파이썬 활용 (#2/5)

2023. 02. 21

초빙교수 김현용
충북대학교 산업인공지능연구센터

CHUNGBUK NATIONAL UNIVERSITY

INDUSTRIAL AI RESEARCH CENTER

강의 일정

INDUSTRIAL AI RESEARCH CENTER

	1일차	2일차	3일차	4일차	5일차
주제	파이참과 엑셀파일	데이터 분석	웹 자동화	컴퓨터 비전	AI와 딥러닝
세부 내용	• 강사/강의 소개 • **PyCharm** • [실습] script 작성 • 가상환경 • 파일 경로 • 파일 입출력 • [실습] 파일 입출력 • **OpenPyXL** - 엑셀파일 입출력 - 셀 편집 - 차트 그리기 • [실습] 셀서식 지정 • [실습] 차트 그리기	• 데이터분석 • **NumPy** • 회귀분석 • [실습] 선형회귀 • **Pandas** • [실습] 데이터프레임 다루기 • **Matplotlib** • [실습] 시각화 • [실습] 회귀분석	• **웹스크래핑** • [실습] 기온 가져오기 • **Open API** (파파고) • **Selenium** • [실습] 구글 검색 • **PyAutoGui** • [실습] 구글 검색	• 컴퓨터 비전 개요 • **OpenCV** - 영상 입출력 - 영상 데이터 조작 • Matplotlib 영상표시 • OpenCV - 그리기 함수 - 동영상 입출력 • OpenCV - 히스토그램 - 이진화 - 에지 검출	• 인공신경망 • 딥러닝 • **PyTorch** • 객체검출 • **YOLOv8-객체검출** • [실습] 객체검출 • YOLOv8-객체분할 • chatGPT

NumPy (#1/15)

▣ Numpy(Numerical Python)

NumPy https://numpy.org/

- C언어로 구현된 고성능 과학계산용 라이브러리
- **다차원 배열(ndarry, n-dimensional array)**이라는 자료구조를 통해 **행렬연산**에 특화된 라이브러리
  - 파이썬의 리스트 대비 **효율적인 메모리 사용**
  - BLAS(basic linear algebra subprogram) 등의 수치연산 라이브러리를 사용하여 **초고속 벡터 연산** 수행

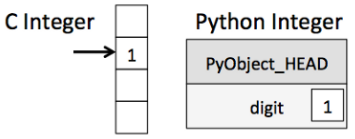
  예) [10000x1000] x [1000x10000] 행렬곱의 연산속도 비교 :

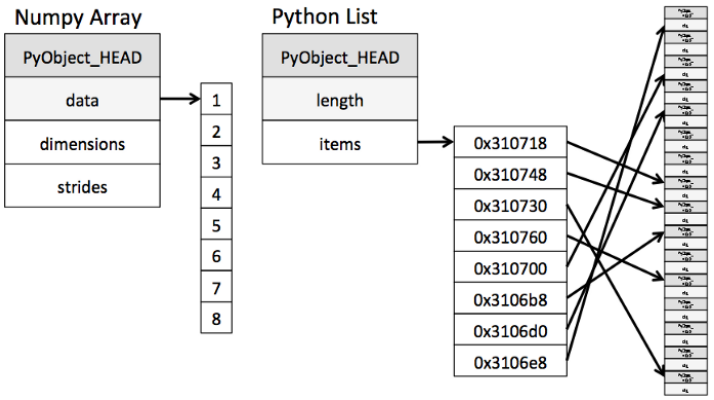
언어	C	파이썬	파이썬+Numpy
소요시간	409초	120,000초	5초

- 설치 : pip install numpy → import numpy as np

▣ 파이썬의 **객체모델**(PyObject)

- "Everything is object in Python"
- 파이썬에는 기본형(primitive type) 자료형이 없고 모두 **참조형(refercence type) 자료형**





NumPy (#2/15)

▣ 파이썬 리스트(list)와 넘파이 배열(array)의 차이

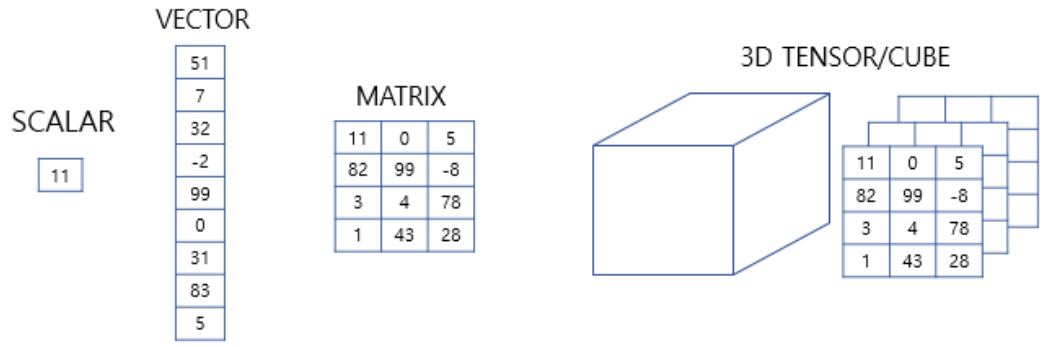
항목	파이썬 리스트	넘파이 배열
데이터형	**다양한** 데이터형이 가능 → **편리성, 유연성**	**동일한** 데이터형만 가능 → **효율성**, 벡터연산
메모리	객체모델로 많은 용량 사용 메모리 **단편화** → 메모리 로딩 overhead 발생 (폰 노이만 병목)	적은 용량, 효율적인 접근 메모리에 연속적으로 저장 → 한번에 블럭 로딩 가능 (denslely packed in memory)
연산방식	**자료구조**로써 수치연산과 다름	벡터, 행렬 등의 **수치연산**과 동일함 **병렬연산** 수행
인덱싱	list[0][1]	array[0][1], array[0, 1] (속도가 조금 더 빠름)

```
import numpy as np       # pip install numpy

# 리스트 자료형의 연산
li = [1, 2, ' a ' ]                    # 다양한 자료형을 포함
print(li * 2)                          # 곱셈 : [1, 2, ' a ', 1, 2, ' a ']
print(li + li)                         # 덧셈 : [1, 2, ' a ', 1, 2, ' a']

# 넘파이 배열의 연산 → 수치연산과 동일
arr = np.array([1, 2, 3])     # 배열 : 동일한 자료형을 포함
print(arr * 2)                         # 곱셈 : [2 4 6]
print(arr + arr)                       # 덧셈 : [2 4 6]
print(arr * arr)                       # 곱셈 : [1 4 9]
```

▣ 스칼라(scalar) / 벡터(vector) / 행렬(matrix) / 텐서(tensor)

- rank : 텐서를 구성하는 독립적인 축의 개수

Rank	명칭	예시	접근
0	scalar	a = 11	a
1	vector	a = [51, 7, 32, …, 5]	a[1]
2	matrix	a = [[11, 0, 5] , [82, 99, -8] , … [1, 43, 28]]	a[1]
3	3-tensor	a = [[[11, 0, 5], [82, 99, -8], … [1, 43, 28]], [[…], …, […]], …]	a[1]
n	n-tensor	-	



[참고] 행렬연산

▣ 행렬의 덧셈과 뺄셈

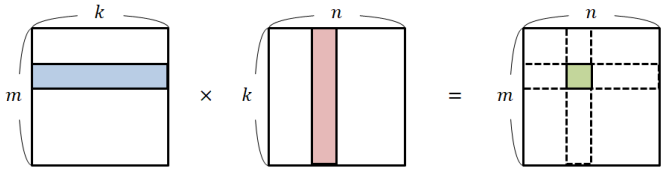
$$a = \begin{bmatrix} 10\ 20\ 30\ 40 \\ 50\ 60\ 70\ 80 \end{bmatrix} \quad b = \begin{bmatrix} 5\ 6\ 7\ 8 \\ 1\ 2\ 3\ 4 \end{bmatrix}$$

$$a + b = \begin{bmatrix} 10\ 20\ 30\ 40 \\ 50\ 60\ 70\ 80 \end{bmatrix} + \begin{bmatrix} 5\ 6\ 7\ 8 \\ 1\ 2\ 3\ 4 \end{bmatrix} = \begin{bmatrix} 15\ 26\ 37\ 48 \\ 51\ 62\ 73\ 84 \end{bmatrix}$$

$$a - b = \begin{bmatrix} 10\ 20\ 30\ 40 \\ 50\ 60\ 70\ 80 \end{bmatrix} - \begin{bmatrix} 5\ 6\ 7\ 8 \\ 1\ 2\ 3\ 4 \end{bmatrix} = \begin{bmatrix} 5\ 14\ 23\ 32 \\ 49\ 58\ 67\ 76 \end{bmatrix}$$

▣ 행렬의 곱셈

- 계산된 행렬의 각 원소값은 해당 행과 열 벡터의 내적 (inner product)

