

## 04. 기계학습과 수학

- 선형대수
  - 벡터와 행렬, 놈, 역행렬, 행렬 분해
- 확률과 통계 기초
- 최적화 이론
- Python 라이브러리

# 벡터(vector)

## ■ 벡터(vector)

- 샘플 데이터는 특징벡터(feature vector)로 표현됨

- (예) Iris 데이터

- $\mathbf{x} = (x_1, x_2, x_3, x_4)^T = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{pmatrix}$

- 여러 개의 특징벡터는 첨자로 구분

$$\mathbf{x}_1 = \begin{pmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix} 4.9 \\ 3.0 \\ 1.4 \\ 0.2 \end{pmatrix}, \mathbf{x}_3 = \begin{pmatrix} 4.7 \\ 3.2 \\ 1.3 \\ 0.2 \end{pmatrix}, \dots, \mathbf{x}_{150} = \begin{pmatrix} 5.9 \\ 3.0 \\ 5.1 \\ 1.8 \end{pmatrix}$$

# 행렬(matrix)

## ■ 행렬(matrix)

- 여러 개의 벡터를 담고 있음
- 설계행렬 : 학습데이터를 담은 행렬
  - (예) Iris 설계행렬(design matrix)

$$\mathbf{X} = \begin{pmatrix} 5.1 & 3.5 & 1.4 & 0.2 \\ 4.9 & 3.0 & 1.4 & 0.2 \\ 4.7 & 3.2 & 1.3 & 0.2 \\ 4.6 & 3.1 & 1.5 & 0.2 \\ \vdots & \vdots & \vdots & \vdots \\ 6.2 & 3.4 & 5.4 & 2.3 \\ 5.9 & 3.0 & 5.1 & 1.8 \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \\ \vdots & \vdots & \vdots & \vdots \\ x_{149,1} & x_{149,2} & x_{149,3} & x_{149,4} \\ x_{150,1} & x_{150,2} & x_{150,3} & x_{150,4} \end{pmatrix}$$

← 행 row

↑ column

- 전치행렬(transpose matrix)

- 행렬의 행과 열을 교환한 행렬

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}, \quad \mathbf{A}^T = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1m} & a_{2m} & \cdots & a_{nm} \end{pmatrix}$$

- (예)  $\mathbf{A} = \begin{pmatrix} 3 & 4 & 1 \\ 0 & 5 & 2 \end{pmatrix} \quad \mathbf{A}^T = \begin{pmatrix} 3 & 0 \\ 4 & 5 \\ 1 & 2 \end{pmatrix}$

## ■ 행렬을 이용하면 수학을 간결하게 표현할 수 있음

### ■ (예) 다항식의 행렬 표현

$$f(\mathbf{x}) = f(x_1, x_2, x_3)$$

$$= 2x_1x_1 - 4x_1x_2 + 3x_1x_3 + x_2x_1 + 2x_2x_2 + 6x_2x_3 - 2x_3x_1 + 3x_3x_2 + 2x_3x_3 + 2x_1 + 3x_2 - 4x_3 + 5$$

$$= (x_1 \ x_2 \ x_3) \begin{pmatrix} 2 & -4 & 3 \\ 1 & 2 & 6 \\ -2 & 3 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + (2 \ 3 \ -4) \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + 5$$

$$= \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c$$

## ■ 특수한 행렬들

$$\text{정사각행렬} \begin{pmatrix} 2 & 0 & 1 \\ 1 & 21 & 5 \\ 4 & 5 & 12 \end{pmatrix}, \quad \text{대각행렬} \begin{pmatrix} 50 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 8 \end{pmatrix},$$

$$\text{단위행렬} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \text{대칭행렬} \begin{pmatrix} 1 & 2 & 11 \\ 2 & 21 & 5 \\ 11 & 5 & 1 \end{pmatrix}$$

## ■ 행렬의 성질

### ■ 교환법칙 (X)

- $AB \neq BA$

### ■ 분배법칙 (O), 결합법칙 (O)

- $A(B + C) = AB + AC$

- $A(BC) = (AB)C$

## ■ 행렬의 곱셈

- $C = AB$

- 행렬 A의 열과 B의 행의 개수가 같아야 함

$$c_{ij} = \sum_{k=1,s} a_{ik}b_{kj}$$

2\*3 행렬  $A = \begin{pmatrix} 3 & 4 & 1 \\ 0 & 5 & 2 \end{pmatrix}$ 와 3\*3행렬  $B = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 0 & 5 \\ 4 & 5 & 1 \end{pmatrix}$ 을 곱하면 2\*3 행렬  $C = AB = \begin{pmatrix} 14 & 5 & 24 \\ 13 & 10 & 27 \end{pmatrix}$

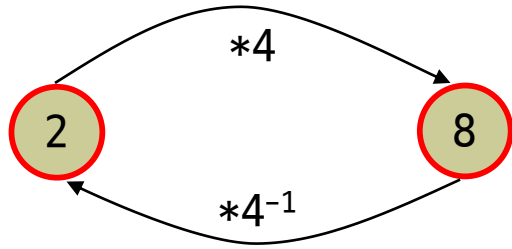
## ■ 텐서(tensor)

- 3차원 이상의 구조를 갖는 배열

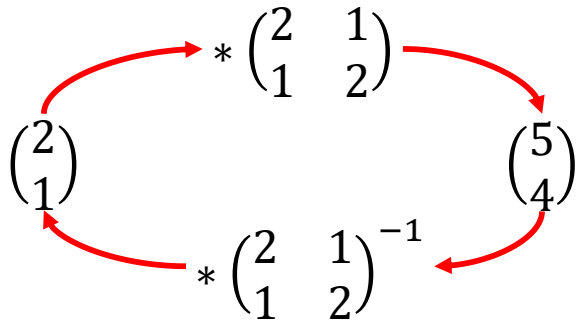
$$\mathbf{A} = \begin{pmatrix} 4 & 1 & 0 & 3 & 2 & 2 \\ 2 & 0 & 2 & 2 & 3 & 1 \\ 3 & 0 & 1 & 2 & 6 & 7 \\ 3 & 1 & 2 & 3 & 5 & 6 \\ 1 & 2 & 2 & 2 & 2 & 3 \\ 3 & 0 & 0 & 1 & 1 & 0 \\ 5 & 4 & 1 & 3 & 3 & 3 \\ 2 & 2 & 1 & 2 & 2 & 1 \end{pmatrix} \begin{pmatrix} 6 \\ 3 \\ 0 \\ 3 \\ 1 \end{pmatrix}$$

■ 역행렬 ( $A^{-1}$  : inverse matrix)

- 행렬  $A$ 와 곱하면 단위행렬  $I$ 가 나오는 행렬  $A^{-1}$ 를 역행렬이라고 함
- $A^{-1}A = AA^{-1} = I$



$$4^{-1} * 4 = 1$$



$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}^{-1} * \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$



## ■ 역행렬 ( $A^{-1}$ : inverse matrix)

- 정방행렬에 대해서만 정의됨
- 역행렬이 없으면 특이행렬(singular matrix)

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

### ■ (예)

$$A = \begin{pmatrix} 2 & 1 \\ 6 & 4 \end{pmatrix}$$

$$A^{-1} = \frac{1}{2 \cdot 4 - 1 \cdot 6} \begin{pmatrix} 4 & -1 \\ -6 & 2 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 4 & -1 \\ -6 & 2 \end{pmatrix} = \begin{pmatrix} 2 & -0.5 \\ -3 & 1 \end{pmatrix}$$

## ■ Determinant(행렬식)

### ■ 어떤 행렬의 역행렬 존재여부에 대한 판별값

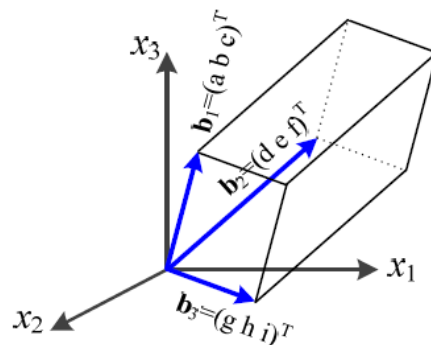
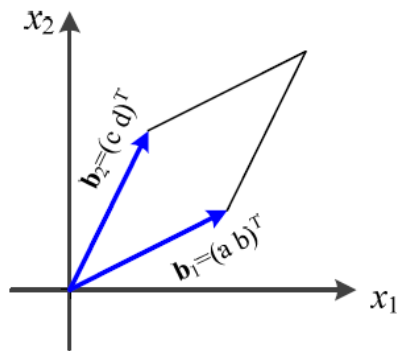
- $\det$ 의 값이 0이면 역행렬 없음

### ■ 정방행렬에 대해서만 정의됨

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \det(A) = ad - bc$$

### ■ 기하학적 의미

- 2차원 : 2개의 행벡터가 이루는 평행사변형의 넓이
- 3차원 : 3개의 행벡터가 이루는 평행사각기둥의 부피



# 고유벡터와 고유값

■  $A\mathbf{v} = \lambda\mathbf{v}$

■  $\mathbf{v}$  : 고유벡터(eigen vector: 위 식을 만족하는 0이 아닌 벡터)

■  $\lambda$  : 고유값(eigen value)

- $m \times m$ 행렬은 최대  $m$ 개의 고유값과 고유벡터를 가질 수 있음
- 모든 고유벡터는 서로 직교(orthogonal). 길이는 고유값에 따름

■ (예)  $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

■  $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 3 \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$\lambda = 3, \mathbf{v} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

■  $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = 1 \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

$\lambda = 1, \mathbf{v} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

# 고유벡터와 고유값의 기하학적 의미

- 반지름 1인 원에 있는 4개의 벡터

- $x_1 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$ ,  $x_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ ,  $x_3 = \begin{pmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix}$ ,  $x_4 = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$

