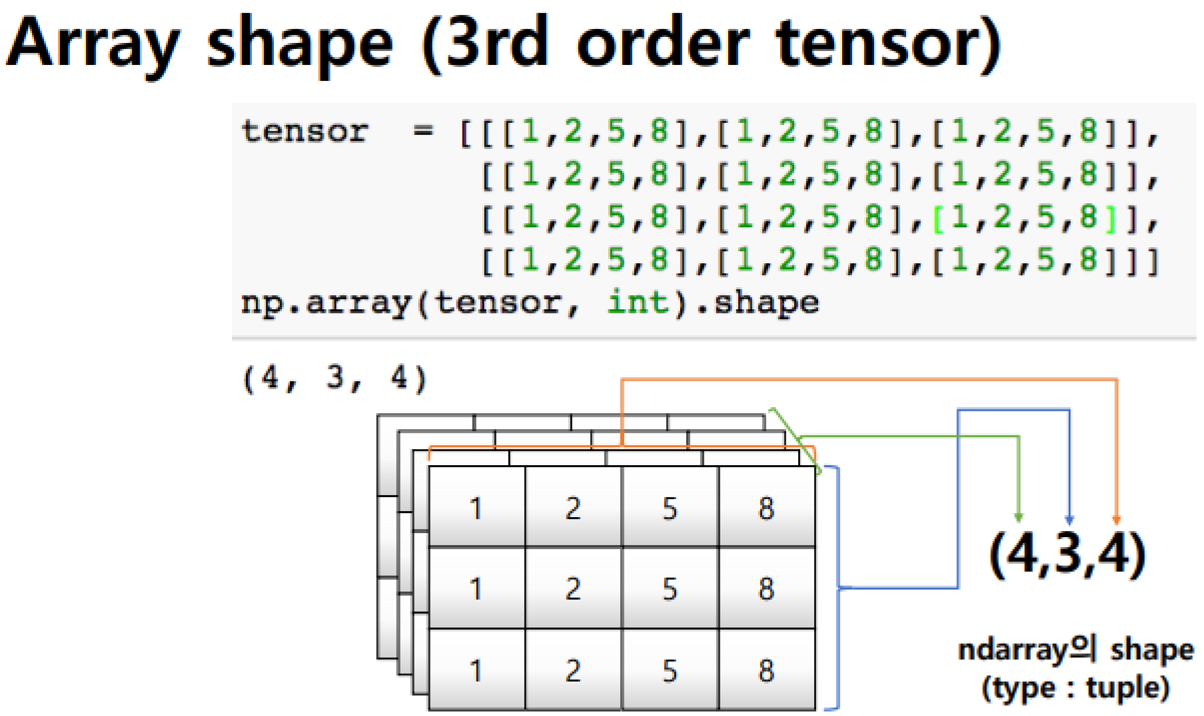
```
False
```

```
[1. 4. 5. 8.]
<class 'numpy.float64'>
float64
(4,)
```

- shape : numpy array의 object의 dimension 구성을 반환함
- dtype : numpy array의 데이터 type을 반환함

코드 표시

```
[[1. 4. 5. 8.]]
float64
(1, 4)
```



차원이 늘어날수록 기존의 숫자는 뒤로 밀려난다.

- 열 개수 -> 행 개수 -> 행렬 개수 -> 텐서 개수

  ndim : 차원의 개수 == 3

  size : 데이터의 개수 == 4 * 3 * 4

```
tensor = [[[1,2,3],[1,2,3]],
          [[1,2,3],[1,2,3]]]
```

```
np.array(tensor, int).shape
```

```
(2, 2, 3)
```

```
np.array(tensor, int).ndim
```

```
3
```

```
np.array(tensor, int).size
```

```
12
```

```
# 12*8
np.array(tensor, int).nbytes, np.array(tensor, dtype=np.int8).nbytes
```

```
(96, 12)
```

numpy part II

```
np.array(tensor, int).reshape(np.array(tensor, int).size)
```

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

```
np.array(tensor, int).reshape(2,3,2)
```

```
array([[[1, 2],
        [3, 1],
        [2, 3]],

       [[1, 2],
        [3, 1],
        [2, 3]]])
```

```
np.array(tensor, int).reshape(-1, 2)
```

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

```
np.array(tensor, int).flatten()
```

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

```
np.array(tensor, int).reshape(-1)
```

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

indexing

- List와 달리 이차원 배열에서 [0,0] 과 같은 표기법을 제공함
- Matrix 일경우 앞은 row 뒤는 column을 의미함

코드 표시

```
[[1 2 3]
 [4 5 6]]
```

```
1
1
```

코드 표시

```
[[12  2  3]
 [ 4  5  6]]
[[5 2 3]
 [4 5 6]]
```

slicing (중요)

- List와 달리 행과 열 부분을 나눠서 slicing이 가능함
- Matrix의 부분 집합을 추출할 때 유용함

코드 표시

```
array([[ 3,  4,  5],
       [ 8,  9, 10]])
```

코드 표시

```
array([7, 8])
```

코드 표시

```
array([[ 6,  7,  8,  9, 10]])
```

코드 표시

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

arrage : array 범위를 지정하여, 값의 list를 생성하는 명령어

```
tmp = np.arange(30).reshape(-1,5)
tmp
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]])
```

```
tmp[:, -1]
```

```
array([ 4,  9, 14, 19, 24, 29])
```

```
np.arange(0,5,0.5)
```

```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

ones, zeros, empty, something_like

```
np.zeros(shape=(10), dtype=np.int8)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int8)
```

```
np.zeros((2,5))
```

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

```
np.ones(shape=(10), dtype=np.int8)
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int8)
```

empty – shape만 주어지고 비어있는 ndarray 생성 (memory initialization 이 되지 않음)

```
np.empty(shape=(10), dtype=np.int8)
```

```
array([  0,  56,  86,  -9,  67,  86,   0,   0, 114, 116], dtype=int8)
```

something_like

- 기존 ndarray의 shape 크기 만큼 1, 0 또는 empty array를 반환

```
tmp = np.arange(30).reshape(-1,5)
```

```
np.ones_like(tmp)
```

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

```
np.identity(n=3, dtype=np.int8)
```

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]], dtype=int8)
```

```
np.identity(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

eye

- 대각선인 1인 행렬, k값의 시작 index의 변경이 가능

```
np.eye(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
np.eye(N=3, M=5, dtype=np.int8)
```

```
array([[1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0]], dtype=int8)
```

```
# k가 start index
np.eye(3,5,k=2)
```

```
array([[0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

diag

- 대각 행렬의 값을 추출함

```
tmp
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]])
```

```
np.diag(tmp)
```

```
array([ 0,  6, 12, 18, 24])
```

```
np.diag(tmp, k=1) # k는 시작 위치
```

```
array([ 1,  7, 13, 19])
```

```
np.diag(tmp, k=-1) # k는 시작 위치
```

```
array([ 5, 11, 17, 23, 29])
```

```
np.diag(tmp, k=-2) # k는 시작 위치
```

```
array([10, 16, 22, 28])
```

random sampling

- 데이터 분포에 따른 sampling으로 array를 생성

```
np.random.uniform(0,1,10).reshape(2,5)
```

```
array([[0.65417069, 0.99604841, 0.54265776, 0.31915918, 0.25604368],
       [0.71732909, 0.92341006, 0.00290958, 0.62269923, 0.82051635]])
```

```
np.random.normal(0,1,10).reshape(2,5)
```

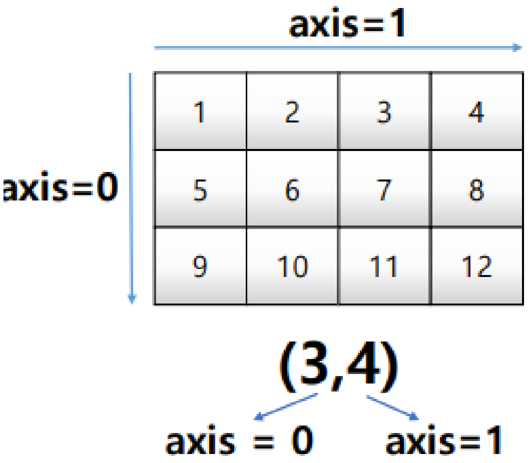
```
array([[ 0.95729854, -0.59040512,  0.12273789,  0.39895121,  0.74972424],
       [-1.10598663, -0.6629631 , -0.75311198,  1.91000071, -1.09619721]])
```

```
np.random.exponential(scale=2, size=100) # 모수값 지정
```

```
array([4.73275167e+00, 6.43817547e-01, 3.63205016e+00, 7.93260483e-01,
       4.77055854e+00, 7.57845268e+00, 1.18762775e+00, 1.50032332e+00,
       7.25003790e-03, 3.86756035e+00, 3.82469454e+00, 2.50185834e+00,
       2.00952559e-01, 3.56404860e-02, 5.20656983e-02, 2.13440775e+00,
       9.52504050e-02, 1.27743795e+00, 3.25767235e+00, 3.02533621e-01,
       1.49502207e+00, 2.90198834e-01, 9.24393567e-01, 8.67822176e-01,
       4.68373955e-01, 2.04050881e-01, 3.66685944e+00, 7.57692874e-01,
       1.09592346e+00, 3.90258634e+00, 2.22884516e-01, 1.36424642e+00,
       1.10277351e+00, 8.90669061e-01, 2.51212037e+00, 5.21412817e+00,
       6.92348372e+00, 2.20158432e+00, 4.07831898e-01, 1.20425325e+00,
       8.10177943e-01, 1.17833407e+00, 1.39273512e-01, 2.81473862e+00,
       1.80383527e+00, 3.43489976e+00, 2.02058208e+00, 1.88515291e-01,
       8.98248553e-01, 1.65467440e+00, 2.67152507e-01, 5.77174874e-01,
       2.39270583e-01, 2.73226457e-01, 1.02211992e+01, 4.00992698e+00,
       5.24445516e-01, 3.37810128e+00, 1.32424704e+00, 4.15659451e+00,
       1.58210218e+00, 8.24257071e-01, 4.09541093e-01, 1.14896940e+00,
       3.47736266e-01, 2.26570850e-01, 3.91256142e-01, 5.70658624e-01,
       1.20941583e+00, 3.38363340e-01, 4.64465564e+00, 1.89647206e+00,
       8.18433686e-01, 5.19698427e+00, 1.54685724e+00, 3.54987403e-01,
       1.43901482e+00, 6.70031376e-01, 9.80960500e-02, 2.18118208e+00,
       3.15022092e+00, 1.36132272e+00, 6.15328728e-01, 1.07547455e+01,
       1.00495487e+00, 4.27301399e-01, 1.13750553e+00, 4.13666178e+00,
       1.07633486e+00, 8.78573751e-01, 7.37054384e+00, 6.23567022e-01,
       1.13528559e+00, 2.02879750e+00, 1.02494806e+00, 2.75484984e+00,
       6.61774180e-01, 2.77578109e+00, 1.23023482e+00, 2.15160107e+00])
```