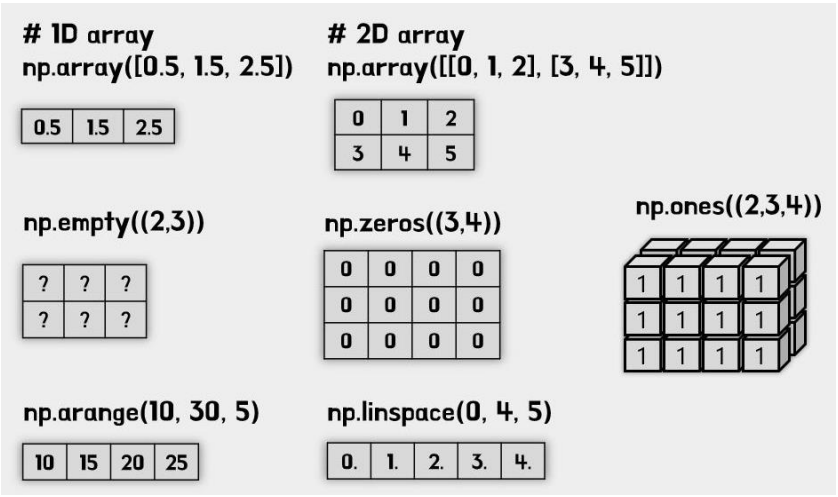


$$a = \begin{bmatrix} 1\ 3 \\ 2\ 4 \end{bmatrix} \quad b = \begin{bmatrix} 5\ 7 \\ 6\ 8 \end{bmatrix}$$

$$ab = \begin{bmatrix} 1\ 3 \\ 2\ 4 \end{bmatrix} \begin{bmatrix} 5\ 7 \\ 6\ 8 \end{bmatrix} = \begin{bmatrix} 1\times5+3\times6 & 1\times7+3\times8 \\ 2\times5+4\times6 & 2\times7+4\times8 \end{bmatrix} = \begin{bmatrix} 23\ 31 \\ 34\ 46 \end{bmatrix}$$

▣ 다차원 배열(ndarray)의 생성

- np.**array**(p_object [, dtype]) : 리스트, 튜플을 넘파이 배열로 변환. dtype 생략 시에는 float64 또는 int64
- np.**empty/zeros/ones/eye**(shape [,dtype, like]) : 각 요소가 임의의 수(쓰레기)/0/1로 구성된 배열 생성
- np.**full**(shape, value [,dtype, like]) : 각 요소가 value로 채워진 배열을 생성
- np.empty/zeros/ones/eye/full**_like**(a [,dtype]) : 주어진 배열 a와 모양과 데이터형이 같은 배열을 반환
- np.**arange**([start,] stop[, step,]) : 주어진 구간에서 균일한 간격의 숫자(실수)를 만듦. **range**()는 정수만 생성
- np.**linspace**(start, stop, num) : [start, end]를 num 개로 균등하게 분포하는 데이터 생성

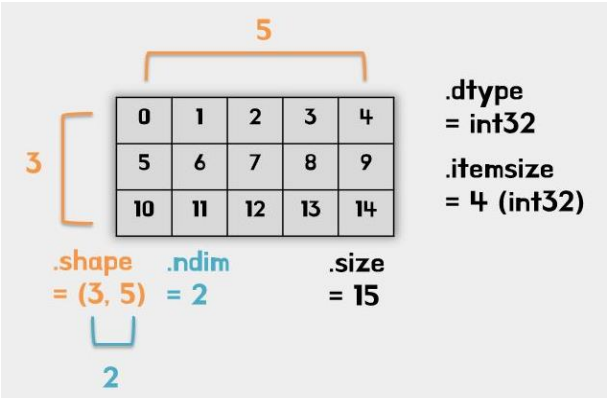


구분	dtype	파이썬
부호있는 정수형	np.int8 np.int16 np.int32 np.int64	int
부호없는 정수형	np.uint8 np.uint16 np.uint32 np.uint64	-
부동 소수형	np.float16 np.float32 np.float64 np.float128	float

dtype=np.int8, 'int8', int

▣ 넘파이 배열의 속성

- a.**shape** : 각 차원(축)의 크기(튜플)
- a.**ndim** : 차원(축)의 수
- a.**dtype** : 요소의 자료형
- a.**itemsize** : 하나의 요소 길이(bytes)
- a.**size** : 배열의 요소 수
- a.**T** : 전치된 배열




```
li = [[1, 2, 3], [4, 5, 6]]      # list
arr = np.array(li)               # array

arr[0][0]    # 1
arr[1, 1]    # 5

arr.shape    # (nrow, ncol) = (2, 3)
arr.ndim     # num of axis(rank) = 2
arr.size     # num of elements = 6
```

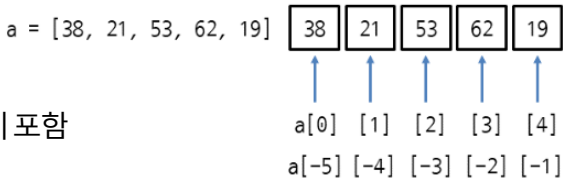
```
arr.reshape(3, 2)  # (-1, 2) 또는 (3, -1)
# array([[1, 2],
#        [3, 4],
#        [5, 6]])
np.arange(3)       # array([0, 1, 2])
np.arange(3.0)     # array([ 0.,  1.,  2.])
np.arange(3, 7)    # array([3, 4, 5, 6])
np.arange(3, 7, 2) # array([3, 5])
np.linspace(1, 5, 5) # array([1, 2, 3, 4, 5])
```

NumPy (#6/15)

INDUSTRIAL AI RESEARCH CENTER

▣ 인덱싱(indexing), 슬라이싱(slicing)

- 음수(-1, -2, ...)는 끝에서 역순으로 가리킴
- 슬라이싱 : arr[start:end:stride]일 경우 start ~ end-1까지 포함
- 콜론(:)은 전체, ...은 차원 생략
- 함수인자에 들어가는 차원값 -1은 차원을 자동으로 추정할 때 사용




```
arr = np.arange(20).reshape(4, 5)    # 배열생성
# array([[ 0,  1,  2,  3,  4],
#        [ 5,  6,  7,  8,  9],
#        [10, 11, 12, 13, 14],
#        [15, 16, 17, 18, 19]])

# indexing : slicing[start:end:stride]
arr[:]            # arr (주소 참조), arr[:] (복사본 할당)
arr[1]            # array([5, 6, 7, 8, 9])
arr[1, :]         # arr[1] == arr[1, :]
arr[-3:, :3]
# array([[ 5,  6,  7],
#        [10, 11, 12],
#        [15, 16, 17]])

arr[[1, 3], -2]    # array([ 8, 18])
```

```
arr[:, :4:2]
# array([[ 0,  2],
#        [ 5,  7],
#        [10, 12],
#        [15, 17]])

arr[::-1, 2]
# ::-1(역순) -> array([17, 12,  7,  2])
# 차원 생략(...), 자동 추정(-1)

arr[1, ...]
# arr[1, :, :] -> array([5, 6, 7, 8, 9])

arr.reshape(-1, 10).shape
# (2, 10) : -1은 자동적으로 2로 정해짐
```


NumPy (#7/15)

INDUSTRIAL AI RESEARCH CENTER

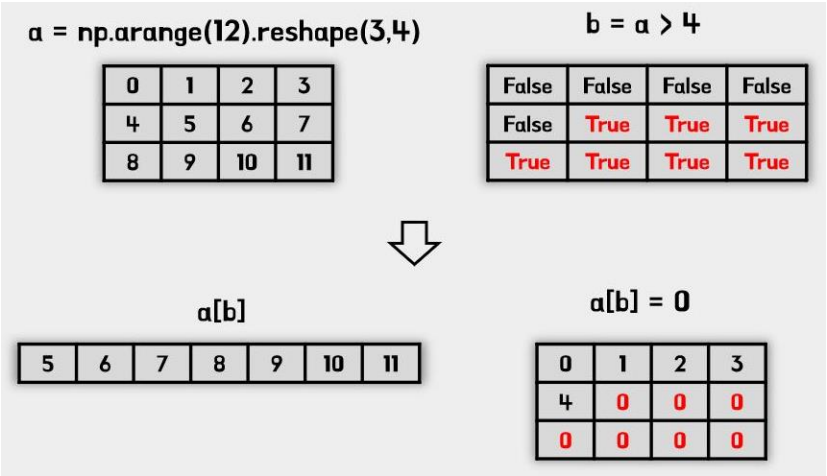
▣ 조건식을 이용한 인덱싱

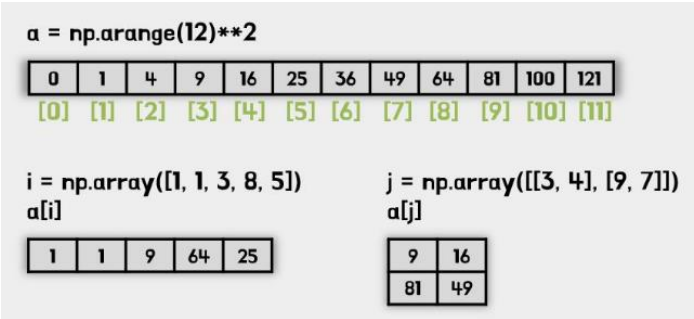
- arr[(arr > 4) | ~(arr % 2 == 0)] = 0

```
arr = np.arange(20).reshape(4, 5)
# array([[ 0,  1,  2,  3,  4],
#        [ 5,  6,  7,  8,  9],
#        [10, 11, 12, 13, 14],
#        [15, 16, 17, 18, 19]])

arr % 3 == 0
# array([[ True, False, False,  True, False],
#        [False,  True, False, False,  True],
#        [False, False,  True, False, False],
#        [ True, False, False,  True, False]])

arr[arr % 3 == 0]
# array([ 0,  3,  6,  9, 12, 15, 18])
arr[(arr % 4 == 0) | ~(arr <= 15)]
# array([ 0,  4,  8, 12, 16, 17, 18, 19])
```