- $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ 로 변환

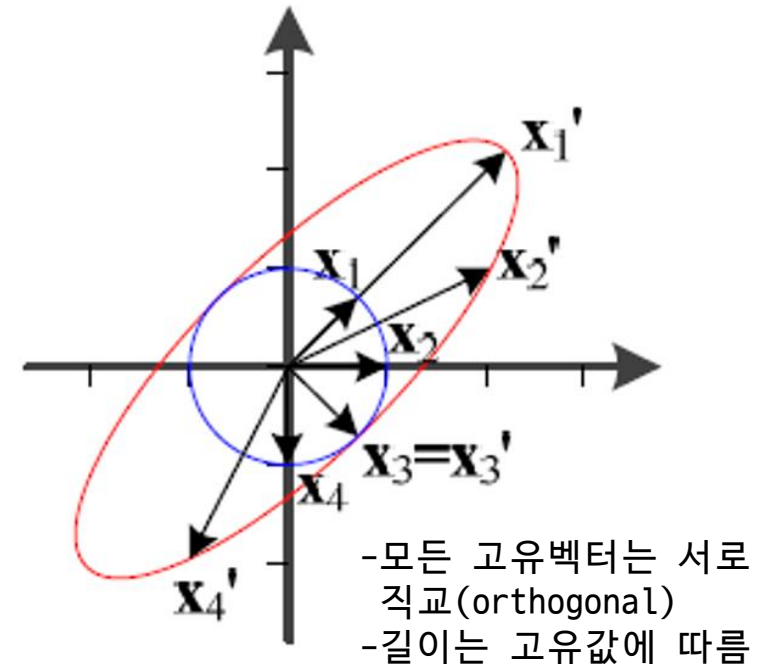
- $x'_1 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} = 3 \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$

- $x'_2 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$

- $x'_3 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix} = 1 \begin{pmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix}$

- $x'_4 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ -2 \end{pmatrix}$

- 파란색 원이 빨간색 타원으로 바뀌더라도  
방향이 바뀌지 않는 것은 **고유벡터**인  $x_1$ 과  $x_3$  뿐이다.



# 고유값 분해

- 분해(decomposition)
  - 소인수분해(factorization)
- 고유값 분해(행렬  $A$ 를 다음과 같이 분해)
  - $A = Q\Lambda Q^{-1}$ 
    - $Q$  :  $A$ 의 고유벡터를 열에 배치한 행렬
    - $\Lambda$  : 고유값을 대각선에 배치한 대각행렬
  - 참고: 특이값 분해( $n*m$ 행렬인 경우)
    - 고유값 분해는 정방행렬만 분해 가능
    - 정방행렬이 아닌 경우에도 분해가 가능
    - $A = U\Sigma V^{-T}$ 
      - $U$ : 특이행렬( $AA^T$ 의 고유벡터를 열에 배치한  $n*n$ 행렬)
      - $\Sigma$ :  $AA^T$ 의 고유값의 제곱근을 대각선에 배치한 대각행렬( $n*m$ 행렬)
      - $V^{-T}$ : 특이행렬( $A^TA$ 의 고유벡터를 열에 배치한  $m*m$ 행렬)

# 고유값분해(예1)

■  $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

■  $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 3 \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$$\lambda = 3, \mathbf{v} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

■  $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = 1 \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

$$\lambda = 1, \mathbf{v} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

■  $Q = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad Q^{-1} = \frac{1}{(1*-1)-(1*1)} \begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{pmatrix}$

■  $\Lambda = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$

■  $A = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{pmatrix}$

# 고유값분해(예2)

■  $A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 4 \\ 0 & 4 & 9 \end{pmatrix}$

■  $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 4 \\ 0 & 4 & 9 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} = 11 \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$

$\lambda = 11, \mathbf{v} = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$

■  $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 4 \\ 0 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 2 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

$\lambda = 2, \mathbf{v} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

■  $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 4 \\ 0 & 4 & 9 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix} = 1 \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix}$

$\lambda = 1, \mathbf{v} = \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix}$

■  $Q = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ 2 & 0 & -1 \end{pmatrix}$

$Q^{-1} = \begin{pmatrix} 0 & 0.2 & 0.4 \\ 1 & 0 & 0 \\ 0 & 0.4 & -0.2 \end{pmatrix}$

$\Lambda = \begin{pmatrix} 11 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

■  $A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ 2 & 0 & -1 \end{pmatrix} \begin{pmatrix} 11 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0.2 & 0.4 \\ 1 & 0 & 0 \\ 0 & 0.4 & -0.2 \end{pmatrix}$

# 확률과 통계

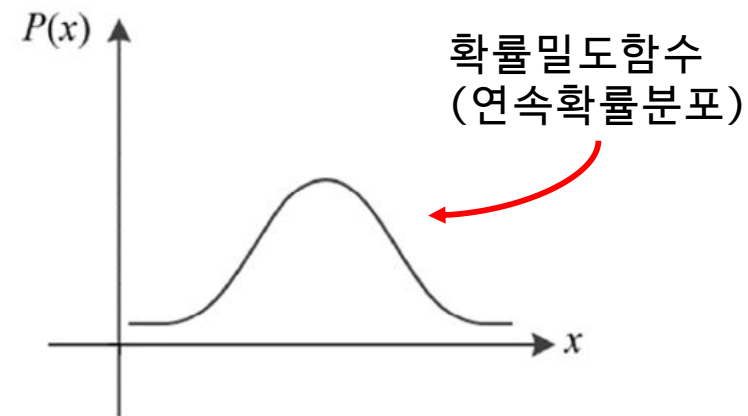
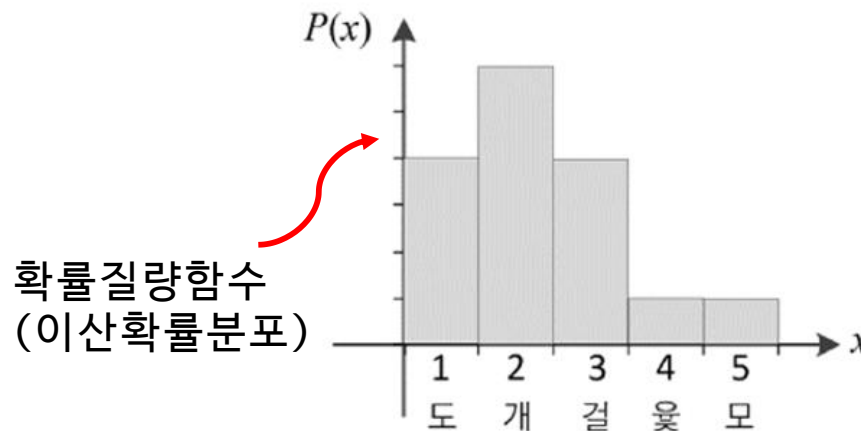
## ■ 확률분포(probability distribution)

### ■ 정의역 전체의 확률을 표현한 것

- 확률변수: 확률을 수식으로 표현하기 위한 변수
- 정의역: 확률변수가 가질 수 있는 값의 범위

### ■ (예)

- $P(x = \text{도}) = \frac{4}{16}$ ,  $P(x = \text{개}) = \frac{6}{16}$ ,  $P(x = \text{걸}) = \frac{4}{16}$
- $P(x = \text{웃}) = \frac{1}{16}$ ,  $P(x = \text{모}) = \frac{1}{16}$





## ■ 결합확률(joint probability)

- 두 사건이 결합된 상태의 확률  $P(y, x)$

- $P(y, x) = P(x|y)P(y)$

- $P(x|y)$  : 조건부확률(conditional probability)

- (예)

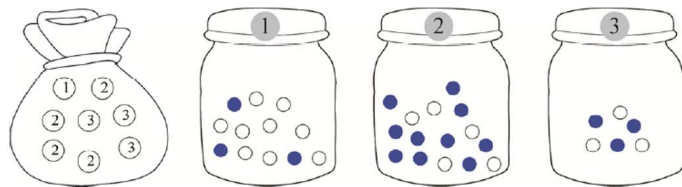
- 확률변수  $y = \{1, 2, 3\}$   $x = \{\text{흑색}, \text{흰색}\}$

- 주머니에서 1을 꺼낼 확률  $1/8$ , 1번 병에서 흰색공을 꺼낼 확률  $9/12$

- 1번 카드이고 흰색 공일 확률

- $P(y = 1, x = \text{흰색})$

$$= P(x = \text{흰색} | y = 1)P(y = 1) = \frac{9}{12} \cdot \frac{1}{8} = \frac{3}{32}$$



# 베이즈 정리(Bayes formula)

- 일반적으로  $x$ 와  $y$ 가 동시에 일어날 결합확률과  $y$ 와  $x$ 가 동시에 일어날 결합확률을 같음

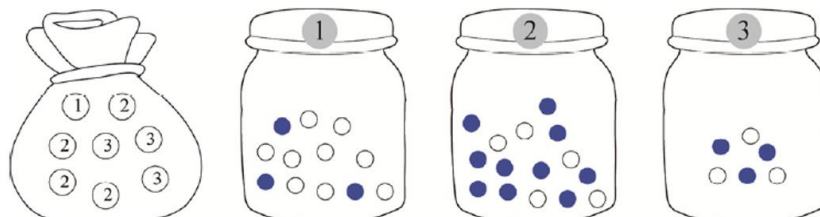
- $P(y, x) = P(x|y)P(y) = P(x, y) = P(y|x)P(x)$

- 따라서,

우도(likelihood)      사전확률

사후확률

$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$



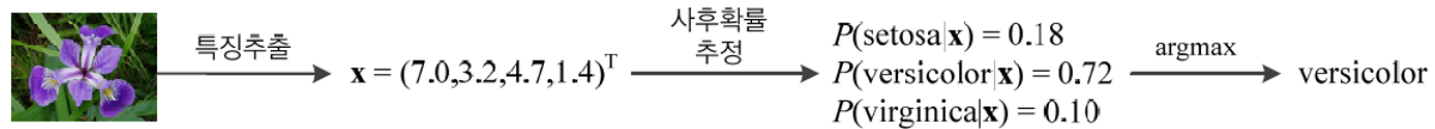
- 사전확률과 우도를 구할 수 있다면, 사후확률을 간접적으로 계산할 수 있음
  - (예) 흰공이 나왔다는 사실은 알고 있지만, 어느 병에서 나온지는 모른다. 어느 병에서 나왔는지 추정하시오.

$$\hat{y} = \operatorname{argmax}_y P(y|x = \text{흰색}) = \operatorname{argmax}_y \frac{P(x = \text{흰색}|y)P(y)}{P(x = \text{흰색})}$$