axis

- 모든 operation function을 실행할 때, 기준이 되는 dimension 축




```
test_array = np.arange(1,13).reshape(3,4)
test_array
```

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

```
test_array.sum(axis=1), test_array.sum(axis=0)
```

```
(array([10, 26, 42]), array([15, 18, 21, 24]))
```

```
tmp = np.arange(9).reshape(-1,3)
tmp
```

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

```
tmp.sum()
```

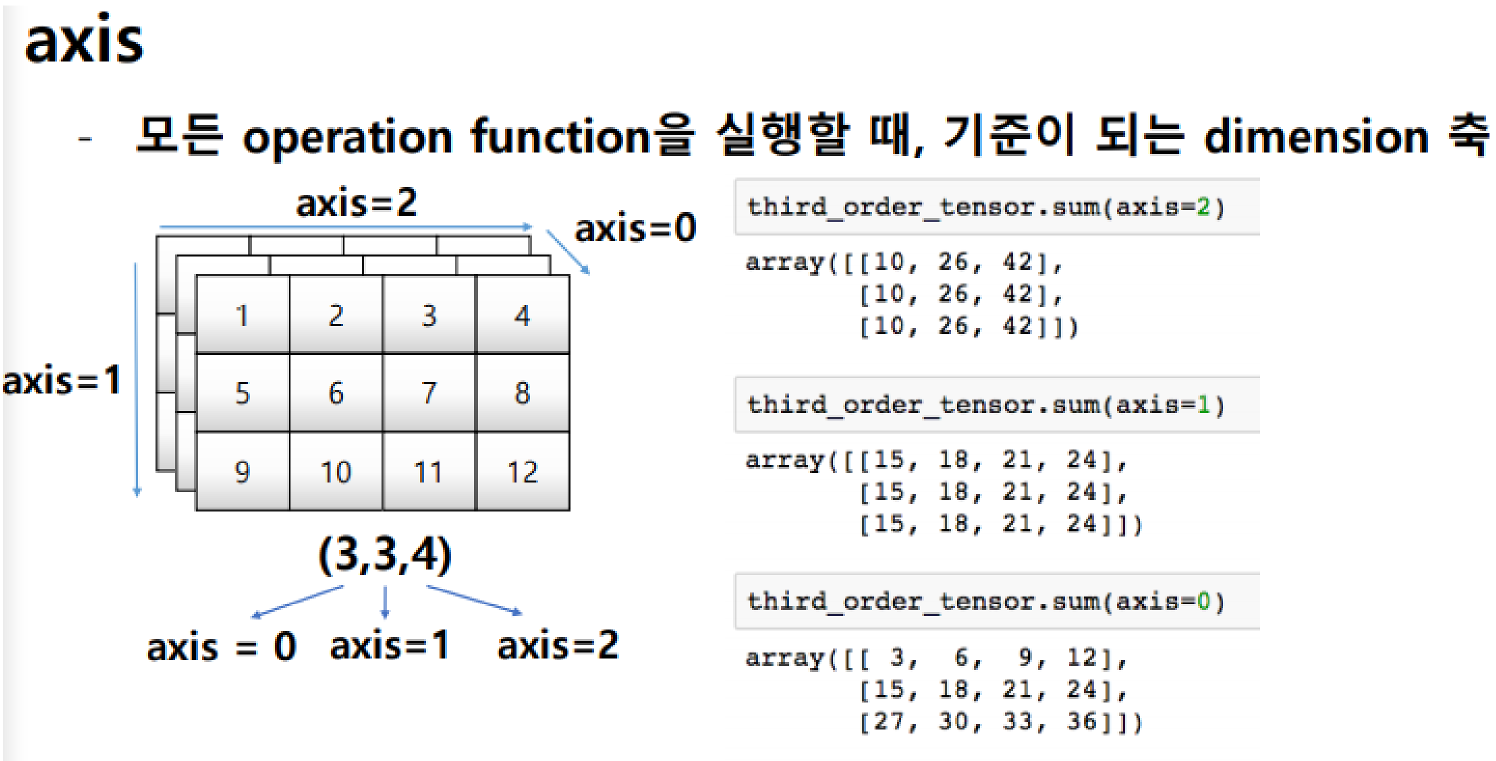
```
36
```

```
tmp.sum(axis=0)
```

```
array([ 9, 12, 15])
```

```
tmp.sum(axis=1)
```

```
array([ 3, 12, 21])
```




```
tmp = np.arange(8).reshape(2,2,2)
tmp
```

```
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

```
tmp.sum(axis=0)
```

```
array([[ 4,  6],
       [ 8, 10]])
```

```
tmp.sum(axis=1)
```

```
array([[ 2,  4],
       [10, 12]])
```

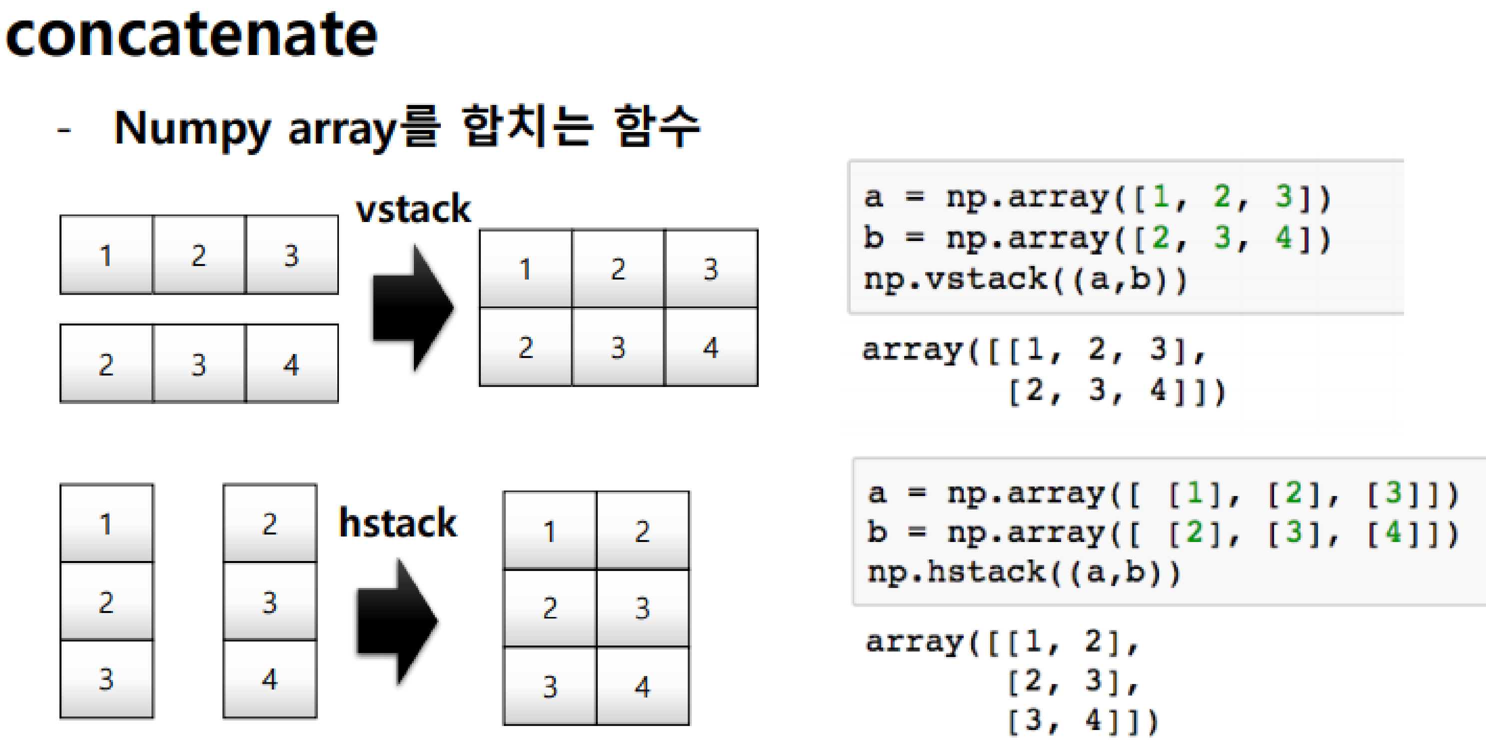
```
tmp.sum(axis=2)
```