▣ 정수 배열을 이용한 인덱싱

- arr1[arr2]



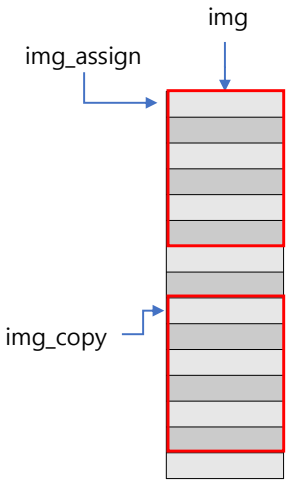


NumPy (#8/15)

INDUSTRIAL AI RESEARCH CENTER

▣ 얕은(shallow) 복사와 깊은(deep) 복사

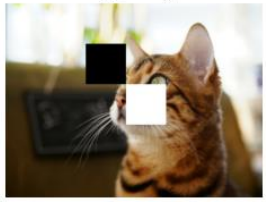
- 얕은 복사 : 원본 객체의 **참조(reference, 메모리 주소)만 복사**하는 것
- 깊은 복사 : 원본 객체의 **데이터까지 완전히 복사**하는 것
  - 파이썬에는 copy(얕은 복사)와 deepcopy가 있으나 넘파이에서는 copy()만으로 깊은 복사가 됨



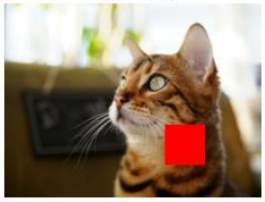

```
img = cv2.imread('cat.bmp')

img_assign = img          # shallow copy (reference) : 이름만 다른 같은 변수
img_copy = img.copy()     # deep copy : 완전히 다른 변수

id(img)          # 1557621973392 (원본)
id(img_assign)   # 1557621973392 (원본과 동일)
id(img_copy)     # 1557606882192

img[100:200, 200:300] = 0                  # black
img_assign[200:300, 300:400] = 255         # white
img_copy[300:400, 400:500] = [0, 0, 255]   # red
```

img

img_assign



img_copy

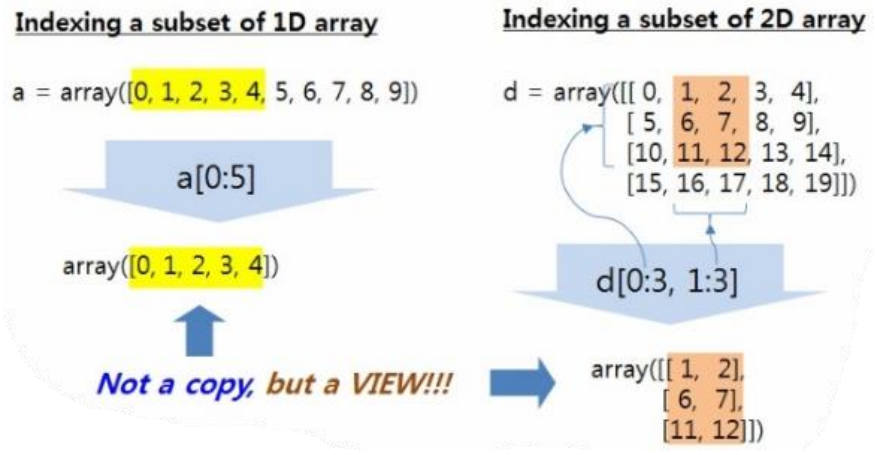


NumPy (#9/15)

INDUSTRIAL AI RESEARCH CENTER

▣ 인덱싱과 **뷰(view)**

- 기본(basic) 인덱싱 : 인덱싱, 슬라이싱 → **view(참조)**를 생성 vs. 파이썬에서는 사본이 만들어 짐
  - 원래 배열의 일부분이나 전체를 참조하는 새로운 배열 객체를 생성하는 것
  - view를 생성하면 메모리를 공유하므로 **메모리를 절약**할 수 있으나 뷰를 통해 수정한 내용은 원래 배열에도 반영됨
- 응용(fancy) 인덱싱 : 조건(bool), 정수 배열 인덱싱 → **사본(deep copy)**이 생성

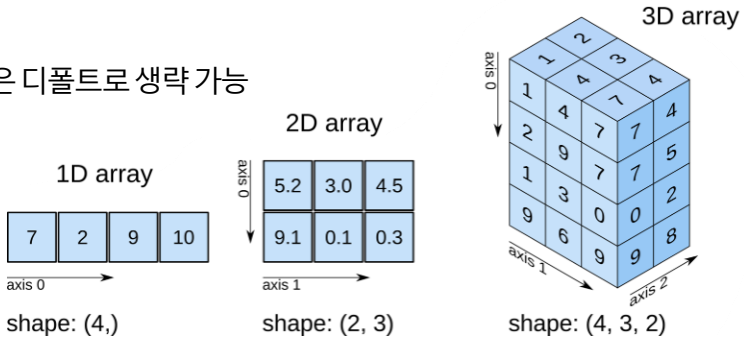



```
arr = np.array([[1, 2, 3], [4, 5, 6]])

view = arr[:, :2]  # arr의 뷰 생성
print(view)
# 출력: array([[1, 2],
#              [4, 5]])

view[0, 0] = 10  # 뷰의 요소를 수정
print(arr)
# 출력: array([[10,  2,  3],
#              [ 4,  5,  6]])
```

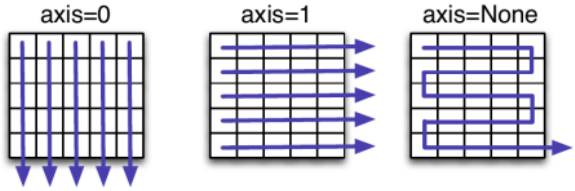
INDUSTRIAL AI RESEARCH CENTER

▣ 축(axis) 이해하기

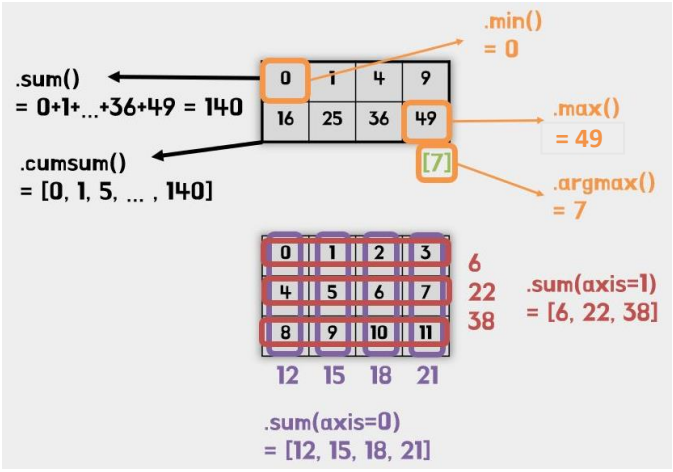
- 차원별 축: axis=0은 디폴트로 생략 가능



- 2D 배열(행렬)의 축




```
arr = np.array([[1, 2, 3], [4, 5, 6]])
arr.max()              # 6
arr.min(axis=0)        # array([1, 2, 3])
arr.max(axis=1)        # array([3, 6])
arr.mean()             # 3.5
```