## ■ 기계 학습에 적용

### ■ (예) Iris 데이터 분류 문제

- 특징 벡터  $\mathbf{x}$ ,  $y \in \{\text{setosa}, \text{versicolor}, \text{virginica}\}$
- 분류 문제를  $\text{argmax}$ 로 표현하면
- $\hat{y} = \text{argmax}_y P(y|\mathbf{x})$



- 사후확률을 직접 추정하는 것을 거의 불가능
- 따라서 Bayes formula를 이용하여 추정

# 평균과 분산

■ 평균  $\mu = \frac{1}{n} \sum_{i=1}^n x_i$       분산  $\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2$

■ 특징벡터(확률벡터)의 평균과 분산

■  $\mu = (\mu_1, \mu_2, \dots, \mu_d)^T$ , 특징이 d개인 데이터에 대한 평균벡터.  $\mathbf{x}_i$  : i번째 sample

■ 평균 :  $\mu = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$

■ 분산은 개별 확률변수의 분산과 다른 변수간의 공분산이 있으므로 분산은 다음과 같이 표현

■  $\Sigma = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$

$$\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \dots & \sigma_{1d} \\ \sigma_{21} & \sigma_{22} & & \sigma_{2d} \\ & \vdots & \ddots & \vdots \\ \sigma_{d1} & \sigma_{d2} & \dots & \sigma_{dd} \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \dots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & & \sigma_{2d} \\ & \vdots & \ddots & \vdots \\ \sigma_{d1} & \sigma_{d2} & \dots & \sigma_d^2 \end{pmatrix}$$

Iris 데이터베이스의 샘플 중 8개만 가지고 공분산 행렬을 계산하자.

$$\mathbb{X} = \left\{ \mathbf{x}_1 = \begin{pmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix} 4.9 \\ 3.0 \\ 1.4 \\ 0.2 \end{pmatrix}, \mathbf{x}_3 = \begin{pmatrix} 4.7 \\ 3.2 \\ 1.3 \\ 0.2 \end{pmatrix}, \mathbf{x}_4 = \begin{pmatrix} 4.6 \\ 3.1 \\ 1.5 \\ 0.2 \end{pmatrix}, \mathbf{x}_5 = \begin{pmatrix} 5.0 \\ 3.6 \\ 1.4 \\ 0.2 \end{pmatrix}, \mathbf{x}_6 = \begin{pmatrix} 5.4 \\ 3.9 \\ 1.7 \\ 0.4 \end{pmatrix}, \mathbf{x}_7 = \begin{pmatrix} 4.6 \\ 3.4 \\ 1.4 \\ 0.3 \end{pmatrix}, \mathbf{x}_8 = \begin{pmatrix} 5.0 \\ 3.4 \\ 1.5 \\ 0.2 \end{pmatrix} \right\}$$

먼저 평균벡터를 구하면  $\boldsymbol{\mu} = (4.9125, 3.3875, 1.45, 0.2375)^T$ 이다. 첫 번째 샘플  $\mathbf{x}_1$ 을 식 (2.39)에 적용하면 다음과 같다.

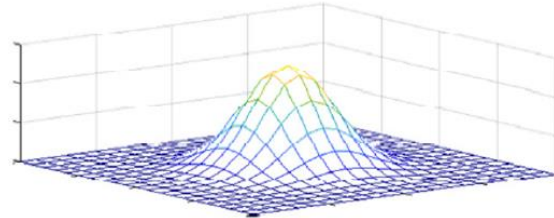
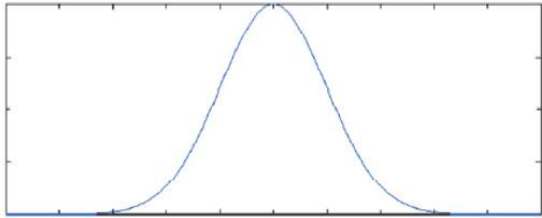
$$\begin{aligned} (\mathbf{x}_1 - \boldsymbol{\mu})(\mathbf{x}_1 - \boldsymbol{\mu})^T &= \begin{pmatrix} 0.1875 \\ 0.1125 \\ -0.05 \\ -0.0375 \end{pmatrix} (0.1875 \quad 0.1125 \quad -0.05 \quad -0.0375) \\ &= \begin{pmatrix} 0.0325 & 0.0211 & -0.0094 & -0.0070 \\ 0.0211 & 0.0127 & -0.0056 & -0.0042 \\ -0.0094 & -0.0056 & 0.0025 & 0.0019 \\ -0.0070 & -0.0042 & 0.0019 & 0.0014 \end{pmatrix} \end{aligned}$$

나머지 7개 샘플도 같은 계산을 한 다음, 결과를 모두 더하고 8로 나누면 다음과 같은 공분산 행렬을 얻는다.

$$\boldsymbol{\Sigma} = \begin{pmatrix} 0.0661 & 0.0527 & 0.0181 & 0.0083 \\ 0.0527 & 0.0736 & 0.0181 & 0.0130 \\ 0.0181 & 0.0181 & 0.0125 & 0.0056 \\ 0.0083 & 0.0130 & 0.0056 & 0.0048 \end{pmatrix}$$

## ■ 가우시안 확률분포

$$N(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x - \mu}{\sigma}\right)^2\right)$$



## ■ 다차원 가우시안 분포

### ■ 평균벡터와 공분산행렬로 정의

$$N(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{|\boldsymbol{\Sigma}|}\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

# 정보이론과 자기정보

## ■ 어떤 메시지(message)가 지닌 정보를 수치화할 수 있을까?

### ■ 정보이론의 기본원리

- 낮은 확률의 사건일수록 더 많은 정보를 전달한다
  - 사하라 사막에 눈이 왔다.(많은 정보 전달)
  - 히말라야 산에 눈이 왔다.

### ■ 자기정보(self-information)

- 어떤 사건  $e_i$ 에 대한 확률변수  $\mathbf{x} = \{e_1, e_2, \dots, e_k\}$  일 때, 사건이 일어날 확률을 추정할 수 있다면, 그 사건에 대한 정보량을 측정할 수 있음
- $h(e_i) = -\log_2 P(e_i)$  또는  $h(e_i) = -\log_e P(e_i)$ 
  - (예) 모가 나온 확률의 자기정보  $h(\text{모}) = \log_2 \frac{1}{16} = 4$
  - (예) 개가 나온 확률의 자기정보  $h(\text{개}) = \log_2 \frac{6}{16} = 1.415$ 
    - 모가 나왔다는 메시지가 개가 나왔다는 메시지보다 정보량이 2.8배 많다.

# 엔트로피(entropy)

- 확률분포의 불확실성(uncertainty), 무질서 정도

$$H(x) = - \sum_{i=1,k} P(e_i) \log_2 P(e_i) \quad H(x) = - \sum_{i=1,k} P(e_i) \log_e P(e_i)$$

- 자기정보 : 특정사건  $e_i$ 의 정보량

- (예) 주사위가 윷보다 불확실성이 더 크다.

윷을 나타내는 확률변수를  $x$ 라 할 때  $x$ 의 엔트로피는 다음과 같다.

$$H(x) = - \left( \frac{4}{16} \log_2 \frac{4}{16} + \frac{6}{16} \log_2 \frac{6}{16} + \frac{4}{16} \log_2 \frac{4}{16} + \frac{1}{16} \log_2 \frac{1}{16} + \frac{1}{16} \log_2 \frac{1}{16} \right) = 2.0306 \text{비트}$$

주사위는 눈이 6개인데 모두 1/6이라는 균일한 확률을 가진다. 이 경우 엔트로피를 계산하면 다음과 같다.

$$H(x) = - \left( \frac{1}{6} \log_2 \frac{1}{6} + \frac{1}{6} \log_2 \frac{1}{6} + \frac{1}{6} \log_2 \frac{1}{6} + \frac{1}{6} \log_2 \frac{1}{6} + \frac{1}{6} \log_2 \frac{1}{6} + \frac{1}{6} \log_2 \frac{1}{6} \right) = 2.585 \text{ 비트}$$



# 교차 엔트로피(cross-entropy)

## ■ 두 확률분포의 차이

■ 서로 다른 확률분포 P와 Q에 대해

$$E(P, Q) = - \sum_x P(x) \log_2 Q(x) = - \sum_{i=1,k} P(e_i) \log_2 Q(e_i)$$

$$\begin{aligned} E(P, Q) &= - \sum_x P(x) \log_2 Q(x) \\ &= - \sum_x P(x) \log_2 P(x) + \left( \sum_x P(x) \log_2 P(x) - \sum_x P(x) \log_2 Q(x) \right) \\ &= E(P) + \sum_x P(x) \log_2 \frac{P(x)}{Q(x)} \end{aligned}$$

KL-divergence  
(Kullback-Leibler divergence)  
0이면 두 분포가 같음을 의미

# 최적화 이론

- 최적화(optimization)

- 미분 가능한 함수의 최저점을 찾는 문제

- 기계학습의 최적화

- 학습데이터에 따라 정해지는 목적함수의 최저점을 탐색

- 미분하는 과정 필요(오차 역전파)
- 주로 SGD(stochastic gradient descent;스토캐스틱 경사하강법) 사용

1. 난수를 생성하여 초기해  $\Theta$ 를 구한다.