```
array([[ 1,  5],
       [ 9, 13]])
```

```
tmp.std()
```

```
2.29128784747792
```

conaternate




```
a = np.array([1,2])
b = np.array([3,4])
np.vstack((a,b))
```

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

```
a = np.array([[1],[2]])
b = np.array([[3],[4]])
np.hstack((a,b))
```

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

```
a = np.array([1,2])
b = np.array([3,4])
np.concatenate((a,b), axis=0)
```

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

```
a = np.array([[1,2], [3,4]])
```

```
a = np.array([[1,2], [3,4]])
b = np.array([[5,6]])
np.concatenate((a,b.T), axis=1)
```

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

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

# 축 하나 늘리기
b = b[np.newaxis, :]
c = b.reshape(-1, 2)
b, c
```

```
(array([[5, 6]]), array([[5, 6]]))
```

```
np.concatenate((a, b.T), axis=1)
```

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

Element-wise 연산, Dot product, Transpose

```
a = np.array([1,2])
b = np.array([3,4])
a * b
```

```
array([3, 8])
```

```
a @ b
```

```
11
```

```
a = np.array([[1,2],[3,4]])
a.T
```

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

```
a.transpose()
```

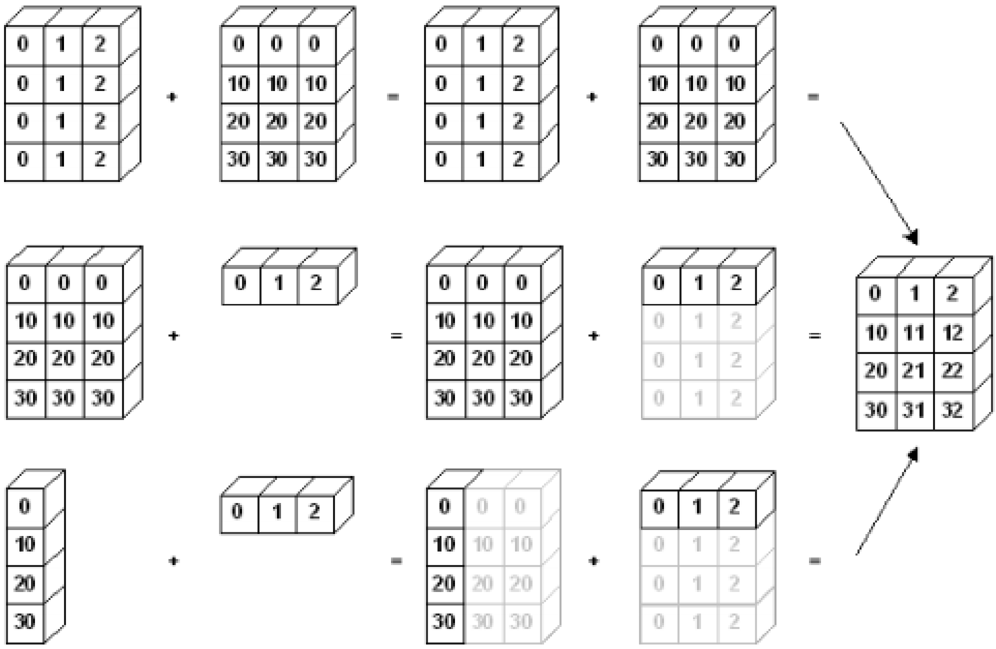
```
array([[1, 3],
       [2, 4]])
```

broadcasting

shape이 다른 배열 간 연산을 지원하는 기능

broadcasting

- Scalar – vector 외에도 vector – matrix 간의 연산도 지원



```
tmp = np.arange(1,13).reshape(-1,3)
vector = np.arange(3) # [0, 1, 2]
tmp
```

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

```
tmp + vector
```

```
array([[ 1,  3,  5],
       [ 4,  6,  8],
       [ 7,  9, 11],
       [10, 12, 14]])
```

Numpy performance

```
def sclar_vector_product(scalar, vector):
 result = []
 for value in vector:
  result.append(scalar * value)
 return result

iternation_max = 100
vector = list(range(iternation_max))
scalar = 2

%timeit sclar_vector_product(scalar, vector) # for loop을 이용한 성능
%timeit [scalar * value for value in range(iternation_max)] # list comprehension을 이용한 성능
%timeit np.arange(iternation_max) * scalar # numpy를 이용한 성능
```

```
100000 loops, best of 5: 10.3 µs per loop
100000 loops, best of 5: 7.31 µs per loop
```

```
The slowest run took 506.30 times longer than the fastest. This could mean that an intermediate result i
100000 loops, best of 5: 1.89 µs per loop
```

- 일반적으로 속도는 아래 순 for loop < list comprehension < numpy
- 100,000,000 번의 loop이 돌 때 약 약 4배 이상의 성능 차이를 보임
- Numpy는 C로 구현되어 있어, 성능을 확보하는 대신
- 파이썬의 가장 큰 특징인 dynamic typing을 포기함
- 대용량 계산에서는 가장 흔히 사용됨
- Concatenate 처럼 계산이 아닌, 할당에서는 연산 속도의 이점이 없음

numpy part III

comparisons

```python
a = np.arange(10)
a
```

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

```python
a < 0
```

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

```python
np.any(a>5)
```

```
True
```

```python
np.all(a>5)
```

```
False
```

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

```python
a > b
```

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

```python
a == b
```

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

```python
(a > b).any()
```

```
False
```

```python
a = np.array([1,3,0])
```

```
a = np.array([1,3,0])
np.logical_and(a>0, a<3)
```

    array([True, False, False])

```
a = np.array([1,3,0])
np.logical_not(a)
```

    array([False, False, True])

```
a = np.array([1,0,0], bool)
b = np.array([1,0,1], bool)
np.logical_or(a, b)
```

    array([True, False, True])

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

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

np.where

```
np.where(a>0, 3, 2)
```

    array([2, 3, 3, 3, 3, 3, 3, 3, 3, 3])

```
# 인덱스 값 반환
np.where(a>0)
```

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

```
a = np.array([1,np.NaN, np.Inf], float)
a
```

    array([1., nan, inf])

```
np.isnan(a)
```

    array([False, True, False])

```
# 수렴값 찾기
np.isfinite(a)
```

    array([True, False, False])

argmax & argmin

- array내 최대값 또는 최소값의 index를 반환함

```
a = np.arange(10)
a, np.argmax(a), np.argmin(a)
```

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

- axis 기반의 반환

```
a = np.arange(10).reshape(-1,5)
a, np.argmax(a, axis=1), np.argmin(a, axis=0)
```

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

```
a.argsort(), a.argsort()[::-1], a[np.argmin(a)]
```

```
(array([[0, 1, 2, 3, 4],
        [0, 1, 2, 3, 4]]), array([[0, 1, 2, 3, 4],
        [0, 1, 2, 3, 4]]), array([0, 1, 2, 3, 4]))
```

```
a = np.array([[1,2],[3,4]])
np.argmax(a, axis=1), np.argmin(a, axis=0)
```

```
(array([1, 1]), array([0, 0]))
```

boolean index

- numpy는 배열은 특정 조건에 따른 값을 배열 형태로 추출 할 수 있음
- Comparison operation 함수들도 모두 사용가능
- boolean list 사용
- array shape <= boolean index shape

```
a = np.arange(10)
bi = a > 3
bi
```

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

```
# True인 index의 요소 추출
a[a>3]
```

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

```
bi2 = bi[:5]
bi2
```

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

```
a[bi2]
```

```
bi3 = bi[:] + [True]
a[bi3]
```

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

fancy index

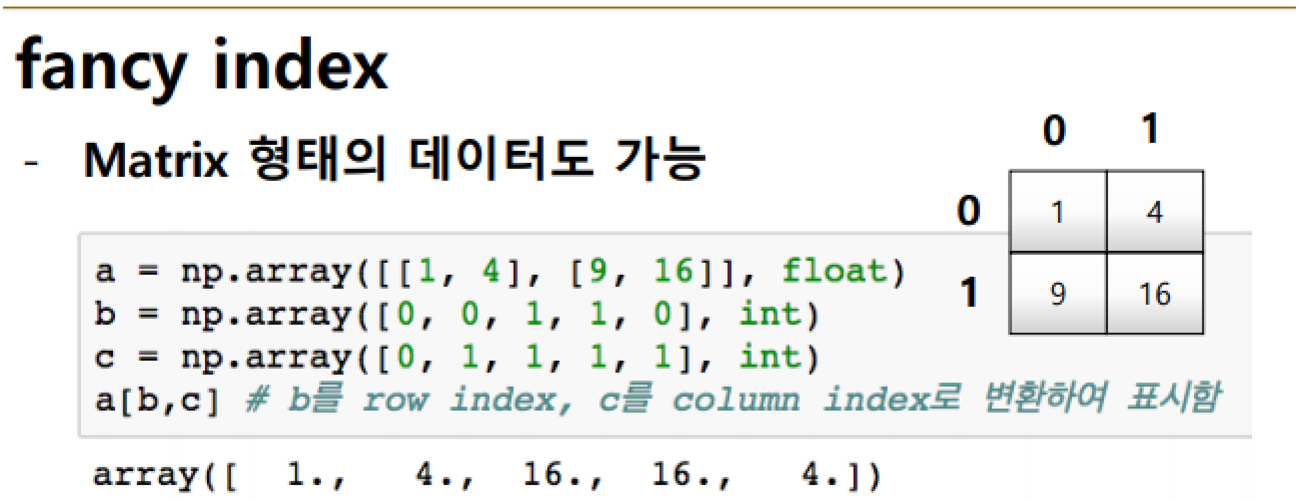
- numpy는 array를 index value로 사용해서 값을 추출하는 방법
- integer list 사용
- array와 boolean index shape 같지 않아도 된다.

```
a = np.arange(10,20)
b = np.array([0,0,1,3,2,1])
a, a[b]
```

```
(array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19]),
 array([10, 10, 11, 13, 12, 11]))
```

```
a.take(b)
```