INDUSTRIAL AI RESEARCH CENTER

▣ 넘파이 배열의 사칙연산 : 기본적으로 **요소별(elementwise)** 연산임 **(vectorization)**

- dtype이 다르면 자동적으로 큰 자료형으로 변경됨
- np.add()
- np.substract()
- np.multiply()
- np.divide()
- np.sum() / product()

```
arr1 = np.array([[5, 10, 15], [20, 25, 25]])
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
arr1 + arr2            # array([[ 6, 12, 18], [24, 30, 31]])
arr1 - arr2            # array([[ 4,  8, 12], [16, 20, 19]])
arr1 * arr2            # array([[  5,  20,  45], [ 80, 125, 150]])
arr1 / arr2            # array([[5.        , 5.        , 5.        ],
                       #        [5.        , 5.        , 4.16666667]])
arr2 ** 2              # array([[ 1,  4,  9], [16, 25, 36]], dtype=int32)
np.sqrt(arr1)          # array([[2.23606798, 3.16227766, 3.87298335],
                       #        [4.47213595, 5.        , 5.        ]])
```

▣ 행렬의 */내적/곱셈

- * : 요소곱
- np.**dot**(arr1, arr2), arr1.**dot**(arr2), arr1 @ arr2 : 행렬간 곱셈, 벡터간 내적, 스칼라간 곱
- **transpose**() : 행렬의 전치행렬 계산 예) arr.transpose() 또는 arr.T

```
arr1 = np.array([[1, 3], [2, 4]])
arr2 = np.array(([5, 7], [6, 8]))
np.dot(arr1, arr2)     # array([[23, 31], [34, 46]])
arr1.transpose()       # array([[1, 2], [3, 4]])
arr1.T
```

INDUSTRIAL AI RESEARCH CENTER

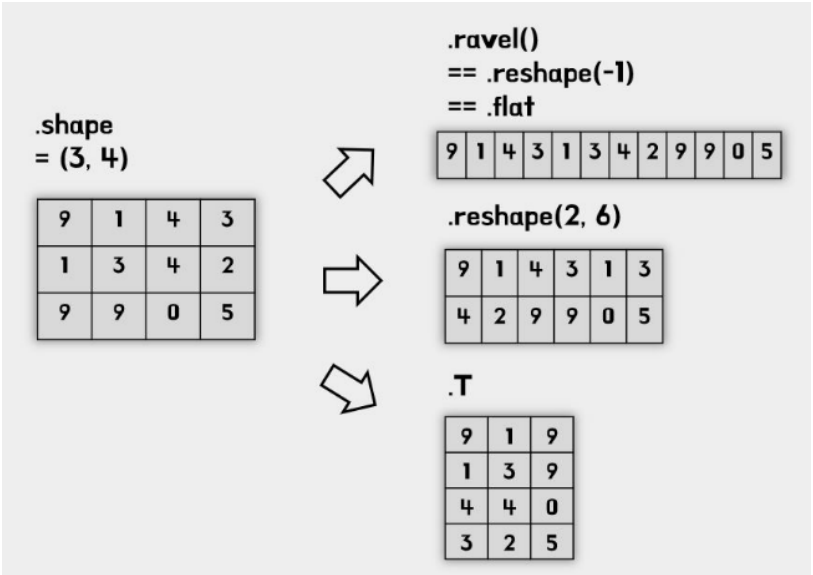
▣ 배열의 메서드(methods) & 넘파이의 함수(routines) → arr.max() or np.max(arr)

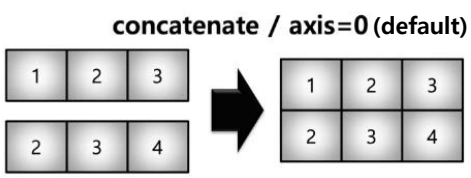
구분	메서드	설명
배열 변환	a.tolist()	파이썬 다차원 리스트로 변환
	a.astype(dtype)	배열의 데이터형을 변경 (casting)
	a.**copy**() vs a.**view**([dtype])	**깊은 복사(deep copy) vs. 얕은 복사 : arr2=arr1**
	a.fill(value)	배열을 특정 값으로 채움
shape 조작	a.**reshape**(shape) / a.resize(shape)	배열의 shape 변경, resize는 in-place
	a.ravel() / a.flatten()	다차원 배열을 1차원으로 변환
	a.transpose() / a.T	전치된 배열을 반환
	a.squeeze([axis])	길이가 1인 축을 제거시킴
	np.concatenate((a1, a2, …), axis)	배열을 axis 방향으로 결합시킴 (axis=0: 수직)
	np.hstack / vstack((a1, a2, …))	배열을 수평/수직 방향으로 결합시킴
계산 및 집계	a.sort()	배열을 정렬시킴
	a.max/min([axis])	축방향으로 최대값을 반환
	a.argmax/min([axis])	축방향으로 최대값의 인덱스를 반환
	a.sum/mean/var/std([axis])	축방향의 합계/평균/분산/표준편차를 반환
	a.round([decimals, out])	소수점 수에 맞춰 모든 요소를 반올림
	a.all/any([axis, out, keepdims, where])	all/any 요소가 True이면 True를 반환
	a.clip([min, max, out])	배열의 최소값, 최대값을 지정

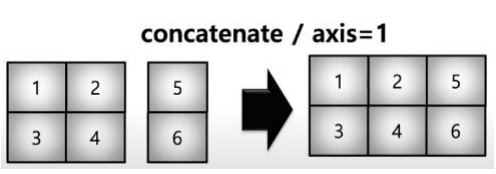
INDUSTRIAL AI RESEARCH CENTER

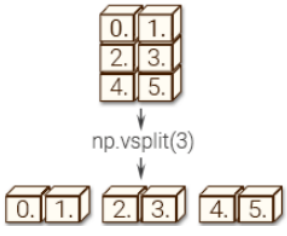
▣ Shape 조작 함수

- arr.reshape(shape)
- arr.resize(shape)
  - shape는 2, 3 or (2, 3) 가능
- arr.transpose() / .T
- arr.ravel()
- arr.flatten()
- np.concatenate((a1, a2), axis=0)
- np.hstack / vstack((a1, a2, …))
- arr.split(num, axis=0)
- arr.hsplit / vsplit(num)
- np.flip(a, axis=0) / .fliplr / flipud
- np.roll(a, shift, axis=0)
- np.rot90(a, k, axis=0)









NumPy (#13/15)

INDUSTRIAL AI RESEARCH CENTER

▣ 난수(random number)

- np.random.**seed**(0) : 랜덤 seed 지정
- np.random.**rand**(*dn) : [0, 1] 실수의 균등분포, *dn은 d0, d1, … *예) rand(2, 3)*
- np.random.**randint**(low, high=None, size=None, dtype=None) : 정수 난수 *예) randint(1, 6, (2, 3))*
- np.random.**randn**(*dn) : 표준정규분포 *예) 평균3, 표준편차2.5인 정규분포로 난수 발생: 3 + 2.5 * randn(2, 4)*

▣ 데이터 샘플링

- np.random.**choice**(a, size=None, replace=True, p=None) : 1D 배열에서 샘플링

a	배열, 혹은 정수. 정수를 입력할 경우 range(a) 명령으로 데이터 생성
size	샘플할 개수 지정
replace	True이면 한 번 선택한 데이터를 다시 추출 가능 (중복 추출)
p	a 배열의 데이터들에 대해 선택될 수 있는 확률 지정

- np.random.**shuffle**(x) : 배열 x의 순서를 무작위로 변경 (in place)

```
x = np.random.choice(10, 10, replace=False)      # shffle과 동일
# array([2, 8, 4, 9, 1, 6, 7, 3, 5, 6])
np.random.choice(x, 5, replace=True)             # 5개 샘플링, 중복 허용
# array([0, 8, 6, 5, 0])
np.random.choice(4, 3, replace=False, p=[0.4, 0.2, 0, 0.4])   # 선택확률 지정
# array([0, 3, 1])
```


NumPy (#14/15)

INDUSTRIAL AI RESEARCH CENTER

▣ 유니버설 함수(ufunc, universal functions)