2. repeat

목적함수  $J(\Theta)$ 가 작아지는 방향  $d\Theta$ 를 구한다.

$$\Theta = \Theta + d\Theta$$

3. until(멈춤조건)

4.  $\hat{\Theta} = \Theta$

## ■ 미분으로 최저점을 찾아가는 방법

### ■ 미분의 정의

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

- 1차 도함수는 함수의 기울기(값이 커지는 방향 지시)
- $-f'(x)$  방향에 목적함수의 최저점이 존재함
- 즉,  $-f'(x)$  방향으로 가면 최저점을 찾을 수 있음

### ■ 편미분

- 변수가 여러 개인 함수의 미분
- 각각의 변수에 대해 독립적으로 미분을 수행
  - 기계학습에서는 매개변수가 많으므로 편미분을 사용

## ■ 경사하강 알고리즘

### ■ 배치 경사하강 알고리즘

- 샘플의 gradient를 평균한 후, 한꺼번에 갱신

### ■ 스토캐스틱 경사하강 알고리즘

- 한 샘플의 gradient를 계산한 후 즉시 갱신

1. 난수를 생성하여 초기해  $\Theta$ 를 구한다.
2. repeat  
    샘플의 순서를 섞는다.  
    for( $i=1$  to  $n$ )  
         $i$ 번째 샘플에 대한 gradient  $\Delta_i$  계산  
         $\Theta = \Theta - \rho \Delta_i$
3. until(멈춤조건)
4.  $\hat{\Theta} = \Theta$

# Python 라이브러리

- **math**
- **numpy**
- **matplotlib**
- **pandas**

# 참고: 수학과 관련된 python 내장함수

- `abs(x)`
  - `x`의 절대값을 반환 # `abs(-2) => 2`
- `max(x1, x2, ...)`
  - 주어진 수 중 가장 큰 값을 반환
- `min(x1, x2, ...)`
  - 주어진 수 중 가장 작은 값을 반환
- `pow(a, b)`
  - $a^b$ 을 계산하여 반환
- `round(x)`
  - `x`에 가장 가까운 정수를 반환(동일하면 짝수를 반환)
- `round(x, n)`
  - 소수점이하  $n+1$ 번째 자리에서 반올림한 실수를 반환

## (1) math 라이브러리

# math 라이브러리

- 다양한 수학과 관련된 함수를 제공

- `import math`

- 상수

- `math.pi`      `math.e`

- 함수

- `fabs(x)` :  $x$ 의 절대값을 실수로 반환

- `ceil(x)`, `floor(x)`

- `exp(x)`, `log(x, base)`, `log(x)`는  $\log_e x$

- `sqrt(x)`

- `sin(x)`, `cos(x)`, `tan(x)`

- `degree(x)` : 라디안을 각도로 변환

- `radian(x)` : 각도를 라디안으로 변환



# 예제

```
import math

print("exp(1.0)=", math.exp(1.0))
print("log(2.78)=", math.log(2.78))      #  $\log_e 2.78$ 
print("log10(20)=", math.log(20, 10))    #  $\log_{10} 2.78$ 
print("sqrt(4.0)=", math.sqrt(4.0))

print("sin(PI/2)=", math.sin(math.pi / 2.0))
print("cos(PI/2)=", math.cos(math.pi / 2.0))
print("tan(PI/2)=", math.tan(math.pi / 2.0))

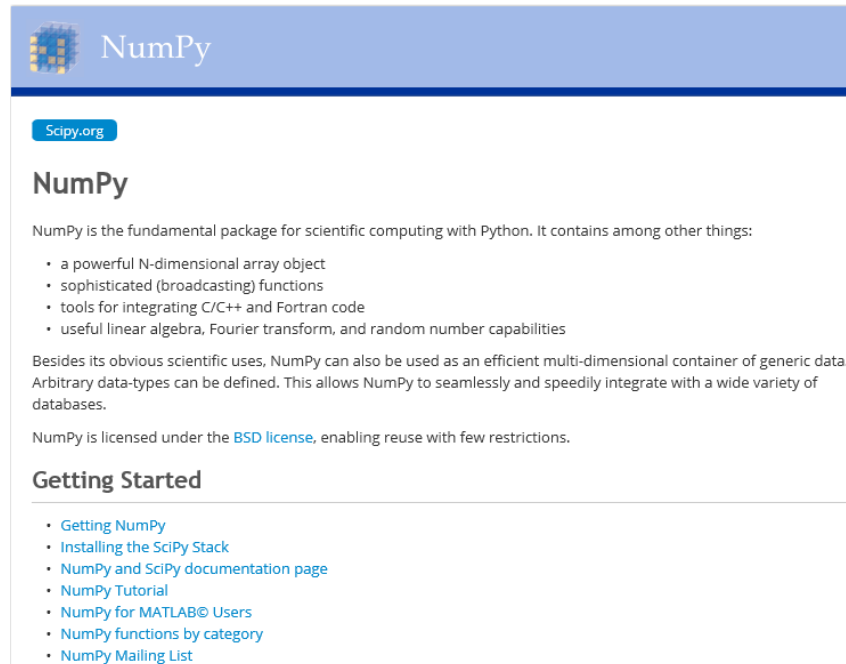
print("degree(1.57)=", math.degrees(1.57))
print("radian(90)=", math.radians(90))
```

```
exp(1.0)= 2.718281828459045
log(2.78)= 1.0224509277025455
log10(20)= 1.301029995663981
sqrt(4.0)= 2.0
sin(PI/2)= 1.0
cos(PI/2)= 6.123233995736766e-17
tan(PI/2)= 1.633123935319537e+16
degree(1.57)= 89.95437383553924
radian(90)= 1.5707963267948966
```

## (2) numpy 라이브러리

# mumpy 라이브러리

- mumpy는 데이터 수치분석을 위한 python 패키지
- 행렬의 연산에 좋은 성능을 보임
- numpy의 공식 홈페이지
  - <http://www.numpy.org/>



# ndarray 객체

- ndarray(또는 array)
  - n차원 배열객체
  - 같은 종류의 데이터만 배열에 담을 수 있음
- axes : 차원의 번호
- rank : 차원의 수
- shape : 배열의 차원을 나타내는 Tuple
- (예)
  - `[[1.0, 0.0, 0.0],`
  - `[0.0, 1.0, 2.0]]`
  - rank : 2
  - 첫번째 axis는 2, 두번째 axis는 3
  - shape : (2,3)

- `ndarray.ndim`

- axes의 수 : rank

- `ndarray.shape`

- 배열의 차원(dimension)을 나타내는 tuple
- (예) n행, m열 => (n,m)

- `ndarray.size`

- 배열요소의 총 개수

- `ndarray.dtype`

- 자료형 (numpy.int32, numpy.int16, numpy.float64 등)

- `ndarray.itemsize`

- 각 요소의 크기(byte)

## ■ (예)

```
import numpy as np

a = np.arange(15).reshape(3, 5)
print(a)
print("shape = ", a.shape)
print("ndim = ", a.ndim)
print("datatype = ", a.dtype)
print("itemsize =", a.itemsize)
print("size =", a.size)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
shape = (3, 5)
ndim = 2
datatype = int32
itemsize = 4
size = 15
```

- rank가 2인 배열(2차원 배열)이고...
- shape은 (3, 5)입니다.

## ■ (예)

```
import numpy as np

a = np.arange(15).reshape(3, 5)
print(a)
print("shape = ", a.shape)
print("ndim = ", a.ndim)
print("datatype = ", a.dtype)
print("itemsize =", a.itemsize)
print("size =", a.size)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
shape = (3, 5)
ndim = 2
datatype = int32
itemsize = 4
size = 15
```

- rank가 2인 배열(2차원 배열)이고...
- shape은 (3, 5)입니다.

# ndarray의 생성

## ■ 생성방법

- `array(x)` : List x를 이용하여 배열 생성
  - 반드시 List를 초기값으로 주어야 함
- `arange(start, end, step)` : 순차값을 가진 배열 생성
  - 1개의 매개변수 n인 경우  $0 \sim n-1$  순으로 초기화됨
- `linspace(start, end, num)` : 순차값을 가진 배열을 num개 생성
  - end값 포함, 자동으로 step을 결정
- `zeros(s)` : 주어진 shape s에 따라 0값으로 초기화된 배열 생성
  - shape은 (n,m)과 같이 Tuple로 주어짐
- `ones(s)` : 주어진 shape s에 따라 1값으로 초기화된 배열 생성
- `empty(s)` : 주어진 shape s에 따라 초기화되지 않은 배열 생성
- `full(s, value)` : 주어진 shape s에 따라 value로 초기화된 배열



- `random.randint(start, end)` : start부터 end-1사이의 정수 중에서 1개를 임의로 생성
  - 인수가 1개(end)만 있는 경우에는 0 ~ end-1 사이의 정수 1개 생성
- `random.rand(m, n)` : 균등분포인 m행 n열의 실수를 0 ~ 1사이에서 생성
  - 인수가 1개(m)만 있는 경우에는 m개의 난수 발생
- `random.randn(m, n)` : 가우시안 정규분포(평균이 0이고 분산이 1인)인 m행 n열의 실수를 생성
  - 인수가 1개(m)만 있는 경우에는 m개의 난수 발생
- `fromfunction(func, s)` : 주어진 shape s에 따라 func함수를 이용하여 생성. 단, 각 행과 열의 요소의 값은 index와 동일

## ■ 예 : array()

```
import numpy as np

a = np.array([1, 2, 3, 4, 5])
b = np.array([[1, 2, 3, 4],
              [10, 20, 30, 40]])

print(a, "\n")
print(b)
```

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

```
[[ 1  2  3  4]
 [10 20 30 40]]
```

## ■ 예 : arange()

```
import numpy as np

a = np.arange(5)
b = np.arange(10, dtype=np.float)
c = np.arange(3.1, 5, 0.25)

print(a, "\n")
print(b, "\n")
print(c)
```

```
[0 1 2 3 4]
```

```
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

```
[3.1 3.35 3.6 3.85 4.1 4.35 4.6 4.85]
```

■ 예 : linspace()

```
import numpy as np  
  
a = np.linspace(0, 0.2, 6)  
  
print(a)
```

```
[0.    0.04 0.08 0.12 0.16 0.2 ]
```

■ 예 : empty(), ones(), zeros(), full()

```
import numpy as np

a = np.empty((2, 3))
print(a, '\n')
b = np.ones((2, 3))
print(b, '\n')
c = np.zeros((2, 3))
print(c, '\n')
d = np.full((2, 3), np.pi)
print(d)
```

```
[[0.  0.04 0.08]
 [0.12 0.16 0.2 ]]
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
[[3.14159265 3.14159265 3.14159265]
 [3.14159265 3.14159265 3.14159265]]
```

## ■ 예 : randint(), rand() randn()

```
import numpy as np

a = np.random.randint(10)
print(a, '\n')
b = np.random.randint(1, 10)
print(b, '\n')
c = np.random.rand(5)
print(c, '\n')
d = np.random.rand(3, 4)
print(d, '\n')
e = np.random.randn(3, 4)
print(e)
```

0

2

[0.5781477 0.31452747 0.00683025 0.93562899  
0.91016152]

[[0.71352295 0.9689287 0.14166002 0.23908673]  
[0.61533556 0.15867572 0.75578179 0.07109531]  
[0.1749837 0.63295633 0.6426747 0.67090928]]

[[ -0.06546462 -0.75385209 0.56705616 -1.10309302]  
[ 1.33097749 -1.30430042 0.82065111 -2.95746286]  
[ 0.38655304 0.05706479 -0.06809105 0.27864592]]

## ■ 예 : fromfunction()