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




```
a = np.array([[1, 4], [9, 16]], float)
b = np.array([0, 0, 1, 1, 0], int)
c = np.array([0, 1, 1, 1, 1], int)
a[b,c] # b를 row index, c를 column index로 변환하여 표시함
```

```
array([ 1.,  4., 16., 16.,  4.])
```

```
a = np.array([[1, 4], [9, 16]], float)
a[b] # 행만
```

```
array([[ 1.,  4.],
       [ 1.,  4.],
       [ 9., 16.],
       [ 9., 16.],
       [ 1.,  4.]])
```

loadtxt & savetxt

- Text type의 데이터를 읽고, 저장하는 기능

```
!wget https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/codes/ch_2/2/populat
```

```
--2021-08-06 04:19:09--  https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/code
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.109.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 525 [text/plain]
Saving to: 'populations.txt'

populations.txt     100%[===================>]     525  --.-KB/s    in 0s

2021-08-06 04:19:09 (31.2 MB/s) - 'populations.txt' saved [525/525]
```

```
# 파일 호출
a = np.loadtxt("./populations.txt")
a[:10]
```

```
array([[ 1900., 30000.,  4000., 48300.],
       [ 1901., 47200.,  6100., 48200.],
       [ 1902., 70200.,  9800., 41500.],
       [ 1903., 77400., 35200., 38200.],
       [ 1904., 36300., 59400., 40600.],
       [ 1905., 20600., 41700., 39800.],
       [ 1906., 18100., 19000., 38600.],
       [ 1907., 21400., 13000., 42300.],
       [ 1908., 22000.,  8300., 44500.],
       [ 1909., 25400.,  9100., 42100.]])
```

```
# 형 변환
a_int = a.astype(int)
a_int[:3]
```

```
array([[ 1900, 30000,  4000, 48300],
       [ 1901, 47200,  6100, 48200],
       [ 1902, 70200,  9800, 41500]])
```

```
# 파일 저장
np.savetxt('int_data.csv',a_int, delimiter=",")
```

```
!cat int_data.csv
```

```
1.900000000000000000e+03,3.000000000000000000e+04,4.000000000000000000e+03,4.830000000000000000e+04
1.901000000000000000e+03,4.720000000000000000e+04,6.100000000000000000e+03,4.820000000000000000e+04
1.902000000000000000e+03,7.020000000000000000e+04,9.800000000000000000e+03,4.150000000000000000e+04
1.903000000000000000e+03,7.740000000000000000e+04,3.520000000000000000e+04,3.820000000000000000e+04
1.904000000000000000e+03,3.630000000000000000e+04,5.940000000000000000e+04,4.060000000000000000e+04
```

```
1.905000000000000000e+03,2.060000000000000000e+04,4.170000000000000000e+04,3.980000000000000000e+04
1.906000000000000000e+03,1.810000000000000000e+04,1.900000000000000000e+04,3.860000000000000000e+04
1.907000000000000000e+03,2.140000000000000000e+04,1.300000000000000000e+04,4.230000000000000000e+04
1.908000000000000000e+03,2.200000000000000000e+04,8.300000000000000000e+03,4.450000000000000000e+04
1.909000000000000000e+03,2.540000000000000000e+04,9.100000000000000000e+03,4.210000000000000000e+04
1.910000000000000000e+03,2.710000000000000000e+04,7.400000000000000000e+03,4.600000000000000000e+04
1.911000000000000000e+03,4.030000000000000000e+04,8.000000000000000000e+03,4.680000000000000000e+04
1.912000000000000000e+03,5.700000000000000000e+04,1.230000000000000000e+04,4.380000000000000000e+04
1.913000000000000000e+03,7.660000000000000000e+04,1.950000000000000000e+04,4.090000000000000000e+04
1.914000000000000000e+03,5.230000000000000000e+04,4.570000000000000000e+04,3.940000000000000000e+04
1.915000000000000000e+03,1.950000000000000000e+04,5.110000000000000000e+04,3.900000000000000000e+04
1.916000000000000000e+03,1.120000000000000000e+04,2.970000000000000000e+04,3.670000000000000000e+04
1.917000000000000000e+03,7.600000000000000000e+03,1.580000000000000000e+04,4.180000000000000000e+04
1.918000000000000000e+03,1.460000000000000000e+04,9.700000000000000000e+03,4.330000000000000000e+04
1.919000000000000000e+03,1.620000000000000000e+04,1.010000000000000000e+04,4.130000000000000000e+04
1.920000000000000000e+03,2.470000000000000000e+04,8.600000000000000000e+03,4.730000000000000000e+04
```

numpy object - npy

- Numpy object (pickle) 형태로 데이터를 저장하고 불러옴
- Binary 파일 형태로 저장함

```
np.save("npy_test", arr=a_int) # 저장할 파일명, 저장할 행렬
```

```
npy_array = np.load(file="npy_test.npy")
npy_array[:3]
```

```
array([[ 1900, 30000,  4000, 48300],
       [ 1901, 47200,  6100, 48200],
       [ 1902, 70200,  9800, 41500]])
```

(Python 7-1강) pandas I

PPT 필기

↳ 숨겨진 셀 5개

이하 코드 (프리코스 강의에서 변형)

[ ] ↳ 숨겨진 셀 6개

Series

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

```
list_data = [1,2,3,4,5]
list_name = ["a","b","c","d","e"]
```

```
example_obj = Series(data = list_data, index=list_name)
example_obj
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

```
example_obj.index
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

```
example_obj.values
```

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

```
type(example_obj.values)
```

```
numpy.ndarray
```

```
dict_data = {"a":1, "b":2, "c":3, "d":4, "e":5}
example_obj = Series(dict_data, dtype=np.float32, name="example_data")
example_obj
```

```
a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
Name: example_data, dtype: float32
```

```
# object 이름
example_obj.name = "number"

# index 이름
example_obj.index.name = "alphabet"
example_obj
```

```
alphabet
a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
Name: number, dtype: float32
```

```
example_obj.to_dict()
```

```
{'a': 1.0, 'b': 2.0, 'c': 3.0, 'd': 4.0, 'e': 5.0}
```

```
"b" in example_obj
```

```
True
```

```
np.exp(example_obj) #np.abs , np.log
```

```
np.exp(example_obj) #np.abs , np.log
```

```
alphabet
a      2.718282
b      7.389056
c     20.085537
d     54.598148
e    148.413162
Name: number, dtype: float32
```

```
dict_data_1 = {"a":1, "b":2, "c":3, "d":4, "e":5}
indexes = ["a","b","c","d","e","f","g","h"]
series_obj_1 = Series(dict_data_1, index=indexes)
series_obj_1
```

```
a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
f    NaN
g    NaN
h    NaN
dtype: float64
```

DataFrame

```
raw_data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'last_name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze'],
        'age': [42, 52, 36, 24, 73],
        'city': ['San Francisco', 'Baltimore', 'Miami', 'Douglas', 'Boston']}
df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'city'])
df
```

```
pd.DataFrame(raw_data, columns = ["age", "city"])
```

```
pd.DataFrame(raw_data,
          columns = ["first_name","last_name","age", "city", "debt"]
         )
```

```
df = pd.DataFrame(raw_data, columns = ["first_name","last_name","age", "city", "debt"])
df.first_name
```

```
0    Jason
1    Molly
2     Tina
3     Jake
4      Amy
Name: first_name, dtype: object
```

```
df["first_name"]
```

```
0    Jason
1    Molly
2     Tina
3     Jake
4      Amy
Name: first_name, dtype: object
```

```
# index 이름
df.loc[1]
```

```
first_name        Molly
last_name      Jacobson
age                  52
city          Baltimore
debt                NaN
Name: 1, dtype: object
```

```
df.loc[:3]
```

```
df.loc[:, 'last_name']
```

```
0      Miller
1    Jacobson
2         Ali
3      Milner
4       Cooze
Name: last_name, dtype: object
```

```
df.loc[:, ['first_name', 'last_name']]
```

```
# index 번호
df["age"].iloc[1:]
```

```
1    52
2    36
3    24
4    73
Name: age, dtype: int64
```

```
s = pd.Series(np.nan, index=[49,48,47,46,45, 1, 2, 3, 4, 5])
s
```

```
49   NaN
48   NaN
47   NaN
46   NaN
45   NaN
1    NaN
2    NaN
3    NaN
4    NaN
5    NaN
dtype: float64
```

```
s.loc[:3]
```

```
49   NaN
48   NaN
47   NaN
46   NaN
45   NaN
1    NaN
2    NaN
3    NaN
dtype: float64
```

```
s.iloc[:3]
```

```
49   NaN
```

```
48   NaN
47   NaN
dtype: float64
```

```
df.debt = df.age > 40
df
```

```
values = Series(data=["M","F","F"],index=[0,1,3])
values
```

```
0    M
1    F
3    F
dtype: object
```

```
df["sex"] = values
df
```

```
df.T
```

```
df.values
```

```
array([['Jason', 'Miller', 42, 'San Francisco', True, 'M'],
       ['Molly', 'Jacobson', 52, 'Baltimore', True, 'F'],
```

```
['Tina', 'Ali', 36, 'Miami', False, nan],
['Jake', 'Milner', 24, 'Douglas', False, 'F'],
['Amy', 'Cooze', 73, 'Boston', True, nan]], dtype=object)
```

```
df.to_csv()
```