- 배열의 원소별로 연산을 수행할 때, for 구문 없이 일괄적으로 처리하는 함수

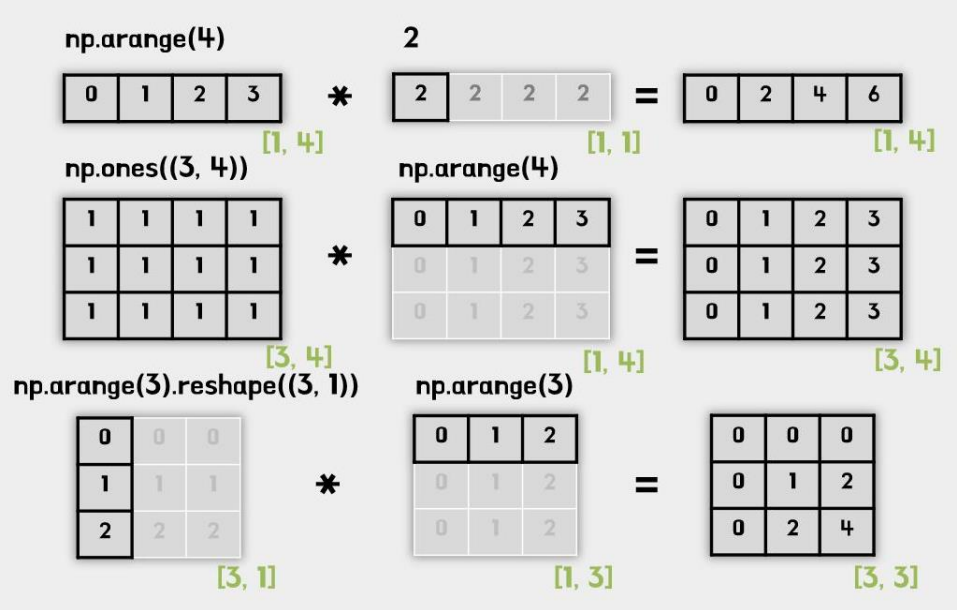
→ **벡터(vectorized) 연산**

- 연산속도가 수십~수백 배 빠름

```
np.sqrt(x)              # 제곱근 연산 √x
np.exp(x)               # 지수함수 연산 e^x (e=2.718281…)
np.abs(x)               # 절대값 연산
np.ceil(x)              # 소수점 올림 연산
np.floor(x)             # 소수점 내림 연산
np.round(x)             # 소수점 반올림 연산
np.cos(x), np.sin(x)    # sin함수, cos함수 처리 (그 외 다양한 삼각함수들 존재)
np.power(x, n)          # n 제곱 처리
np.log(x)               # 자연로그 연산 (log_e x)
np.log10(x)             # 로그10 연산  (log_10 x)
np.log2(x)              # 로그2 연산   (log_2 x)
np.mod(x, n)            # 각 요소별로 n으로 나눈 뒤의 나머지 추출
```

▣ 브로드캐스팅(broadcasting, 확장) : 연산의 편의성

- shape이 맞지않아 연산이 불가능할 때, 배열의 크기를 확장하여 자동으로 shape을 맞춰주는 것
- 성립 조건 : 최소한 하나의 배열의 차원의 크기가 1일 것, 차원의 짝이 일치할 것



▣ Pandas (**pip install pandas → import pandas as pd**) https://pandas.pydata.org

- 빅데이터(3V)의 시대

  최근 들어 분석할 데이터의 양(Volume)이 커지고, 데이터의 입출력 속도(Velocity)가 빨라지고,

  데이터의 종류가 다양해(Variety)지면서 데이터 분석이 어렵고 중요해짐

- Numpy 기반으로 개발된 모듈로, **대용량 데이터를 빠르고 쉽게 다룰 수 있음 (엑셀 ~100MB, 판다스 ~100GB)**
- 고성능의 다양한 데이터 분석 도구와 Series, **DataFrame**이라는 강력하고 편리한 데이터 구조를 제공

▣ Series와 DataFrame (from pandas import Series, DataFrame)

- **Series** : 인덱스 라벨이 붙은 1차원 데이터 구조로 리스트와 달리, 값과 함께 인덱스가 추가된 구조
- **DataFrame** : 행과 열로 구성된 2차원 테이블 형식의 데이터 구조

Series

index	
A	3
B	8
C	0
D	2
E	1

DataFrame

index \ column	가	나	다	라
A	3	1	7	0
B	8	2	8	0
C	0	3	9	0
D	2	4	0	1
E	1	5	1	0

- 행과 열을 label로 접근 가능
- 각 열은 자료형이 다를 수 있음
- 데이터베이스와 쉽게 연동
- DataFrame끼리 Join 연산 가능

Pandas (#2/6) - Series

▣ Series 생성 : **pd.Series(values, index=index)**

- values=리스트 예) Series([40, 36, 15, 11]) # index를 지정하지 않으면 디폴트로 0, 1, 2 ... 정수값
- values, index=리스트 예) s = Series([40, 36, 15, 11], index=['아빠', '엄마', '딸', '아들'])
- values=딕셔너리 : key → index 예) Series({'아빠': 40, '엄마': 36, '딸': 15, '아들': 11})

```
In[17]: s
Out[17]:
아빠    40
엄마    36
딸      15
아들    11
dtype: int64
```

```
In[18]: for index in s.index:
   ...:     print(index)
   ...:
아빠
엄마
딸
아들
```

▣ Series 접근

- 속성 : **s.values, s.index, s.shape** # (4,)
- 자료형 정보 : s.info() 등 dir(s)로 확인
- 요소 접근 : s['아빠'] or s[0] = 50, s.딸
- 인덱스 슬라이싱 : s['아빠':'딸'] or s[:3]
- 조건식을 이용한 인덱싱 : s[s>20]

```
In[70]: s['아빠':'딸']
Out[70]:
아빠    50
엄마    36
딸      15
Name: 나이, dtype: int64
```

```
In[71]: s[:3]
Out[71]:
아빠    50
엄마    36
딸      15
Name: 나이, dtype: int64
```

```
In[95]: s[s>20]
Out[95]:
아빠    50
엄마    36
Name: 나이, dtype: int64
```


Pandas (#3/6) - DataFrame

▣ DataFrame 생성 : **pd.DataFrame(data, index=index, columns=cols)**

- data=리스트 : DataFrame([[40, '남', '회사원'], [36, '여', '주부'], [15, '여', '학생'], [11, '남', '학생']],
  index=['아빠', '엄마', '딸', '아들'], columns=['나이', '성별', '직업'])
- data=딕셔너리 : key → column명

```
df = DataFrame({'나이': [40, 36, 15, 11], '성별': ['남', '여', '여', '남'],
                '직업': ['회사원', '주부', '학생', '학생']},
               index=['아빠', '엄마', '딸', '아들'])
```

	나이	성별	직업
아빠	40	남	회사원
엄마	36	여	주부
딸	15	여	학생
아들	11	남	학생

▣ DataFrame 속성/메서드

- 속성 : values, index, columns, shape, dtypes : objec는 범주형 데이터
- info() : 데이터의 구성, 타입, 수량 등 전반적인 정보 표시
- describe() : 수치 데이터에 대한 통계량 출력
- head(n=5), tail(n=5) : top5, bottom 5 행을 표시
- isnull().sum() : 결측치 개수 확인

```
In[51]: df.values
Out[51]:
array([[40, '남', '회사원'],
       [36, '여', '주부'],
       [15, '여', '학생'],
       [11, '남', '학생']], dtype=object)
```

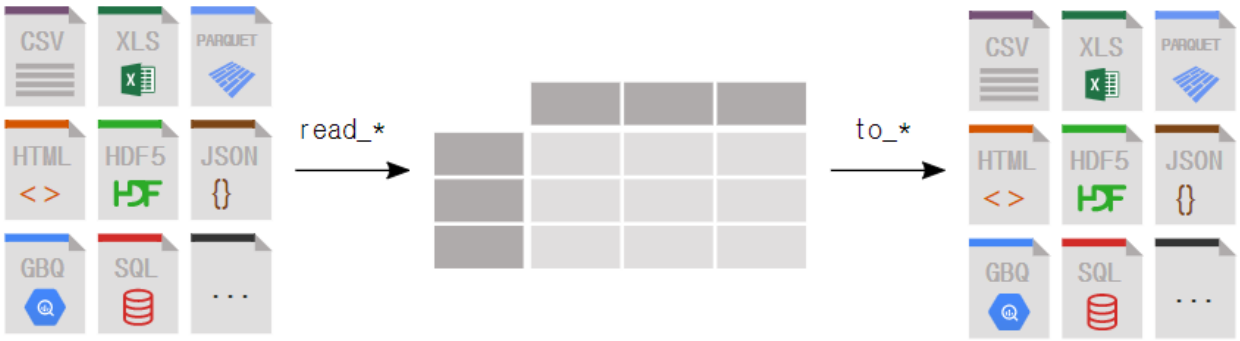
Pandas (#4/6) – 파일 불러오기

INDUSTRIAL AI RESEARCH CENTER

▣ csv 파일 불러오기 : **pd.read_csv('파일명.csv',** sep, index_col, header , name, encoding, skiprows, na_values, ...)