```
import numpy as np

def my_func(x, y):
    return x + y

a = np.fromfunction(my_func, (3, 4))
print(a)
b = np.fromfunction(my_func, (3, 4), dtype=np.int)
print(b)
```

```
[[0. 1. 2. 3.]
 [1. 2. 3. 4.]
 [2. 3. 4. 5.]]
[[0 1 2 3]
 [1 2 3 4]
 [2 3 4 5]]
```

# 기본연산

## ■ 요소간의 연산

- +, -, \*, /, //, %, \*\* 등
- 배열의 크기는 같아야 함

```
import numpy as np

a = np.array([[20, 30],
              [40, 50]])
b = np.array([[1, 2],
              [3, 4]])

c = a + b
print(c, "\n")
d = a - b
print(d, "\n")
e = a * b    # elementwise product
print(e)
```

```
[[21 32]
 [43 54]]

[[19 28]
 [37 46]]

[[ 20  60]
 [120 200]]
```



## ■ 행렬의 곱 : dot(A,B)

- 행렬의 곱셈은 A행렬 열의 크기와 B행렬 행의 크기가 같아야 함

```
import numpy as np

a = np.array([[20, 30],
              [40, 50]])
b = np.array([[1, 2],
              [3, 4]])
c = np.dot(a, b)
print(c)
```

```
[[110 160]
 [190 280]]
```

# ndarray의 reshape

## ■ 방법

- shape 속성 변경 (자기 수정)
- reshape() 사용

```
import numpy as np

a = np.arange(12)
print(a, '\n')
a.shape = (3, 4)
print(a, '\n')
b = a.reshape(2, 2, 3)
print(b)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
[[[ 0  1  2]
   [ 3  4  5]]
```

```
[[ 6  7  8]
 [ 9 10 11]]]
```

## ■ ravel() - 실을 감다...

```
a =  
[[[ 0  1]  
 [ 2  3]  
 [ 4  5]  
 [ 6  7]]
```

```
[[ 8  9]  
 [10 11]  
 [12 13]  
 [14 15]]
```

```
[[16 17]  
 [18 19]  
 [20 21]  
 [22 23]]]
```

```
b =  
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

```
import numpy as np
```

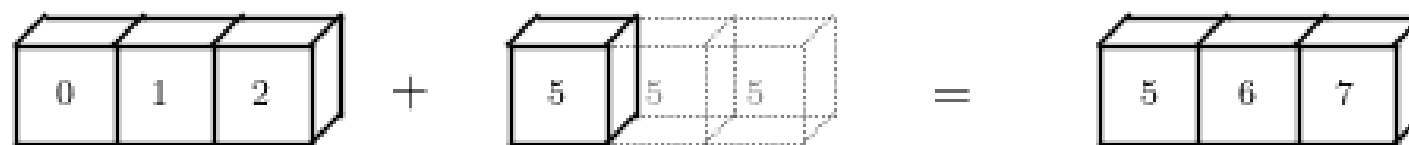
```
a = np.arange(24).reshape(3,4,2)  
print(f"a = \n{a}", '\n')
```

```
b = a.ravel()  
print(f"b = \n{b}", '\n')
```

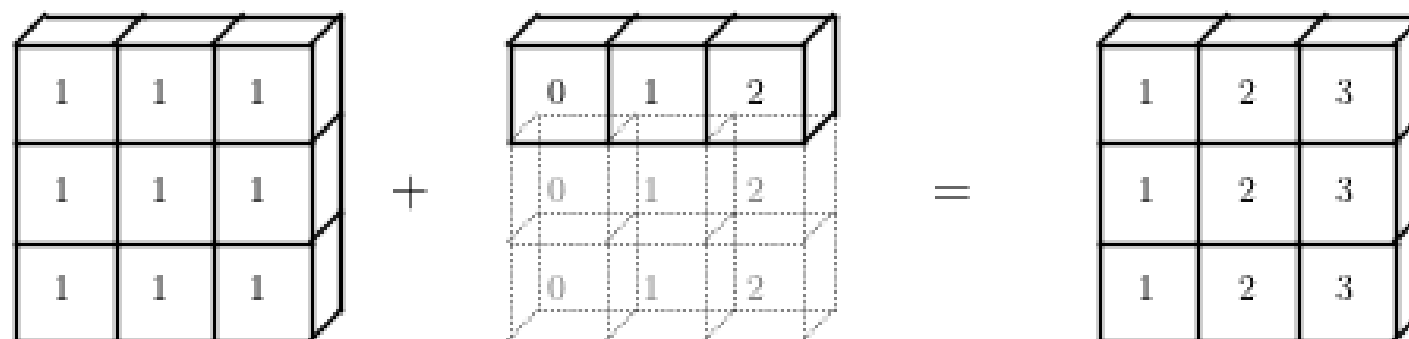
# Broadcasting

- 대부분의 연산시 동일한 크기의 ndarray를 기대함
- 만일, 크기가 일치하지 않고 일정 조건만 충족되면 broadcasting이 적용됨
  - 모양이 다른 배열끼리의 연산도 가능하게 해주며
  - 모양이 부족한 부분은 확장하여 연산을 수행
- Broadcasting 조건
  - 두 배열 중 하나의 배열이 1차원인 경우
  - 축의 크기가 동일한 경우

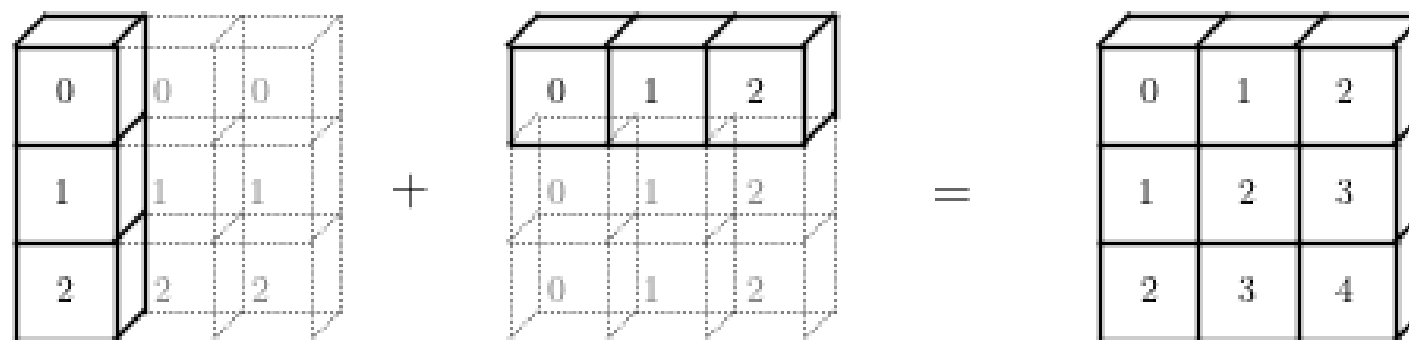
`np.arange(3) + 5`



`np.ones((3, 3)) + np.arange(3)`



`np.arange(3).reshape((3, 1)) + np.arange(3)`



3차원 배열  
(3,4,2)

		16	17
	8	9	
0	1		19
		11	
2	3		21
		13	
4	5		23
		15	
6	7		

2차원 배열  
(4,2)

		0	1
	0	1	
0	1		3
		3	
2	3		5
		5	
4	5		7
		7	
6	7		

+



3차원 배열  
(3,4,2)

		16	18
	8	10	
0	2		22
		14	
4	6		26
		18	
8	10		30
		22	
12	14		

브로드캐스팅  
Broadcasting

출처 :  
<https://sacko.tistory.com/16>

■ 예 : 두 배열 중 하나의 배열이 1차원인 경우

```
import numpy as np
```

```
a = np.arange(3)
```

```
print(a, '\n')
```

```
b = a + 5
```

```
print(b)
```

```
[0 1 2]
```

```
[5 6 7]
```

`np.arange(3) + 5`



## ■ 예 : 축의 크기가 동일한 경우

```
import numpy as np
```

```
a = np.ones((3, 3))
```

```
print(a, '\n')
```

```
b = np.arange(1, 4, dtype=float)
```

```
print(b, '\n')
```

```
c = a + b
```

```
print(c)
```

`np.ones((3, 3)) + np.arange(3)`

1	1	1
1	1	1
1	1	1

+

0	1	2
0	1	2
0	1	2

=

1	2	3
1	2	3
1	2	3

```
[[1. 1. 1.]  
 [1. 1. 1.]  
 [1. 1. 1.]]
```

```
[1. 2. 3.]
```

```
[[2. 3. 4.]  
 [2. 3. 4.]  
 [2. 3. 4.]]
```



## ■ 예 : 축의 크기가 동일한 경우

```
import numpy as np
```

```
a = np.arange(3).reshape(3,1)
```

```
print(a, '\n')
```

```
b = np.arange(3) # (1,3)
```

```
print(b, '\n')
```

```
c = a + b # (3,3)
```

```
print(c)
```

`np.arange(3).reshape((3, 1)) + np.arange(3)`

0	0	0
1	1	1
2	2	2

+

0	1	2
0	1	2
0	1	2

=

0	1	2
1	2	3
2	3	4

```
[[0]  
[1]  
[2]]
```

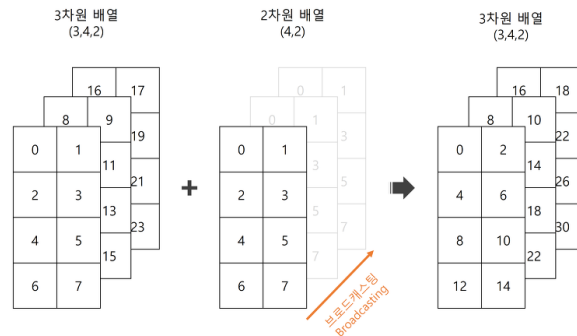
```
[0 1 2]
```

```
[[0 1 2]  
[1 2 3]  
[2 3 4]]
```