- **del** : 메모리에서 삭제
- **drop** : 뷰에서 안 보임

```
del df["debt"]
df
```

```
# Example from Python for data analyis

pop = {'Nevada': {2001: 2.4, 2002: 2.9},
 'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}

DataFrame(pop)
```

Selection & Drop

```
!wget https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_2/3/data/excel-comp-da
```

```
--2021-08-06 04:50:55--  https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_2/3/da
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/codes/ch_2/3/data/e
--2021-08-06 04:50:55--  https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/code
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11990 (12K) [application/octet-stream]
```

```
Saving to: ‘excel-comp-data.xlsx’

excel-comp-data.xls 100%[===================>]  11.71K  --.-KB/s    in 0s

2021-08-06 04:50:55 (94.6 MB/s) - ‘excel-comp-data.xlsx’ saved [11990/11990]
```

```
!cat excel-comp-data.xlsx
```


```
����└�PK└┘¶�─���!�����K•���↔"��‼���xl/theme/theme1.xml�ZKo←7┼�┤�⎕ ��X�%�2"•�
¶M�^
dX�¶�aF�8�(n�+(♬P�S.�Ai��.-���_�₩U��x��┐�}¶ﬁGZ↕$CA3U☼�┐┘�y���7/��7/��>~q��
┬��ş�↑¶q�↑▼��n��qmk�AM����}3&��{�
G$%
�w��┼☼�CJ↔�n�Pp��
=�����$↔�u┐iB�I‼���' �9─�,��└��ڈv��┼�y��┼��6▲(��Xvp□─��*�   y0¶a◄ג
<↔◄�Q�G����
�EGc┼9���'��⎕�#A�g��'B∧��I↕_`��n����H�t33��[T:!{@">C����3�i��,�;�u'tao�
�⎕┤�1���y�↕h��4P�/I�♬f�┐���`��|�┬H└�0H��j�─^����‼7@7�49      M����0
P��zÏ┌─₩&├x]��↔�"8��a⎕n2�p�x��☼�2�¶كنv�@&���2��a^�!�R3N6�/V������→]L���
���Z��P┴�]xb��─��R>PD└L‼�e└����:TA�▲�0I7┴A�������y�┬�<�G1─g�c��8/�:EG�I�
�t��└.Ţ
₩�{{�uSM�2`pg�Ↄ�3�←�&�����d��=�쬆���H�P�♂[Aↄ���j₩♬□�C�5ﻳ��b��[m─•¶p.� &B*B
�d�,
�└����(��9��H�
E�ER¶D9$��(��9��HJ
��←���~����□�œ%�����•�{┤�B:g
─�}�z����:�V�_�₩>)1P┌��┴To財┤�_~�ŭ�F→sN��•��0K��?[}���c↕`6•─�d^�⎕W�����;
�└�0N└[���;�Mk�─oI�W��y¶�<J└x�;)─[��M�ue� ��▼─┐�4w[o�ű�6└C�→f�m7�M�j8����4�
��#└���→�칠;XA�└I��ҕ��!%Q|•ʁ�x�^��ɿ���♂├+�→G�S'┐R�¨:��□� ┐�P�(��qt#←}���
K�Ϭ3��?������Ne��☼LkT~�┤:�G[4��←├Y���♂�i�*` a%dE&(��$�)� ┴{���┼E⎕�+��� �~
,�dNd�;��
��i垠ۊ�G�FH♬/├M����A=�4��┴�$��>6�♀s�=�s��/����!��r▼u�I�gZ�ӥcmN┌8[�v���
��^0�^�#nI �┴♂���&�¶² ←�╕Q+Itv~���n��z♬��#��M^�┌|�d
&^k����┌�i��k,�]_>�▼♬•☼E☼��h_s,��K��`ɪtIJd�;�9�vM�!�|├����S��~P����,
Ӟ�☼X��•���+�◄��(n�ٜ�←Kxo─�(bn�C�����┼6��!□$�m�����▼�A��X���)��♂��^ـn
>��ِ�─��i6�3�ٲ�ڃ}�c��3�6�┐�ٺ}┬b��g◄f�>�;�&�0�L─a�F↕���K�.`�iFĪ�↑◄t� Qt� 
�C;•C�(t#U"�*�R�����1•��*����0�,���"H�
'����¶↔�L┴��42�B7T�_,��r�T�:┴�u��A��0+─���"H+─����pb�%�I¶1���1��P��D�
��~�J�n.��'�`{!:�^�♬�W←SB���♬□�nt����廓5uMS┴e�X���7}�x�₩Y�%X¶�].ow;x/as7��
��g�`┐�#�����>�%@��PD�} �P3@4‼T�&�I<�����└�PK└┘¶�─���!�;!�←�┌��┬└���◄┌docP
4�8└﨟>��B←���a?E6I�6�>xW�J�♂>Q�����C/��F��1;���d���;→�ÉKs�����o�.`♂⎕r&
�������������h↕��xl/styles.xmlPK┌┐─�¶�─���!�0^H⎕�└����¶������������
```


```
# 엑셀 핸들링 모듈
# pip3 install xlrd


import numpy as np
df = pd.read_excel("excel-comp-data.xlsx")
df.head()
```

```
df.head().T
```

```
account_serires = df["account"]
account_serires[:3]
```

```
0    211829
1    320563
2    648336
Name: account, dtype: int64
```

```
# row index 추출
account_serires[[1,5,2]]
```

```
1    320563
5    132971
2    648336
Name: account, dtype: int64
```

```
df.index = df["account"]
```

```
df.loc[[211829,320563],["name","street"]]
```

```
df[["name","street"]][:2]
```

```
df[["name", "street"]].iloc[:10]
```

인덱스 재설정

```
df.index = list(range(0,15))
# df.reset_index(inplace=True)
df.head()
```

```
df.drop(1)
```

```
df.drop([0,1, 2,3])
```

```
df.drop("city",axis=1).head()
```

```
df.drop(["city", "state"],axis=1)
```

Dataframe Operations

```
s1 = Series(
    range(1,6), index=list("abced"))
s1
```

```
a    1
b    2
c    3
e    4
d    5
dtype: int64
```

```
s2 = Series(
    range(5,11), index=list("bcedef"))
s2
```

```
b     5
c     6
e     7
d     8
e     9
f    10
dtype: int64
```

```
s1 + s2
```

```
a     NaN
b     7.0
c     9.0
d    13.0
e    11.0
e    13.0
f     NaN
dtype: float64
```

```
s1.add(s2, fill_value=0)
```

```
a     1.0
b     7.0
c     9.0
d    13.0
e    11.0
e    13.0
f    10.0
dtype: float64
```

```
df1 = DataFrame(
    np.arange(9).reshape(3,3),
    columns=list("abc"))
df1
```

```
df2 = DataFrame(
    np.arange(16).reshape(4,4),
    columns=list("abcd"))
df2
```

```
df1 + df2
```

```
df1.add(df2,fill_value=0)
```

```
df = DataFrame(
    np.arange(16).reshape(4,4),
    columns=list("abcd"))
df
```

```
s = Series(
    np.arange(10,14),
    index=list("abcd"))
s
```

```
a    10
b    11
c    12
d    13
dtype: int64
```

```
df + s
```

```
s2 = Series(np.arange(10,14))
s2
```

```
0    10
1    11
2    12
3    13
dtype: int64
```

```
df + s2
```

```
# axix를 기준으로 row broadcating 실행
df.add(s2, axis=0)
```

lambda, map, apply

map for series

- pandas의 series type의 데이터에도 map 함수 사용가능
- function 대신 dict, sequence형 자료등으로 대체 가능

replace function

- Map 함수의 기능중 데이터 변환 기능만 담당
- 데이터 변환시 많이 사용하는 함수

apply for dataframe

- map과 달리, series **전체(column)**에 해당 함수를 적용
- 입력 값이 series 데이터로 입력 받아 handling 가능
- 내장 연산 함수를 사용할 때도 똑같은 효과를 거둘 수 있음
- mean, std 등 사용가능
- scalar 값 이외에 series값의 반환도 가능함

applymap for dataframe

- series 단위가 아닌 element 단위로 함수를 적용함
- series 단위에 apply를 적용시킬 때와 같은 효과

```
s1 = Series(np.arange(10))
s1
```

```
0    0
1    1
2    2
3    3
4    4
5    5
6    6
7    7
8    8
9    9
dtype: int64
```

```
s1.map(lambda x: x**2).head(5)
```

```
0     0
1     1
2     4
3     9
4    16
dtype: int64
```

```
ex = [1,2,3]
f = lambda x, y: x + y
list(map(f, ex, ex))
```

```
[2, 4, 6]
```

```
z = {1: 'A', 2: 'B', 3: 'C'}
s1.map(z)
```

```
0    NaN
1      A
2      B
3      C
4    NaN
5    NaN
6    NaN
```

```
7    NaN
8    NaN
9    NaN
dtype: object
```

```
s2 = Series(np.arange(10,20))
s1.map(s2)
```

```
0    10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int64
```

```
!wget https://raw.githubusercontent.com/rstudio/Intro/master/data/wages.csv
```

```
--2021-08-06 05:48:42--  https://raw.githubusercontent.com/rstudio/Intro/master/data/wages.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 62250 (61K) [text/plain]
Saving to: 'wages.csv'

wages.csv          100%[===================>]  60.79K  --.-KB/s    in 0.01s

2021-08-06 05:48:42 (4.92 MB/s) - 'wages.csv' saved [62250/62250]
```

```
df = pd.read_csv("wages.csv")
df.head()
```

```
df.sex.unique()
```

```
array(['male', 'female'], dtype=object)
```

```
df["sex_code"] =  df.sex.map({"male":0, "female":1})
df.head(5)
```

```
df.sex.replace(
    {"male":0, "female":1}
).head()
```

```
0    0
1    1
2    1
3    1
4    1
Name: sex, dtype: int64
```

```
df.sex.replace(
    ["male", "female"],
    [0,1], inplace=True)
```

```
df
```

```
del df["sex_code"]
```

```
df
```

```
df = pd.read_csv("wages.csv")
df.head()
```

```
df_info = df[["earn", "height","age"]]
df_info.head()
```

```
f = lambda x : x.max() - x.min()
df_info.apply(f)
```

```
earn      318047.708444
height        19.870000
age           73.000000
dtype: float64
```

```
df_info.apply(sum)
```

```
earn      4.474344e+07
height    9.183125e+04
```

```
age       6.250800e+04
dtype: float64
```

```
df_info.sum()
```

```
earn      4.474344e+07
height    9.183125e+04
age       6.250800e+04
dtype: float64
```

```
def f(x):
    return Series([x.min(), x.max(), x.mean()],
                  index=["min", "max", "mean"])
df_info.apply(f)
```

```
f = lambda x : -x
df_info.applymap(f).head(5)
```

```
f = lambda x : -x
df_info["earn"].apply(f).head(5)
```

```
0   -79571.299011
1   -96396.988643
2   -48710.666947
3   -80478.096153
4   -82089.345498
Name: earn, dtype: float64
```