- sep=' , ' : 구분자(, \t |) 지정
- index_col=0 / 'name': 인덱스로 사용할 컬럼의 번호(0)나 컬럼명('name')을 지정
- header=0 / None : 1번째 행을 컬럼명으로 사용할 경우 0(default) / 컬럼명이 없으면 None
- names=['나이', '성별', ...] : 파일에 컬럼명이 없는 경우에 추가
- encoding='utf-8' / 'cp949' : 한글파일을 불러올 때 UnicodeDecodeError가 발생하면 둘 중 하나를 적용
- skiprows=5 / [1, 2] : 5개 줄을 제외. 1행과 2행을 제외할 경우에는 [1, 2]로 지정
- na_values=['?', 'N/A', 'nan', 'NaN'] : 결측치로 간주할 값 지정

▣ 엑셀파일 불러오기 : **pd.read_excel('파일명.xlsx', sheet_name,** header, name, index_col, usecols, skiprows, na_values**)**



Pandas (#5/6) - 인덱싱

INDUSTRIAL AI RESEARCH CENTER

▣ df**[ ]** 인덱싱

- 열은 열명 list 형태로 인덱싱 *예) df['나이']=df.나이, df[['나이', '성별']]* *주의) df['나이':'직업'] 불가*
- 행은 숫자/인덱스 slice 형태로 인덱싱 *예) df[:3] = df['아빠':'딸']* *주의) df[1], df['초1'] 불가*
- 중첩 사용하여 원하는 데이터에 접근 *예) df[:3][['직업', '성별']]*

▣ df.**loc**[행명 [, 열명]] – **레이블** 기반 인덱싱

- 열명은 생략가능 *예) df.loc['엄마':'딸']*
- 행/열 이름으로 list, slice 모두 가능 *예) df.loc[:, "나이"], df.loc['아들', '성별':'직업'], df.loc[:, ['성별', '직업']]*

▣ df.**iloc**[행번호 [, 열번호]] – **숫자** 기반 인덱싱

- 행/열을 숫자(0, 1, ...)로 인덱싱
- 숫자, list, slice 모두 가능

▣ df.**at**[행명, 열명]

df.**iat**[행번호, 열번호]

- 하나의 요소만 선택할 때

```
data = [38, '남', '', '운동']
df.loc['삼촌'] = data
```

- df.insert(3, '취미', ['술', '요리', '미술', '게임'])
- df['취미'] = ['술', '요리', '미술', '게임']

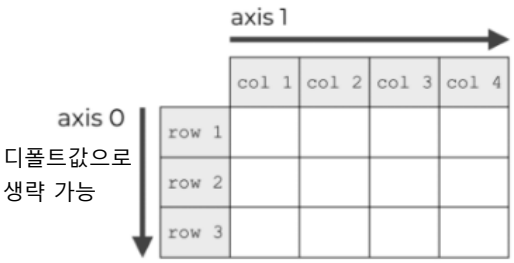
	index	나이	성별	직업	취미
0	아빠	40	남	회사원	술
1	엄마	36	여	주부	요리
2	딸	15	여	학생	미술
3	아들	11	남	학생	게임
4	삼촌	38	남		운동

Pandas (#6/6)

INDUSTRIAL AI RESEARCH CENTER

▣ DataFrame 함수

- 삭제
  - df.**drop**([label, ...,], axis=0 또는 1)
    행 삭제 : df.drop(['아빠', '삼촌'])
    열 삭제 : df.drop('취미', axis=1) *# axis='columns'*
  - df.**dropna**(inplace=True) : 결측치 삭제
  - df.**drop_duplicates**() : 중복행 삭제



- 정렬
  - df.**sort_values**(by='열명') *# Default는 오름차순*
  - df.sort_values(by=['열명1',...], ascending=[False, True]) *#[내림차순, 오름차순]*

- 조건식을 이용한 데이터 필터링
  - df['나이'].**value_counts**() : 값에 대한 빈도를 표시
  - df['성별'].value_counts() *# unique() : 범주형 데이터 확인 →array(['남', '여'], dtype=object)*
  - df[df'나이'] > 30 | df['성별']=='남'] *# 논리연산자(*/& +/| ~) 사용 : and, or 불가*

- 집계 함수 : df.sum / mean / std / max / argmax / argmin (axis)

[실습] Pandas – Dataframe 다루기 (#1/2)

INDUSTRIAL AI RESEARCH CENTER

```
import pandas as pd
import matplotlib.pyplot as plt
plt.rc('font', family='Malgun Gothic')              # 그래프 한글 출력

# 엑셀 파일로 데이터프레임 생성
df = pd.read_excel('연령별신장.xlsx', usecols='B:D', index_col='학년')
# df = pd.read_csv('연령별신장.csv', index_col='학년', name=['남', '여'])

# 데이터프레임의 기본정보 확인
df                          # 데이터 출력
df.index                    # Index(['초1', '초2', ..., '고3'], dtype='object', name='학년')
df.columns                  # Index(['남성', '여성'], dtype='object')
df.head()                   # Top 5의 데이터 출력
df.dtypes                   # 각 열의 dtypes : interger(int64), float(float64), string(object)
df.info()                   # 데이터 정보
df.describe()               # 수치 데이터의 평균, 표준편차 등의 기술 통계량 출력

# 행, 열 접근
df[:2]                  # 0~1행 데이터
df[2:3]                 # 2행 데이터
df['남성']              # df.남성
df[['남성', '여성']]
df.loc['초1']
df.loc['초1', ['남성', '여성']]
df.남성[:2]
```

	A	B	C	D
1	나이	학년	남성	여성
2	8	초1	128.5	127.2
3	9	초2	134.3	133.3
4	10	초3	142	139.9
5	11	초4	145.8	146.7
6	12	초5	152.5	153.1
7	13	초6	159.9	157.9
8	14	중1	166.7	161
9	15	중2	171.6	162.6
10	16	중3	174.3	163.3
11	17	고1	175.8	163.6
12	18	고2	176.9	163.9
13	19	고3	177.9	164.3

```
In [9]: df.info()
<class 'pandas.core.frame.DataFrame'>
Index: 12 entries, 초1 to 고3
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   남성      12 non-null     float64
 1   여성      12 non-null     float64
dtypes: float64(2)
memory usage: 288.0+ bytes
```

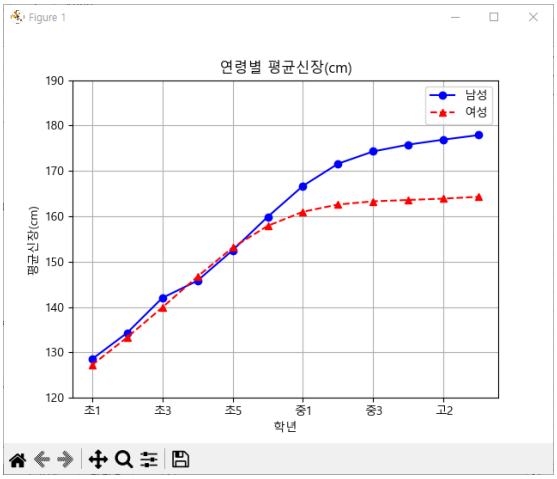
```
In [37]: df.describe()
Out[37]:
              남성          여성
count   12.000000   12.000000
mean   158.850000  153.066667
std     17.731251   13.177483
min    128.500000  127.200000
25%    144.850000  145.000000
50%    163.300000  159.450000
75%    174.675000  163.375000
max    177.900000  164.300000
```

[실습] Pandas – Dataframe 다루기 (#2/2)

INDUSTRIAL AI RESEARCH CENTER

```
# 조건식을 이용한 인덱싱과 정렬
df[(df.남성 > 130) & (df.남성 < 160)]
df.sort_values(by='남성', ascending=False)

# 통계 데이터 추가하기
df['평균'] = df[['남성', '여성']].mean(axis=1)
df['평균'] = (df['남성'] + df['여성']) / 2
df.loc['mean'] = df.mean()
df.loc['std'] = df[:-1].std()
df.loc['max'] = df[:-2].max()
df.loc['min'] = df[:'고3'].min()
df.loc['median'] = df[:'고3'].median()

# 행/열 삭제
df.drop(['중1', '고1'])
df.drop('여성', axis=1)

# 그래프 그리기
df.plot(y=['남성', '여성'], style=['b-o', 'r--^'],
        title='연령별 평균신장(cm)', ylim=(120, 190),
        grid=True, ylabel='평균신장(cm)')
```

학년	남성	여성	평균
초1	128.500000	127.200000	127.850000
초2	134.300000	133.300000	133.800000
고3	177.900000	164.300000	171.100000
mean	158.850000	153.066667	155.958333
std	17.731251	13.177483	15.365810
max	177.900000	164.300000	171.100000
min	128.500000	127.200000	127.850000



Matplotlib (#1/7)

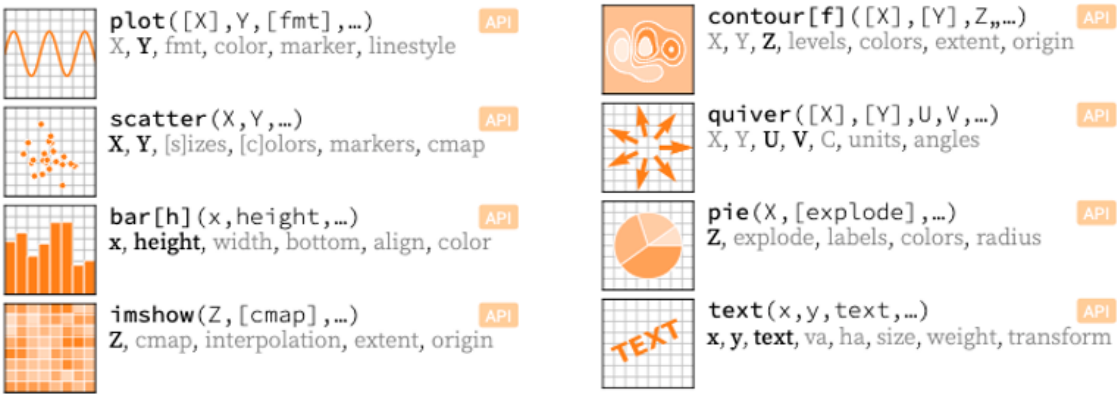
INDUSTRIAL AI RESEARCH CENTER

▣ Matplotlib이란?