## ■ 예 : 행, 열 축의 크기가 동일한 경우

```
import numpy as np
```

```
a = np.arange(24).reshape(3,4,2)
print(f"a = \n{a}", '\n')
b = np.arange(8).reshape(4,2)
print(f"b = \n{b}", '\n')
c = a + b                                     # (3,4,2)
print(f"a + b = \n{c}")
```



```
a =
[[[ 0  1]
   [ 2  3]
   [ 4  5]
   [ 6  7]]
```

```
[[ 8  9]
 [10 11]
 [12 13]
 [14 15]]
```

```
[[16 17]
 [18 19]
 [20 21]
 [22 23]]]
```

```
b =
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
```

```
a + b =
[[[ 0  2]
   [ 4  6]
   [ 8 10]
   [12 14]]
```

```
[[ 8 10]
 [12 14]
 [16 18]
 [20 22]]
```

```
[[16 18]
 [20 22]
 [24 26]
 [28 30]]]
```

## ■ 예 :

```
import numpy as np

a = np.arange(5).reshape(1,1,5)    # (1, 1, 5)
print(a, '\n')
b = a + 5
print(b, '\n')
c = np.arange(5).reshape(5,1)      # (5, 1)
print(c, '\n')
d = a + np.arange(5).reshape(5,1)  # (1, 5, 5)
print(d)
```

```
[[[0 1 2 3 4]]]
```

```
[[[5 6 7 8 9]]]
```

```
[[0]
```

```
[1]
```

```
[2]
```

```
[3]
```

```
[4]]
```

```
[[[0 1 2 3 4]
```

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

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

```
[3 4 5 6 7]
```

```
[4 5 6 7 8]]]
```

# Indexing과 Slicing

## ■ Indexing

- python의 인덱싱과 비슷함
- index는 0부터 시작
- index생략시 all을 의미

## ■ Slicing

- python의 슬라이싱과 비슷함

## ■ indexing/slicing rule

### ■ **[i:j:k]**

- i:start, j:end(j-1위치), k:step

## ■ 1차원 Indexing

```
import numpy as np
```

```
a = np.arange(10) # 0 ~ 9
```

```
print(a[3]) # 3번째 있는 원소
```

```
print(a[:]) # 전체 원소 출력
```

```
print(a[:3]) # 처음부터 3번 원소 앞까지
```

```
print(a[:-1]) # 처음부터 마지막-1 까지
```

```
# 인덱스 2부터 2개씩(step)
```

```
print(a[2::2])
```

```
# 인덱스 7부터 2번 원소 앞까지 -1씩(step)
```

```
print(a[7:2:-1])
```

```
# 마지막부터 처음까지 -1씩(step)
```

```
print(a[::-1])
```

→ 3

→ [0 1 2 3 4 5 6 7 8 9]

→ [0 1 2]

→ [0 1 2 3 4 5 6 7 8]

→ [2 4 6 8]

→ [7 6 5 4 3]

→ [9 8 7 6 5 4 3 2 1 0]

## ■ 1차원 Slicing

```
import numpy as np

a = np.arange(10)    # 0 ~ 9
print(a)
s = a[5:8]           # 주의: referencing
print(s)             # [5 6 7]
s[:] = 500           # 전체 수정 (broadcasting)
print(s)
print(a)
```

```
[0 1 2 3 4 5 6 7 8 9]
[5 6 7]
[500 500 500]
[ 0  1  2  3  4 500 500 500  8  9]
```

## ■ 2차원 Indexing & Slicing

```
import numpy as np

a = np.arange(20).reshape((5,4))
print(a)
print(f"a[2,3] = {a[2,3]}")
print(f"a[0:5, 1] = {a[0:5, 1]}")
print(f"a[:, 1] = {a[:, 1]}")
print(f"a[1:3, :]=\n{a[1:3, :]}")

print(f"a[-1] = {a[-1]}")
print(f"a[:, -1] = {a[:, -1]}")
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
a[2,3] = 11
a[0:5, 1] = [ 1  5  9 13 17]
a[:, 1] = [ 1  5  9 13 17]
a[1:3, :]=
[[ 4  5  6  7]
 [ 8  9 10 11]]
a[-1] = [16 17 18 19]
a[:, -1] = [ 3  7 11 15 19]
```

## ■ copy() : ndarray의 깊은 복사

```
import numpy as np

a = np.arange(0, 3.6, 0.3).reshape((3,4))
print("a = \n", a)
b = a.copy()
print("copy of a = \n", b)
```

```
a =
[[ 0.   0.3  0.6  0.9]
 [ 1.2  1.5  1.8  2.1]
 [ 2.4  2.7  3.   3.3]]
copy of a =
[[ 0.   0.3  0.6  0.9]
 [ 1.2  1.5  1.8  2.1]
 [ 2.4  2.7  3.   3.3]]
```



## ■ fancy indexing

```
import numpy as np

a = np.arange(20).reshape((5,4))
print(a, '\n')

b = a[(0,2,3), 2] # fancy indexing
print(b)
```

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

[ 2 10 14]
```

## ■ boolean indexing

### ■ 하나의 축에만 적용 가능

```
import numpy as np
```

```
a = np.arange(20).reshape((5,4))  
print(a, '\n')
```

```
rows = np.array([True, False, True, True, False])  
b = a[rows, :] # boolean indexing  
print(b)
```

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

```
[[ 0  1  2  3]  
 [ 8  9 10 11]  
 [12 13 14 15]]
```

## ■ 반복을 이용한 item추출

### ■ item 추출(1)

```
import numpy as np

a = np.arange(24).reshape(2, 3, 4)
print(f"a =\n{a}")

for item in a:
    print(f"item =\n{item}")
```

```
a =
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]

item =
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

item =
[[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

## ■ 반복을 이용한 item추출

### ■ item 추출(2)

```
import numpy as np

a = np.arange(24).reshape(2, 3, 4)
print(f"a =\n{a}")

for i in range(len(a)):
    print(f"a[{i}] =\n{a[i]}")
```

```
a =
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]

a[0] =
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

a[1] =
[[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

## ■ 반복을 이용한 item추출

### ■ item 추출(3)

- flat를 이용하여 모든 item을 추출

```
import numpy as np
```

```
a = np.arange(24).reshape(2, 3, 4)
```

```
print(f"a =\n{a}")
```

```
for item in a.flat:  
    print(item, end=' ')
```

```
a =  
[[[ 0  1  2  3]  
  [ 4  5  6  7]  
  [ 8  9 10 11]]  
  
 [[12 13 14 15]  
  [16 17 18 19]  
  [20 21 22 23]]]  
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14  
15 16 17 18 19 20 21 22 23
```

# 배열 쌓기와 분리하기

## ■ 배열 쌓기

- `vstack()` : 수직으로 배열 연결
- `hstack()` : 수평으로 배열 연결
- 주의: 수직 또는 수평의 크기가 같아야 함
  - 참고: `a.c_[x, y]` # `x`와 `y`를 컬럼으로 간주하여 연결

## ■ 배열 분리하기

- `vsplit()` : 수직으로 배열 분리
- `hsplit()` : 수평으로 배열 분리
- 주의: 분리시에 수직 또는 수평으로 동일한 크기로 분리될 수 있어야 함

## ■ vstack()

```
import numpy as np

a = np.arange(8, dtype=float).reshape(2, 4)
b = np.full((3, 4), 2.0)
c = np.ones((2, 4))
print(f"a =\n{a}")
print(f"b =\n{b}")
print(f"c =\n{c}")

d = np.vstack((a, b, c)) # tuple
print(f"d =\n{d}")
```

```
a =
[[0. 1. 2. 3.]
 [4. 5. 6. 7.]]
b =
[[2. 2. 2. 2.]
 [2. 2. 2. 2.]
 [2. 2. 2. 2.]]
c =
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
d =
[[0. 1. 2. 3.]
 [4. 5. 6. 7.]
 [2. 2. 2. 2.]
 [2. 2. 2. 2.]
 [2. 2. 2. 2.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

## ■ hstack()

```
import numpy as np

a = np.arange(8, dtype=float).reshape(2, 4)
b = np.full((3, 4), 2.0)
c = np.ones((2, 4))
print(f"a =\n{a}")
print(f"b =\n{b}")
print(f"c =\n{c}")

d = np.hstack((a, c)) # tuple
print(f"d =\n{d}")
```

- b는 차수가 달라 연결할 수 없음.

```
a =
[[0. 1. 2. 3.]
 [4. 5. 6. 7.]]
b =
[[2. 2. 2. 2.]
 [2. 2. 2. 2.]
 [2. 2. 2. 2.]]
c =
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
d =
[[0. 1. 2. 3. 1. 1. 1. 1.]
 [4. 5. 6. 7. 1. 1. 1. 1.]]
```



## ■ vsplit()

```
import numpy as np

a = np.arange(24, dtype=float).reshape(6, 4)
print(f"a =\n{a}")

v1, v2 = np.vsplit(a, 2)
print(f"v1 =\n{v1}")
print(f"v2 =\n{v2}")
```

```
a =
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
 [12. 13. 14. 15.]
 [16. 17. 18. 19.]
 [20. 21. 22. 23.]]

v1 =
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]]

v2 =
[[12. 13. 14. 15.]
 [16. 17. 18. 19.]
 [20. 21. 22. 23.]]
```

## ■ hsplit()

```
import numpy as np

a = np.arange(24, dtype=float).reshape(6, 4)
print(f"a =\n{a}")

h1, h2 = np.hsplit(a, 2)
print(f"h1 =\n{h1}")
print(f"h2 =\n{h2}")
```

```
a =
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
 [12. 13. 14. 15.]
 [16. 17. 18. 19.]
 [20. 21. 22. 23.]]

h1 =
[[ 0.  1.]
 [ 4.  5.]
 [ 8.  9.]
 [12. 13.]
 [16. 17.]
 [20. 21.]]

h2 =
[[ 2.  3.]
 [ 6.  7.]
 [10. 11.]
 [14. 15.]
 [18. 19.]
 [22. 23.]]
```