Pandas Built-in functions

```
df.describe()
```

```
df.race.unique()
```

```
array(['white', 'other', 'hispanic', 'black'], dtype=object)
```

```
# 라벨 인코딩, 사용 빈도 적음
dict(enumerate(sorted(df["race"].unique())))
```

```
{0: 'black', 1: 'hispanic', 2: 'other', 3: 'white'}
```

```
np.array(list(enumerate(df["race"].unique())), dtype=str)
```

```
array([['0', 'white'],
       ['1', 'other'],
       ['2', 'hispanic'],
       ['3', 'black']], dtype='<U8')
```

```
np.array(list(enumerate(df["race"].unique())))[:, 0]
```

```
array(['0', '1', '2', '3'], dtype='<U21')
```

```
value = list(map(int, np.array(list(enumerate(df["race"].unique())))[:, 0].tolist()))
key = np.array(list(enumerate(df["race"].unique())), dtype=str)[:, 1].tolist()

value, key
```

```
([0, 1, 2, 3], ['white', 'other', 'hispanic', 'black'])
```

```
df["race"].replace(to_replace=key, value=value, inplace=True)
```

```
df["race"]
```

```
0       0
1       0
2       0
3       1
4       0
       ..
1374    0
1375    0
1376    0
1377    0
1378    0
Name: race, Length: 1379, dtype: int64
```

코드 표시

```
earn       float64
height     float64
sex         object
race         int64
ed           int64
age          int64
dtype: object
```

코드 표시

코드 표시

```
earn                                               4.47434e+07
height                                                 91831.2
sex       malefemalefemalefemalefemalefemalefemalemalema...
race                                                       561
ed                                                       18416
age                                                      62508
dtype: object
```

코드 표시

```
0       79710.189011
1       96541.218643
2       48823.436947
3       80653.316153
4       82212.425498
            ...     
1374    30290.060363
1375    25018.829514
1376    13823.311312
1377    95563.664410
1378     9686.681857
Length: 1379, dtype: float64
```

코드 표시

코드 표시

```
0.07400349177836055
```

코드 표시

```
36523.6992104089
```

코드 표시

```
0.3141178872518905
```

코드 표시

```
earn      1.000000
height    0.291600
race     -0.063977
ed        0.350374
age       0.074003
dtype: float64
```

코드 표시

코드 표시

```
0    0.831762
3    0.091371
2    0.055838
1    0.021030
Name: race, dtype: float64
```

(Python 7-2강) pandas II

PPT 필기

↳ 숨겨진 셀 4개

pandas part II - 1

Group by - Basic

```
import pandas as pd
```

```
# data from:
ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
         'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
         'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
         'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
         'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}

df = pd.DataFrame(ipl_data)
df
```

```
#(기준 컬럼)[적용 컬럼]
df.groupby("Team")["Points"].sum()
```

```
Team
Devils     1536
Kings      2285
Riders     3049
Royals     1505
kings       812
Name: Points, dtype: int64
```

```
h_index = df.groupby(["Team", "Year"])["Points"].sum()
h_index
```

```
Team    Year
Devils  2014    863
        2015    673
Kings   2014    741
        2016    756
        2017    788
Riders  2014    876
        2015    789
        2016    694
        2017    690
Royals  2014    701
        2015    804
kings   2015    812
Name: Points, dtype: int64
```

```
h_index.index
```

```
MultiIndex([('Devils', 2014),
            ('Devils', 2015),
            ( 'Kings', 2014),
            ( 'Kings', 2016),
            ( 'Kings', 2017),
            ('Riders', 2014),
            ('Riders', 2015),
            ('Riders', 2016),
            ('Riders', 2017),
            ('Royals', 2014),
            ('Royals', 2015),
            ( 'kings', 2015)],
           names=['Team', 'Year'])
```

```
h_index["Devils":"Kings"]
```

```
Team    Year
Devils  2014    863
        2015    673
Kings   2014    741
        2016    756
        2017    788
Name: Points, dtype: int64
```

```
h_index.unstack()
```

```
# 인덱스 순서 변경
h_index.swaplevel()
```

```
Year  Team
2014  Devils    863
2015  Devils    673
2014  Kings     741
2016  Kings     756
2017  Kings     788
2014  Riders    876
2015  Riders    789
2016  Riders    694
2017  Riders    690
2014  Royals    701
2015  Royals    804
      kings     812
Name: Points, dtype: int64
```

코드 표시

코드 표시

```
Team    Year
Devils  2014    863
        2015    673
Kings   2014    741
        2016    756
        2017    788
Riders  2014    876
        2015    789
        2016    694
        2017    690
Royals  2014    701
        2015    804
kings   2015    812
Name: Points, dtype: int64
```

코드 표시

```
Team    Year
Devils  2014    863
Kings   2014    741
Riders  2014    876
Royals  2014    701
Devils  2015    673
Riders  2015    789
Royals  2015    804
kings   2015    812
Kings   2016    756
Riders  2016    694
Kings   2017    788
Riders  2017    690
Name: Points, dtype: int64
```

코드 표시

```
Year  Team
2014  Devils    863
      Kings     741
      Riders    876
      Royals    701
2015  Devils    673
      Riders    789
      Royals    804
      kings     812
2016  Kings     756
      Riders    694
2017  Kings     788
      Riders    690
Name: Points, dtype: int64
```

코드 표시

```
pandas.core.series.Series
```

코드 표시

```
Team
Devils    134.350288
Kings      24.006943
Riders     88.567771
Royals     72.831998
kings            NaN
Name: Points, dtype: float64
```

코드 표시

```
Year
2014    87.439026
2015    65.035888
2016    43.840620
2017    69.296465
Name: Points, dtype: float64
```

코드 표시

```
Team
```

```
Devils    1536
Kings     2285
Riders    3049
Royals    1505
kings      812
Name: Points, dtype: int64
```

코드 표시

```
Year
2014    3181
2015    3078
2016    1450
2017    1478
Name: Points, dtype: int64
```

Groupby - gropuped

```
df
```

```
grouped = df.groupby("Team")
```

코드 표시

```
Devils
     Team  Rank  Year  Points
2  Devils     2  2014     863
3  Devils     3  2015     673
Kings
    Team  Rank  Year  Points
4  Kings     3  2014     741
6  Kings     1  2016     756
7  Kings     1  2017     788
Riders
```

```
      Team  Rank  Year  Points
0   Riders     1  2014     876
1   Riders     2  2015     789
8   Riders     2  2016     694
11  Riders     2  2017     690
Royals
      Team  Rank  Year  Points
9   Royals     4  2014     701
10  Royals     1  2015     804
kings
     Team  Rank  Year  Points
5  kings     4  2015     812
```

코드 표시

```
pandas.core.groupby.generic.DataFrameGroupBy
```

```
for name,group in grouped:
    print(type(name))
    print(type(group))
```

```
<class 'str'>
<class 'pandas.core.frame.DataFrame'>
<class 'str'>
<class 'pandas.core.frame.DataFrame'>
<class 'str'>
<class 'pandas.core.frame.DataFrame'>
<class 'str'>
<class 'pandas.core.frame.DataFrame'>
<class 'str'>
<class 'pandas.core.frame.DataFrame'>
```

```
grouped.get_group("Riders")
```

Aggregation

```
grouped.get_group('Devils')
```

```
grouped.describe().T
```

```
grouped.agg(min)
```

```
import numpy as np
grouped.agg(np.mean)
```

```
grouped['Points'].agg([np.sum, np.mean, np.std])
```