matplotlib
https://matplotlib.org/

- 파이썬에서 그래프를 그릴 때 가장 널리 사용되는 라이브러리
- MATLAB(공학용 프로그래밍 언어)에서 그래프를 그리는 강력한 함수인 plot과 사용법이 거의 유사
- 그래프의 종류, 축, 눈금선 그래프 이름 등 그림 요소의 상세한 서식 설정 가능
- 편리한 툴바(toolbar)로 그래프의 커스터마이즈와 다양한 출력 형식(PNG, SVG, JPG 등)으로 저장 가능
- 설치 : **pip install matplotlib → import matplotlib.pyplot as plt**

▣ 다양한 그래프 함수



Matplotlib (#2/7)

INDUSTRIAL AI RESEARCH CENTER

▣ matplotlib 그래프의 계층구조

- matplotlib의 그래프는 다음과 같은 객체가 계층적으로 필요함

Figure 객체 > subplot(Axes) 객체 > Axes 객체 > Axis 객체 (x축, y축)

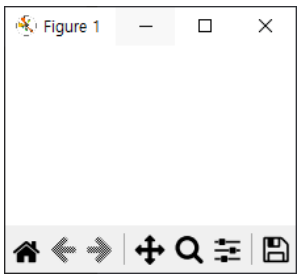
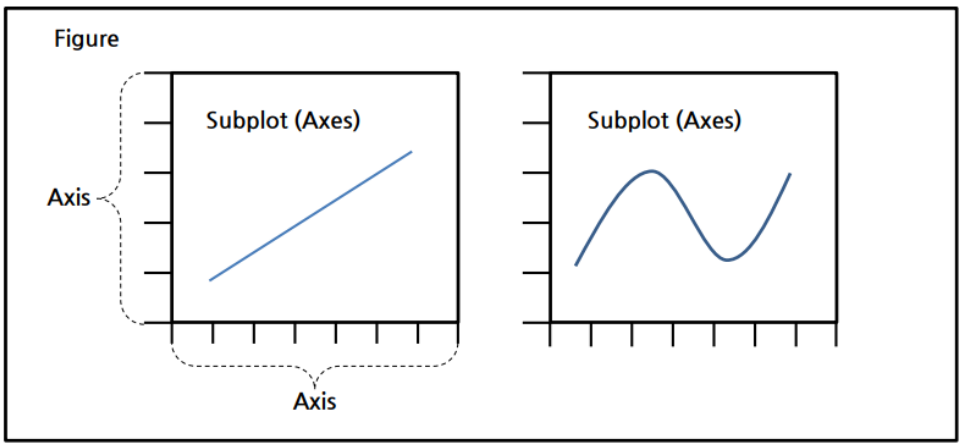


Figure	서브 플롯 (그래프영역) 작성 하는 틀
Subplot	그래프를 작성하기 위한 영역
Axes	Figure 안의 (Sub)Plot들은 각각의 Axes 객체 속함
Axis	플롯의 가로 축이나 세로 축을 의미

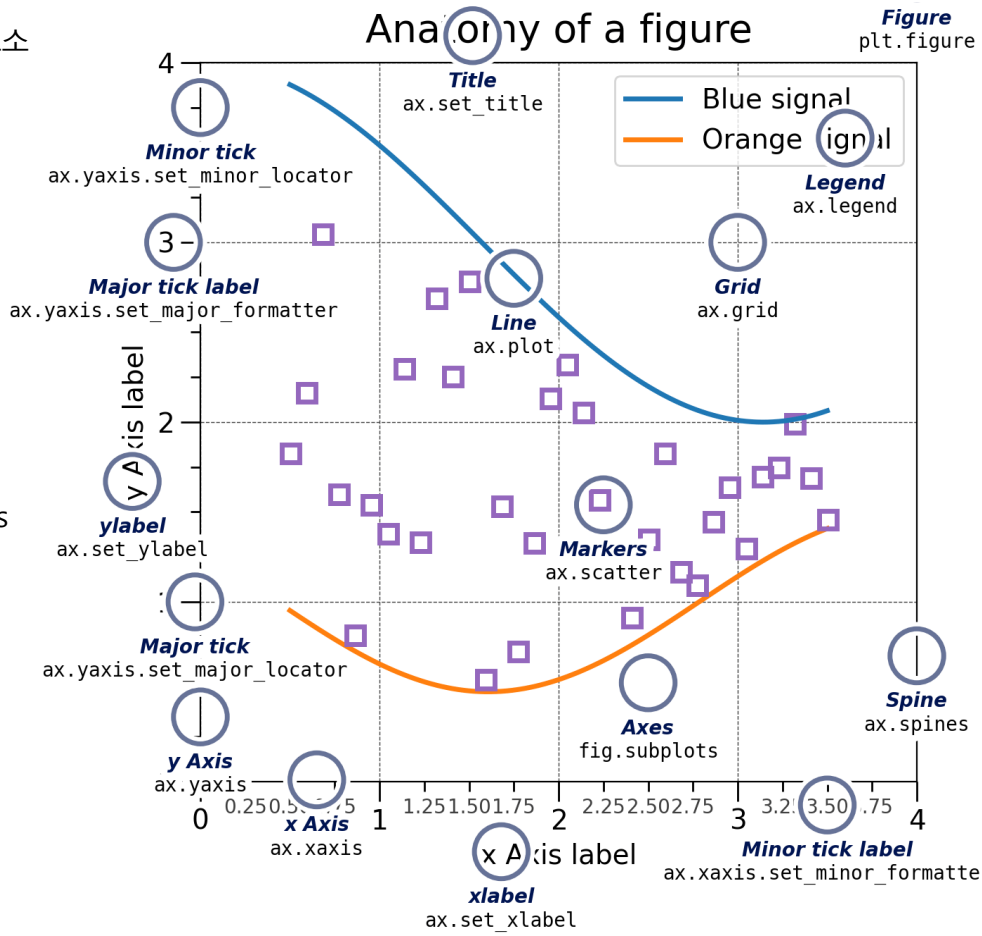


Matplotlib (#3/7)

INDUSTRIAL AI RESEARCH CENTER

▣ Axes 객체의 구성요소

- title
- xlabel
- ylabel
- grid
- legend
- spine
- plot 메서드
- axis
  - major/minor ticks
  - major/minor ticklabels



Matplotlib (#4/7)

INDUSTRIAL AI RESEARCH CENTER

▣ 그래프 그리기 절차

- 1단계 : fig = plt.figure() 로 윈도우를 만든다.
- 2단계 : ax = fig.add_subplot(행수, 열수, 순서)로 그래프 영역(axes) 추가
  예) subplot(2, 2, 1) 또는 subplot(221) 로 지정
- 3단계 : 다양한 그래프 메서드(ax.plot, bar, text 등)로 그래프 작성하고 적절한 속성(set_xlabel, set_xticks, set_xticklabels, grid, set_title 등) 지정
- 4단계 : plt.show()로 그래프 보여주기

• 1~2단계 : ax = plt.subplot(2, 2, 1)
(자동으로 figure가 생성됨)
• fig, ax = plt.subplots(2, 2) # ax[0, 1]

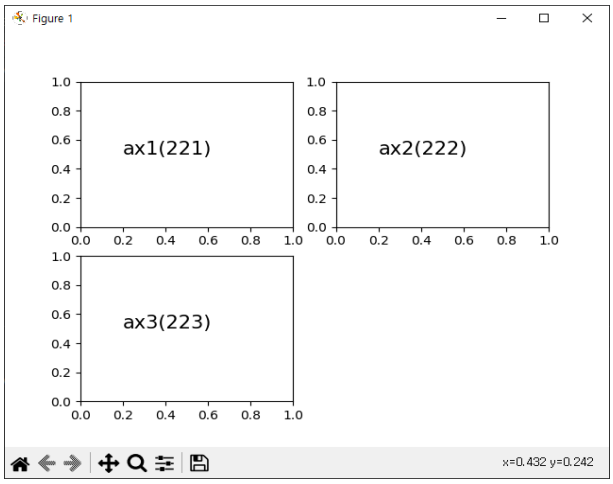
• 1~3단계 : plt.plot
최근의 axes를 찾고, 없으면 자동으로 fig>axes 1개 생성

explicit i/f OO-style (추천)

```
fig = plt.figure(figsize=(5, 4))
ax1 = fig.add_subplot(221)
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(223)
ax1.text(0.2, 0.5, 'ax1(221)', fontsize=15)
ax2.text(0.2, 0.5, 'ax2(222)', fontsize=15)
ax3.text(0.2, 0.5, 'ax3(223)', fontsize=15)
plt.show()
```

implicit i/f pyplot-style

```
plt.subplot(221)
plt.text(0.2, 0.5, 'ax1(221)', fontsize=15)
plt.subplot(222)
plt.text(0.2, 0.5, 'ax2(222)', fontsize=15)
plt.subplot(223)
plt.text(0.2, 0.5, 'ax3(223)', fontsize=15)
plt.show()
```