# 선형대수 관련 연산

- 행렬의 곱셈
  - `np.dot(A, B)` - 앞에서 다루었음
- 단위행렬 만들기
  - `np.eye(n)` -  $n \times n$ 의 단위행렬
- 전치행렬
  - `np.transpose(a, s)`
- 대각원소와 대각합
  - `np.diag()`, `np.trace()`
- 역행렬과 행렬식
  - `np.linalg.inv()`, `np.linalg.det()`
- 고유값과 고유벡터
  - `np.linalg.eig()`

## ■ 단위행렬 만들기

- `np.eye(n)` :  $n \times n$ 의 단위행렬 생성
- 행렬과 그 행렬의 역행렬을 곱하면 단위행렬이 됨
- `np.linalg.inv(a)` :  $a$ 의 역행렬

```
import numpy as np
import numpy.linalg as np1

a = np.eye(3)
print("a = \n", a)

b = np.array([[1,3], [2,4]])
print("b = \n", b)
print("inverse matrix of b = \n", np1.inv(b))
```

```
a =
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
b =
[[1 3]
 [2 4]]
inverse matrix of b =
[[-2.  1.5]
 [ 1. -0.5]]
```

## ■ 전치행렬

### ■ a.T

```
import numpy as np

a = np.arange(12).reshape(3, 4)
b = np.arange(24).reshape(2, 4, 3)
print(a, '\n')
print(a.T, '\n')
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

### ■ 3차원 행렬에 적용하면

- 0, 1, 2번 축이 2, 1, 0번 축으로 전치됨

## ■ 전치행렬

■ `np.transpose(a[, s])`

```
import numpy as np

a = np.arange(24).reshape(2, 4, 3)
print(a, '\n')
print(np.transpose(b, (2, 0, 1))) # (3, 2, 4)
```

- a의 shape이 (2, 4, 3)이고
- a의 축번호는 (0, 1, 2)임. transpose에 의해
- a의 축번호가 (2, 0, 1)로 바뀜.  
즉, **열->면**, **면->행**, **행->열**로 바뀌게 됨
- 따라서 shape (3, 2, 4)로 바뀌게 됨

```
[[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
[[12 13 14]
 [15 16 17]
 [18 19 20]
 [21 22 23]]]
```

```
[[[ 0  3  6  9]
 [12 15 18 21]]
```

```
[[ 1  4  7 10]
 [13 16 19 22]]
```

```
[[ 2  5  8 11]
 [14 17 20 23]]]
```

## ■ 대각원소와 대각합

- np.diag()
- np.trace()

```
import numpy as np

a = np.arange(1, 10).reshape(3, 3)
print(a, '\n')
b = np.diag(a)
print(b, '\n')
c = np.trace(a)
print(c)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
[1 5 9]
```

```
15
```

## ■ 역행렬과 행렬식

- `linalg.inv()`

- `linalg.det()`

```
import numpy as np

a = np.array([[ 1,  2,  3],
               [ 5,  7, 11],
               [21, 29, 31]])

print(a, '\n')
b = np.linalg.inv(a)
print(b, '\n')
c = np.linalg.det(a)
print(c)
```

```
[[ 1  2  3]
 [ 5  7 11]
 [21 29 31]]

[[-2.31818182  0.56818182  0.02272727]
 [ 1.72727273 -0.72727273  0.09090909]
 [-0.04545455  0.29545455 -0.06818182]]

43.99999999999997
```



## ■ 고유값과 고유벡터

### ■ `linalg.eig(a)`

```
import numpy as np
```

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

```
print(f"a =\n{a}")
```

```
eigenvalues, eigenvectors = np.linalg.eig(a)
```

```
print(f"eigenvalues =\n{eigenvalues}")
```

```
print(f"eigenvectors =\n{eigenvectors}")
```

```
a =  
[[4 2]  
 [3 5]]  
eigenvalues =  
[2. 7.]  
eigenvectors =  
[[-0.70710678 -0.5547002 ]  
 [ 0.70710678 -0.83205029]]
```

### (3) matplotlib 라이브러리

# matplotlib

## ■ 데이터의 시각화를 위한 라이브러리

- matlab은 유료
- 인공지능, 빅데이터, 웹, 수학, 과학 등에서 데이터분석을 위해 사용가능

## ■ 사용방법

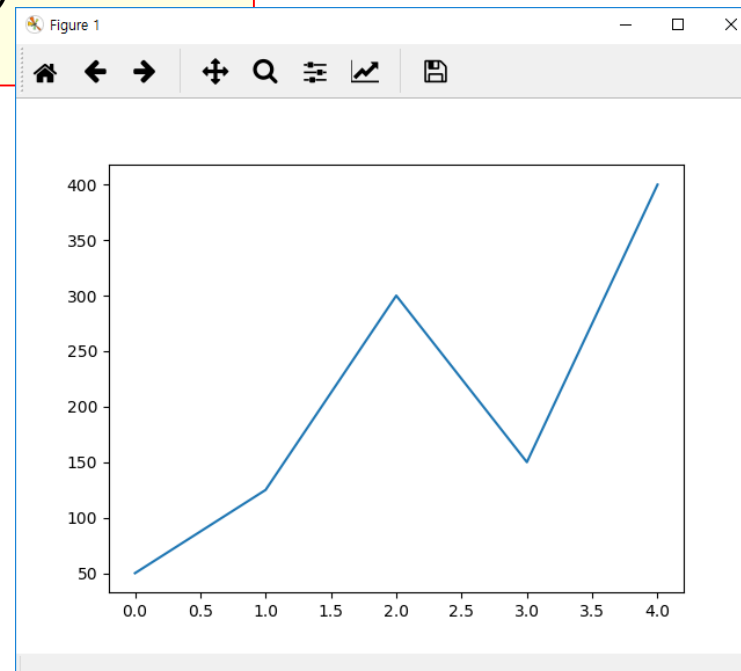
- 1. 라이브러리 import
  - `import matplotlib.pyplot as plt`
- 2. 데이터값 설정
  - x, y 데이터 설정(1개만 입력시 y로 간주하고, x는 자동생성)
- 3. 데이터 plotting

## ■ (예) 선그래프 그리기

```
import numpy as np
import matplotlib.pyplot as plt

data = [50, 125, 300, 150, 400]

plt.plot(range(len(data)), data)
plt.show()
```

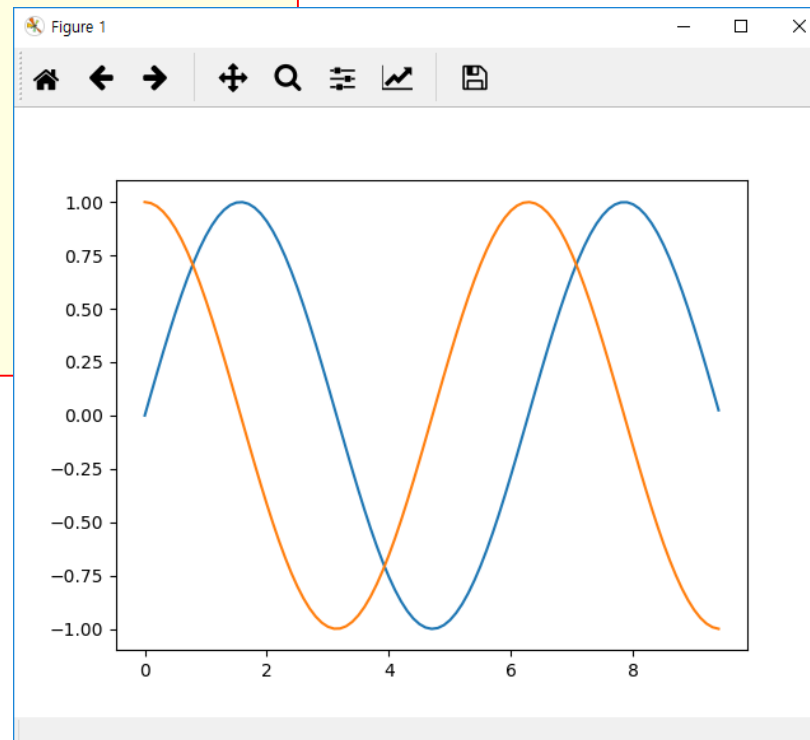


## ■ (예) sin, cos 곡선 그리기

```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.arange(0, 3*np.pi, 0.1)
y1 = np.sin(x)
y2 = np.cos(x)
```

```
plt.plot(x, y1)
plt.plot(x, y2)
plt.show()
```

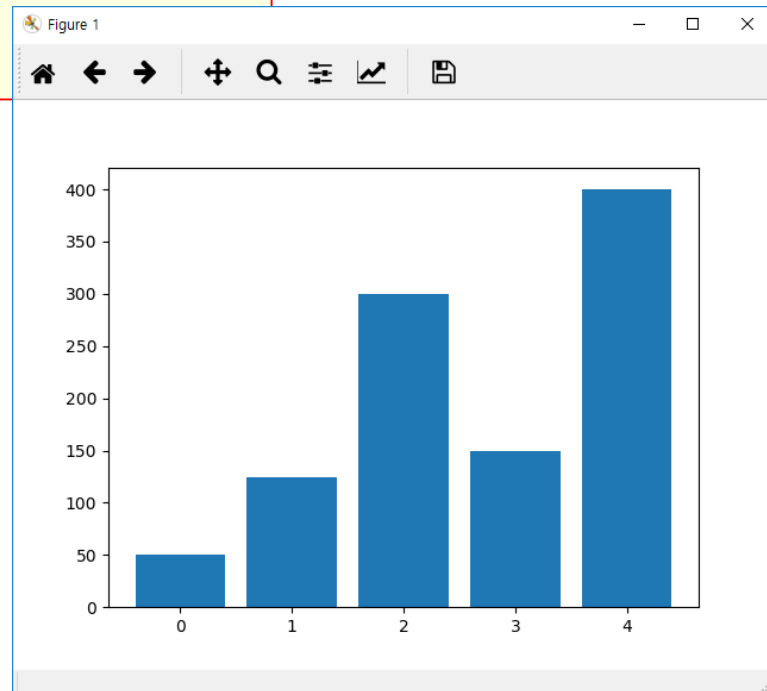


## ■ (예) 막대그래프 그리기