Transofrmation

- Aggregation과 달리 key값 별로 요약된 정보가 아님
- 개별 데이터의 변환을 지원함

코드 표시

코드 표시

```
# 그룹별 정규화
score = lambda x: (x - x.mean()) / x.std()
grouped.transform(score)
```

filter

- 특정 조건으로 데이터를 검색할 때 사용

```
# 데이터 3개 이상인 값만 출력
df.groupby('Team').filter(lambda x: len(x) >= 3)
```

```
# 그룹의 최대값이 800 이상인 값을 출력
df.groupby('Team').filter(lambda x: x["Points"].max() > 800)
```

pandas part II - 2

```
import pandas as pd
import matplotlib.pyplot as plt
from pandas import Series
from pandas import DataFrame
```

```
!wget https://www.shanelynn.ie/wp-content/uploads/2015/06/phone_data.csv
```

```
--2021-08-06 06:38:43--  https://www.shanelynn.ie/wp-content/uploads/2015/06/phone_data.csv
Resolving www.shanelynn.ie (www.shanelynn.ie)... 104.236.88.249
Connecting to www.shanelynn.ie (www.shanelynn.ie)|104.236.88.249|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 40576 (40K) [text/csv]
Saving to: 'phone_data.csv.2'

phone_data.csv.2    100%[===================>]  39.62K  --.-KB/s    in 0.02s

2021-08-06 06:38:44 (2.16 MB/s) - 'phone_data.csv.2' saved [40576/40576]
```

```
df_phone = pd.read_csv("phone_data.csv")
```

```
df_phone = pd.read_csv('phone_data.csv')
df_phone.head()
```

```
df_phone['date'] = df_phone['date'].astype(str)
```

```
import dateutil
df_phone['date'] = df_phone['date'].apply(dateutil.parser.parse, dayfirst=True)
df_phone.head()
```

```
df_phone.groupby('month')['duration'].sum()
```

```
month
2014-11    26639.441
2014-12    14641.870
2015-01    18223.299
2015-02    15522.299
2015-03    22750.441
Name: duration, dtype: float64
```

```
df_phone.groupby('month')['duration'].sum().plot()
```

```
df_phone[df_phone['item'] == 'call'].groupby('month')['duration'].sum()
```

```
month
2014-11    25547.0
2014-12    13561.0
2015-01    17070.0
2015-02    14416.0
2015-03    21727.0
Name: duration, dtype: float64
```

```
df_phone[df_phone['item'] == 'data'].groupby('month')['duration'].sum().plot()
```

```
df_phone.groupby(['month', 'item'])['duration'].sum()
```

```
month    item
2014-11  call    25547.000
         data      998.441
         sms        94.000
2014-12  call    13561.000
         data     1032.870
         sms        48.000
2015-01  call    17070.000
         data     1067.299
         sms        86.000
2015-02  call    14416.000
         data     1067.299
         sms        39.000
2015-03  call    21727.000
         data      998.441
         sms        25.000
Name: duration, dtype: float64
```

```
df_phone.groupby(['month', 'item'])['date'].count()
```

```
month    item
2014-11  call    107
         data     29
         sms      94
2014-12  call     79
         data     30
         sms      48
2015-01  call     88
         data     31
```

```
         sms       86
2015-02  call      67
         data      31
         sms       39
2015-03  call      47
         data      29
         sms       25
Name: date, dtype: int64
```

```
df_phone.groupby(['month', 'item'])['date'].count().unstack()
```

```
df_phone.groupby(['month', 'item'])['date'].count().unstack().plot()
```

```
df_phone.groupby('month', as_index=False).agg({"duration": "sum"})
```

```
df_phone.groupby(['month', 'item']).agg({'duration':sum,      # find the sum of the durations for eacl
                                         'network_type': "count", # find the number of network type entri
                                         'date': 'first'})    # get the first date per group
```

```
df_phone.groupby(['month', 'item']).agg({'duration': [min],      # find the min, max, and sum of the 
                                         'network_type': "count", # find the number of network type entri
                                         'date': [min, 'first', 'nunique']})    # get the min, first, and
```

```
grouped = df_phone.groupby('month').agg( {"duration" : [min, max, np.mean]})
grouped
```

```
grouped.columns = grouped.columns.droplevel(level=0)
grouped
```

```
grouped.rename(columns={"min": "min_duration", "max": "max_duration", "mean": "mean_duration"})
```

```
grouped = df_phone.groupby('month').agg( {"duration" : [min, max, np.mean]})
grouped
```

```
grouped.columns = grouped.columns.droplevel(level=0)
grouped
```

```
grouped.add_prefix("duration_")
```

```
df_phone
```

pivot table

- 우리가 excel에서 보던 그 것!
- Index 축은 groupby와 동일함
- Column에 추가로 labeling 값을 추가하여,
- Value에 numeric type 값을 aggregation 하는 형태

```
df_phone = pd.read_csv("phone_data.csv")
df_phone['date'] = df_phone['date'].apply(dateutil.parser.parse, dayfirst=True)
df_phone.head()
```

```
df_phone.groupby(['month','item','network'])['duration'].sum().unstack()
```

```
df_phone.pivot_table(["duration"],
                     index=[df_phone.month,df_phone.item],
                     columns=df_phone.network, aggfunc="sum", fill_value=0)
```

crosstab

- 특허 두 칼럼에 교차 빈도, 비율, 덧셈 등을 구할 때 사용
- Pivot table의 특수한 형태
- User-Item Rating Matrix 등을 만들 때 사용가능함

```
!wget https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/codes/ch_3/part-2/da
```

```
--2021-08-06 06:38:18--  https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/code
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
```

```
Length: 1047 (1.0K) [text/plain]
Saving to: 'movie_rating.csv'

movie_rating.csv    100%[===================>]   1.02K  --.-KB/s    in 0s

2021-08-06 06:38:19 (47.5 MB/s) - 'movie_rating.csv' saved [1047/1047]
```

```
df_movie = pd.read_csv("movie_rating.csv")
df_movie.head()
```

```
df_movie.pivot_table(["rating"], index=df_movie.critic, columns=df_movie.title,
                     aggfunc="sum", fill_value=0)
```

```
pd.crosstab(index=df_movie.critic,columns=df_movie.title,values=df_movie.rating,
            aggfunc="first").fillna(0)
```

```
df_movie.groupby(["critic","title"]).agg({"rating":"first"}).unstack().fillna(0)
```

Merge & Concat

- SQL에서 많이 사용하는 Merge와 같은 기능
- 두 개의 데이터를 하나로 합침

```
raw_data = {
        'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'test_score': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}
df_a = pd.DataFrame(raw_data, columns = ['subject_id', 'test_score'])
df_a
```

```
raw_data = {
        'subject_id': ['4', '5', '6', '7', '8'],
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}
df_b = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name', 'last_name'])
df_b
```

```
pd.merge(df_a, df_b, on='subject_id')
```

```
pd.merge(df_a, df_b, left_on='subject_id', right_on='subject_id')
```

```
pd.merge(df_a, df_b, on='subject_id', how='left')
```

```
pd.merge(df_a, df_b, on='subject_id', how='right')
```

```
pd.merge(df_a, df_b, on='subject_id', how='outer')
```

```
pd.merge(df_a, df_b, on='subject_id', how='inner')
```

```
pd.merge(df_a, df_b, right_index=True, left_index=True)
```

```
raw_data = {
        'subject_id': ['1', '2', '3', '4', '5'],
        'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}
df_a = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name', 'last_name'])
df_a
```

```
raw_data = {
        'subject_id': ['4', '5', '6', '7', '8'],
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}
df_b = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name', 'last_name'])
df_b
```

concat

```
df_new = pd.concat([df_a, df_b])
df_new.reset_index()
```

```
df_a.append(df_b)
```

```
df_new = pd.concat([df_a, df_b], axis=1)
df_new.reset_index()
```

```
!wget https://github.com/TEAMLAB-Lecture/AI-python-connect/blob/master/codes/ch_3/part-2/data/sales-f
!wget https://github.com/TEAMLAB-Lecture/AI-python-connect/blob/master/codes/ch_3/part-2/data/sales-j
!wget https://github.com/TEAMLAB-Lecture/AI-python-connect/blob/master/codes/ch_3/part-2/data/sales-m
!wget https://github.com/TEAMLAB-Lecture/AI-python-connect/blob/master/codes/ch_3/part-2/data/custome
```

```
--2021-08-06 06:38:19--  https://github.com/TEAMLAB-Lecture/AI-python-connect/blob/master/codes/ch_3/
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_3/part-2/data/sale
--2021-08-06 06:38:19--  https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_3/p
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/codes/ch_3/part-
--2021-08-06 06:38:19--  https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/c
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133,
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected
HTTP request sent, awaiting response... 200 OK
Length: 10565 (10K) [application/octet-stream]
Saving to: 'sales-feb-2014.xlsx?raw=true'

sales-feb-2014.xlsx 100%[===================>]  10.32K  --.-KB/s    in 0s

2021-08-06 06:38:20 (84.8 MB/s) - 'sales-feb-2014.xlsx?raw=true' saved [10565/10565]

--2021-08-06 06:38:20--  https://github.com/TEAMLAB-Lecture/AI-python-connect/blob/master/codes/ch_3/
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_3/part-2/data/sale
--2021-08-06 06:38:20--  https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_3/p
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/codes/ch_3/part-
--2021-08-06 06:38:20--  https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/c
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133,
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected
HTTP request sent, awaiting response... 200 OK
Length: 11765 (11K) [application/octet-stream]
Saving to: 'sales-jan-2014.xlsx?raw=true'

sales-jan-2014.xlsx 100%[===================>]  11.49K  --.-KB/s    in 0s

2021-08-06 06:38:20 (102 MB/s) - 'sales-jan-2014.xlsx?raw=true' saved [11765/11765]

--2021-08-06 06:38:20--  https://github.com/TEAMLAB-Lecture/AI-python-connect/blob/master/codes/ch_3/
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_3/part-2/data/sale
--2021-08-06 06:38:20--  https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_3/p
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/codes/ch_3/part-
--2021-08-06 06:38:20--  https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/c
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.199.110.133,
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443... connected
HTTP request sent, awaiting response... 200 OK
Length: 12059 (12K) [application/octet-stream]
Saving to: 'sales-mar-2014.xlsx?raw=true'

sales-mar-2014.xlsx 100%[===================>]  11.78K  --.-KB/s    in 0s

2021-08-06 06:38:21 (97.2 MB/s) - 'sales-mar-2014.xlsx?raw=true' saved [12059/12059]
```

```
cat sales-feb-2014.xlsx?raw=true
```


```
PK└�J��E��������������♂���_rels/.rels���J└1┼��}��{w�┴Dd��� ЛH}����a7�0├
┐ox�x┼�v2����◀���₩7
→��└|:xL2�/l����♂crE*├�┬n��p�☼^N`��Y_�h���π���T�aT┐┼�N�P�•h(��F����y�D
```