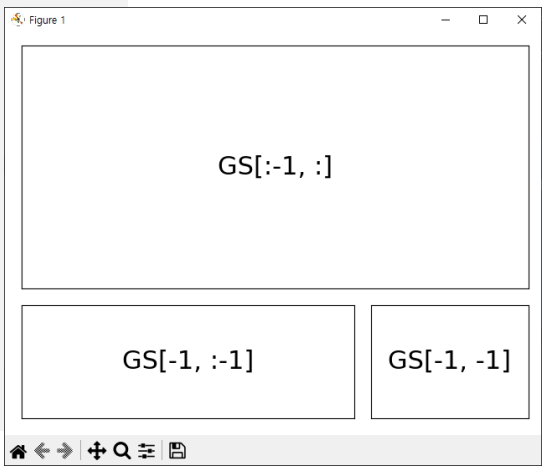
Matplotlib (#5/7)

INDUSTRIAL AI RESEARCH CENTER

▣ GridSpec을 이용한 복잡한 그래프 영역 배치

- GridSpec 함수를 이용하여 subplot의 크기와 위치를 지정

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig = plt.figure(tight_layout=True)

gs = GridSpec(3, 3)          # wspace, hspace로 subplot 간의 간격 지정
ax1 = plt.subplot(gs[:-1, :])
ax1.set_xticks([]),    ax1.set_yticks([])
ax1.text(0.5, 0.5, 'GS[:-1, :]', ha='center', va='center', size=22)

ax2 = plt.subplot(gs[-1, :-1])
ax2.set_xticks([]),    ax2.set_yticks([])
ax2.text(0.5, 0.5, 'GS[-1, :-1]', ha='center', va='center', size=22)

ax3 = plt.subplot(gs[-1, -1])
ax3.set_xticks([]),    ax3.set_yticks([])
ax3.text(0.5, 0.5, 'GS[-1, -1]', ha='center', va='center', size=22)

plt.show()
```



Matplotlib (#6/7) – 한글 사용

INDUSTRIAL AI RESEARCH CENTER

▣ 한글폰트 사용방법

- 방법1: FontProperties를 사용 → 폰트가 필요한 항목마다 지정해 주어야 함
- **방법2: matplotlib.rcParams[]으로 전역글꼴 설정** → 스크립트 내 필요한 곳에 동일하게 적용됨
- 방법3: 설정파일(matplotlibrc)로 지정 → PC 전체에 적용

```
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

# 맑은 고딕체 폰트 확인
font_list = fm.findSystemFonts(fontpaths=None, fontext='ttf')  # 폰트파일 확인[(f.name, f.fname)
for f in fm.fontManager.ttflist if 'Malgun' in f.name]
# [('Malgun Gothic', 'C:\\WINDOWS\\Fonts\\malgun.ttf'),
# ('Malgun Gothic', 'C:\\Windows\\Fonts\\malgunbd.ttf'),
# ('Malgun Gothic', 'C:\\Windows\\Fonts\\malgunsl.ttf')]
# 위 폰트가 없으면 네이버에서 배포한 나눔고딕글꼴(NanumGothic) 추천

plt.rcParams['font.family']                    # ['sans-serif']
plt.plot([1, 2])
plt.title('샘플(sample)')                      # 한글이 깨짐
plt.show()

plt.rc('font', family='Malgun Gothic')         # 폰트를 한글폰트로 지정
plt.plot([1, 2])
plt.title('샘플(sample)')                      # 한글이 제대로 나옴
plt.show()
```

□□(sample)

샘플(sample)

Matplotlib (#7/7) – 꺾은선 그래프(plot) 스타일

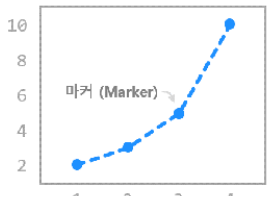
INDUSTRIAL AI RESEARCH CENTER

▣ 그래프 스타일 적용 옵션

- 색상: color = 'r', 'g', 'b' 등
- 선의 종류 : linestyle = 'solid', 'dashed', 'dotted', 'dashdotted' 등
- 마커 모양 : marker = o , >, ^ 등

예) plt.plot(x, y, **color**='b', **linestyle**='--', **marker**='o')

- 색상, 선, 마커의 조합도 가능 (순서 무관)



blue circle solid line

```
plt.plot(x, y, 'bo-')
```

blue circle dashed line

```
plt.plot(x, y, 'bo--')
```

Colors

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Line Styles

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

Markers

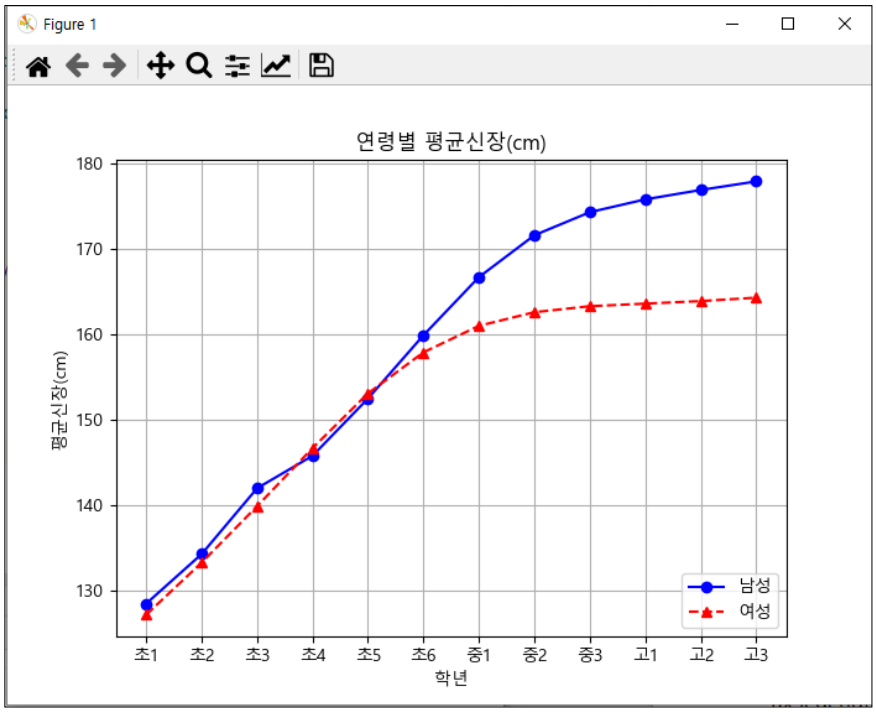
character	description
'.'	point marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

[실습] Matplotlib - 평균신장 시각화 (#1/2)

INDUSTRIAL AI RESEARCH CENTER

▣ 남녀 학생의 연령별 신장 그래프 그리기

- 엑셀/csv 파일에서 데이터를 데이터프레임 자료형으로 가져옴



• 범례 옵션(문자/코드)

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

[실습] Matplotlib - 평균신장 시각화 (#2/2)

INDUSTRIAL AI RESEARCH CENTER

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

plt.rc('font', family='Malgun Gothic')      # Matplotlib 한글폰트로 지정
df = pd.read_excel('연령별신장.xlsx', index_col='학년')

fig, ax = plt.subplots(figsize=(7, 5))

ax.plot(df.index, df.남성, 'b-o', label='남성')     # 색상+선+마크의 조합
ax.plot(df.index, df.여성, linestyle='dashed', color='r', marker='^', label='여성')

ax.set_title('연령별 평균신장(cm)')
ax.set_xlabel('학년')
ax.set_ylabel('평균신장(cm)')
ax.grid('on')
ax.legend(loc='lower right')

plt.tight_layout() # axes가 Figure 내에 최대 크기로 표시
plt.show()
```

```
df.plot(y=['남성', '여성'],     # ax = df.plot(y=...).
        figsize=(7, 5),
        kind='line',
        title='연령별 평균신장(cm)',
        ylim=(120, 200),
        grid=True,
        ylabel='평균신장(cm)',
        style=['b-o', 'r--^'])
plt.show()
```