��i�u��cc���j`.G�B∟��G�W��∟���b?PK�o∟┌���┌��PK└�J��E��������������→
�r▲�3�R�b�+/2•�m��=�?"↔�#ɥ��R6w �_f▲�z♂o P~��$��E����|┐PK0��z�
%�r−�┤�f→!↕����#┘•l��┼bp��,w��"$~┐�s0�|�┌53jP|�($#�]���┼苺�/�u�p�$��Í`s��
W�Bc�M└c�
��K♂@ ├p�┐��EB��@┘^G�k→�1├f淏□ㅌ→d��→�−�−���y�sg�↔!���%+�<��8�K^ /�3���k�
a��☼PK�C[�i|��;┴��PK└�J��E�������������↑���xl/worksheets/sheet1.xml��Ms#��
sC¶�q" �P┴�t�d�↑���4U�z"�m2V�Ɨ �.��+"◀�₩�1i�−%��.Z�C��A7�(m��□]����
�h�X9�┼YR♬b��Aw��┐/��T��↔mp2r�<�↔��∟�−�0�i�~�$��◀��U�↑��♬�#�┤�XU←X┴⟷:8#�
┌��k‼�B→�1����◀P���G��)b��2�$��{┴��lQD7��4��&YE��←+��K��*��+���&g�
λ$�Y"/�h2����#•�G�M���6�Z┐���,�,0┤�@�└�'•�←V����б�ʇH�����¶��F^K�zi�'
��m���F�@♂��L└�' ����J−�k�L��,=s�▲�┐u����∟�♬−�ewȳ�
6��^↑~H�∟┤�,��/�ef��R�┬b0W�|↔�}e�♬��ų�1�̃��GM���L↑]$�0.�jK�gm�ud����#�+q
♬��⁮��H�j���g5��:%0ⅎt←��;��♂���#�B�C4p�!�•���qfZ�?+�a��'�
l=↕{r]��h
Yo�(‼�&�l ┼�U|�
₩'L��M�D�↔�?���ᒡ�Q�,┼
�|
Ob☼n5'�LTp�↕+J��−"A���tbm��A�R�H�┤�?9┴,7D��A9�2���♂;"s
�o#!���tbm�$��m@F�Z_$→H�'���♂�8xQH�j>�;��⁮$�k
���
��‼{�/��k�X��T�↕��ç↔Sm�M�"*����ma�!↕��‼�hT���↕�←o���+��jk└���−8┬�┬`→�
�$��a��←�/�錊▲�,xQ▲5�/lZ��61�−N�!~?g�i�������↕�┴�F�1��↔��
���xl/styles.xml�X]0�0¶}◌��>����) └F�l‼BP�l�▲L�$┬��l┤→~��8l�2�Q&���d���s�
��:Q�
�U!=煮�3�◀x%�Ʊ��w|G┌j┐A.$y2�v┌34+m�−��┬r�@能┬→9┬��Q�ba��D□��y�rl��B�l���T�2�
56�┴�7�x�@▼H�q��N����j`���%|�4`������ql�◀N��gM�#�C����c��}���◀3���
���������������� ��xl/styles.xmlPK┌┐¶�J��E┼��→�┌���┐��☼��������������

```
files = ['customer-status.xlsx',
 'sales-feb-2014.xlsx',
 'sales-jan-2014.xlsx',
 'sales-mar-2014.xlsx']


df_list = [pd.read_excel(str(df_filename)+'?raw=true') for df_filename in files]
status = df_list[0]
sales = pd.concat(df_list[1:])


sales.head()
```

```
merge_df = pd.merge(status, sales, how="inner", on="account number")
merge_df.head()
```

```
merge_df.groupby(["status","name_x"])["quantity","ext price"].sum().reset_index().sort_values(
    by=["status","quantity"], ascending=False)
```

DB Persistence

Load database

```
!wget https://github.com/TEAMLAB-Lecture/AI-python-connect/blob/master/codes/ch_3/part-2/data/flights
```

```
--2021-08-06 06:38:21--  https://github.com/TEAMLAB-Lecture/AI-python-connect/blob/master/codes/ch_3/par
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_3/part-2/data/flights
--2021-08-06 06:38:22--  https://github.com/TEAMLAB-Lecture/AI-python-connect/raw/master/codes/ch_3/part
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/codes/ch_3/part-2/d
--2021-08-06 06:38:22--  https://raw.githubusercontent.com/TEAMLAB-Lecture/AI-python-connect/master/code
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5415936 (5.2M) [application/octet-stream]
Saving to: ‘flights.db?raw=true’

flights.db?raw=true 100%[===================>]   5.17M  --.-KB/s    in 0.1s

2021-08-06 06:38:22 (38.3 MB/s) - ‘flights.db?raw=true’ saved [5415936/5415936]
```

```
mv flights.db?raw=true flights.db
```

```
cat flights.db
```

```
SQLite format 3�┘�┌┌�@  ���       ��¶��������������-���┘����������┌�
[index] INTEGER,
  [airline] TEXT,
  [airline_id] TEXT,
  [source] TEXT,
  [source_id] TEXT,
  [dest] TEXT,
  [dest_id] TEXT,
  [codeshare] TEXT,
  [stops] TEXT,
  [equipment] TEXT
)Z┘─┤/↔┐uindexix_airlines_indexairlines└�CREATE INDEX ix_airlines_index ON airlines ([index])�G└●┤↔
[index] INTEGER,
  [id] TEXT,
  [name] TEXT,
  [alias] TEXT,
  [iata] TEXT,
  [icao] TEXT,
  [callsign] TEXT,
  [country] TEXT,
  [active] TEXT
┌●┤↔↔┌�mtableairportsairports┐CREATE TABLE airports (
[index] INTEGER,
  [id] TEXT,
  [name] TEXT,
  [city] TEXT,
  [country] TEXT,
  [code] TEXT,
  [icao] TEXT,
  [latitude] TEXT,
```

```
    [longitude] TEXT,
    [altitude] TEXT,
    [offset] TEXT,
    [dst] TEXT,
    [timezone] TEXT
  )|����•└�����└�└�└�└�└�└�└�└�└�└�└�└�└�└�└�└�└�└�└|└v└p└j└d└
�����└����x���~�m���}�b���|�W���{�M���z�C���y�:���x�0����w�&
���-�└���,�y���+�o���*�e���)�Z���'�P���&�E���%�;���$�0����#�%�
┐���┌└����┘└�└�└�└�└�└�└�└�└�└~└r└f└Z└N└B└6└*└▲└↕└─┐�┐�┐�┐�┐�┐�┐�┐�
♬��┌J•└┐┐♂�♂���┌@•└┐┐♂Y♂Z��┌5•└┐┐
�♂���┌+•└┐┐
�
���┌▼•└┐┐
K
L��┌┴•└┐┐    �      ���┌
���
♬┌◀%┴←‼┴▼!◀☼☼113HornafjordurHofnIcelandHFNBIHN64.295556-15.227222240NAtlantic/Reykjavik^♬┌◀##←‼┴▼!◀☼☼
11AkureyriAkureyriIcelandAEYBIAR65.659994-18.07270360NAtlantic/ReykjavikZ
���
���
���
���
���
���
���
���
���
```


```python
import sqlite3 #pymysql <- 설치

conn = sqlite3.connect("flights.db")
cur = conn.cursor()
cur.execute("select * from airlines limit 5;")
results = cur.fetchall()
results
```

```
[(0, '1', 'Private flight', '₩₩N', '-', None, None, None, 'Y'),
 (1, '2', '135 Airways', '₩₩N', None, 'GNL', 'GENERAL', 'United States', 'N'),
 (2, '3', '1Time Airline', '₩₩N', '1T', 'RNX', 'NEXTIME', 'South Africa', 'Y'),
 (3,
  '4',
  '2 Sqn No 1 Elementary Flying Training School',
  '₩₩N',
  None,
  'WYT',
  None,
  'United Kingdom',
  'N'),
 (4, '5', '213 Flight Unit', '₩₩N', None, 'TFU', None, 'Russia', 'N')]
```

```python
df_airplines = pd.read_sql_query("select * from airlines;", conn)
df_airplines.head()
```

```
df_airports = pd.read_sql_query("select * from airports;", conn)
df_routes = pd.read_sql_query("select * from routes;", conn)
```

```
df_airports.head()
```

```
df_routes.head()
```

Pandas persistence

```
!pip install openpyxl
!pip install XlsxWriter
```

```
Requirement already satisfied: openpyxl in /usr/local/lib/python3.7/dist-packages (2.5.9)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.7/dist-packages (from openpyxl) (1.1
Requirement already satisfied: jdcal in /usr/local/lib/python3.7/dist-packages (from openpyxl) (1.4.1)
Collecting XlsxWriter
  Downloading XlsxWriter-1.4.5-py2.py3-none-any.whl (149 kB)
     |████████████████████████████████| 149 kB 5.0 MB/s
```

```
Installing collected packages: XlsxWriter
Successfully installed XlsxWriter-1.4.5
```

```python
# 엑셀로 저장
writer = pd.ExcelWriter('df_routes.xlsx', engine='xlsxwriter')
df_routes.to_excel(writer, sheet_name='Sheet1')
```

```python
df_routes.to_pickle("df_routes.pickle")
```

```python
df_routes_pickle = pd.read_pickle("df_routes.pickle")
```

```python
df_routes_pickle.describe()
```