```
import numpy as np
import matplotlib.pyplot as plt

data = [50, 125, 300, 150, 400]

plt.bar(range(len(data)), data)
plt.show()
```



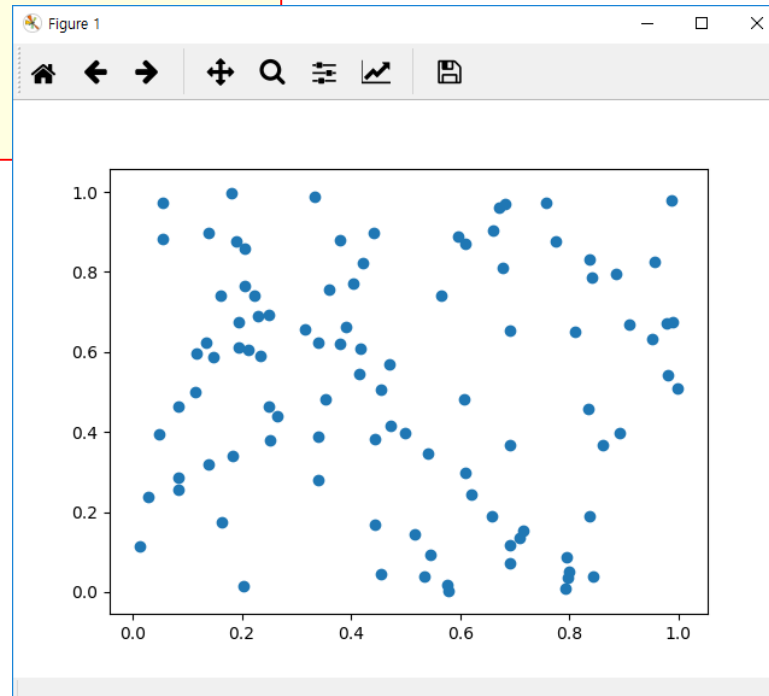
## ■ (예) scatter point 찍기

```
import numpy as np
import matplotlib.pyplot as plt
```

```
p = np.random.rand(100, 2)
```

```
plt.scatter(p[:,0], p[:,1])
plt.show()
```

100행 2열의 행렬

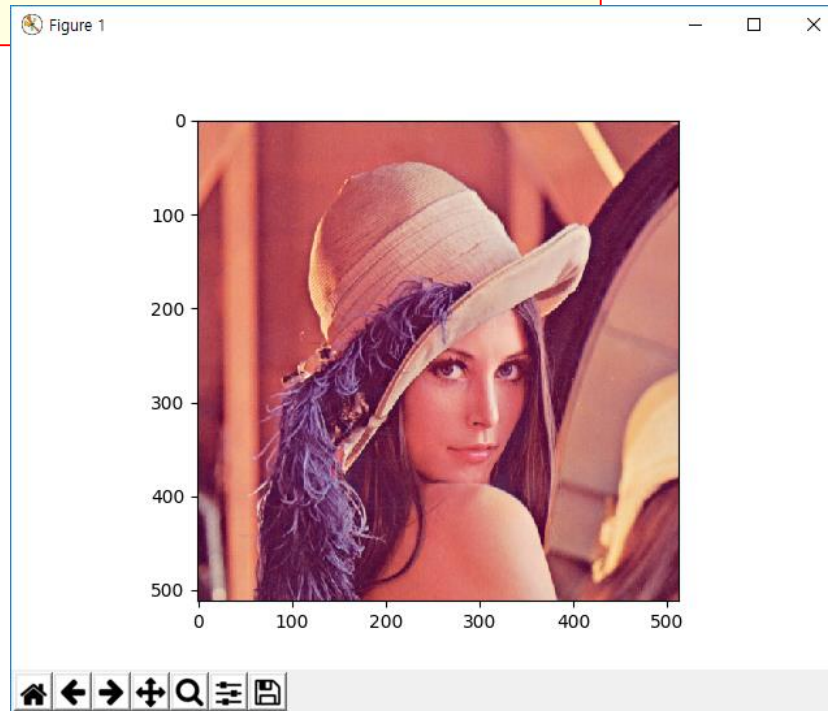


## ■ (예) 이미지 그리기

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("lena.jpg", cv2.IMREAD_COLOR)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

cv2는 BGR 컬러이므로  
RGB로 바꾸어 출력





# 제목 및 레이블 달기

## ■ (예) 선 그래프 그리기

```
import matplotlib.pyplot as plt
```

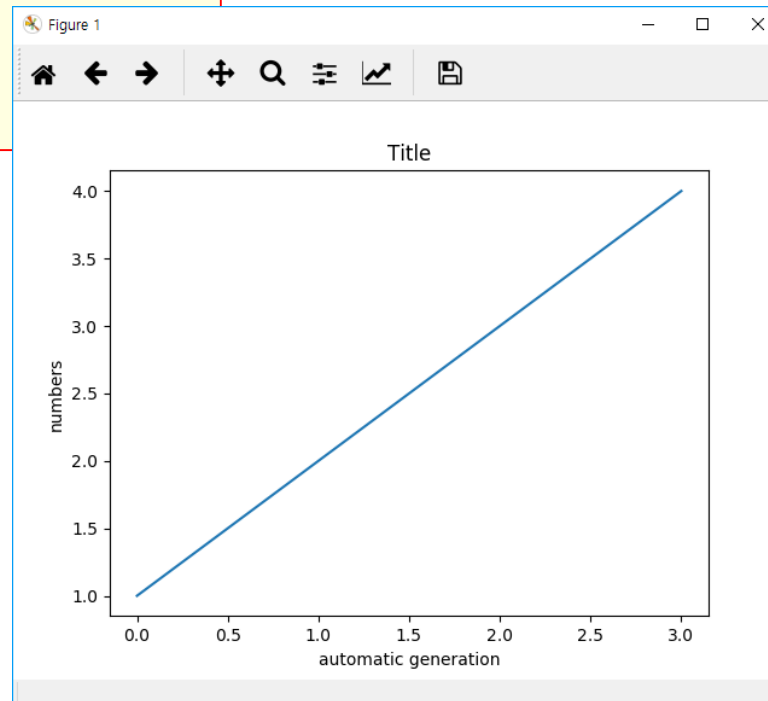
```
plt.title('Title')
```

```
plt.xlabel('automatic generation')
```

```
plt.ylabel('numbers')
```

```
plt.plot([1,2,3,4])
```

```
plt.show()
```



# Line plot & markers

## ■ Plot()

- `plot(x, y)`                   # 기본적인 스타일과 컬러
- `plot(x, y, 'bo')`       # blue circle markers
- `plot(y)`                   # `x = [0...N-1]`
- `plot(y, 'r+')`           # red plus markers
- `plot(x1, y1, 'g^', x2, y2, 'g-.'`)  
  # green triangle\_up marker & green dash-dot line style

# Styles

---

## ■ Line Styles

- ‘\_’ : solid line
- ‘--’ : dashed line
- ‘-.’ : dash-dot line
- ‘:’ : dotted line

## ■ Color

- ‘b’ : blue
- ‘g’ : green
- ‘r’ : red
- ‘c’ : cyan
- ‘m’ : magenta
- ‘y’ : yellow
- ‘k’ : black
- ‘w’ : white

## ■ Markers

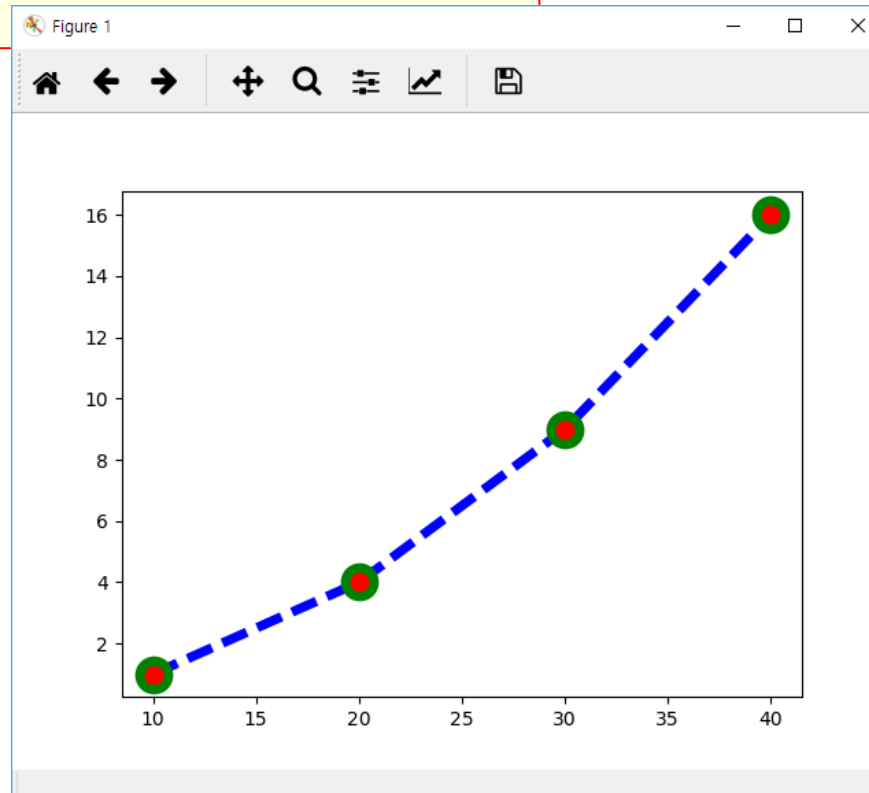
- `'.'` : point marker
- `','` : pixel marker
- `'o'` : circle marker
- `'v'` : triangle\_down marker
- `'^'` : triangle\_up marker
- `'<'` : triangle\_left marker
- `'>'` : triangle\_right marker
- `'s'` : square marker
- `'*'` : star marker
- `'+'` : plus marker
- `'x'` : x marker
- `'D'` : diamond marker
- `'d'` : thin\_diamond marker

## ■ 기타 스타일

- `color(c)` (ex) `c="r"`
- `linewidth(lw)` (ex) `lw=3`
- `linestyle(ls)` (ex) `ls="--"`
- `marker` (ex) `marker="o"`
- `markersize(ms)` (ex) `ms=15`
- `markeredgecolor(mec)`
- `markeredgewidth(mew)`
- `markerfacecolor(mfc)`

```
import matplotlib.pyplot as plt

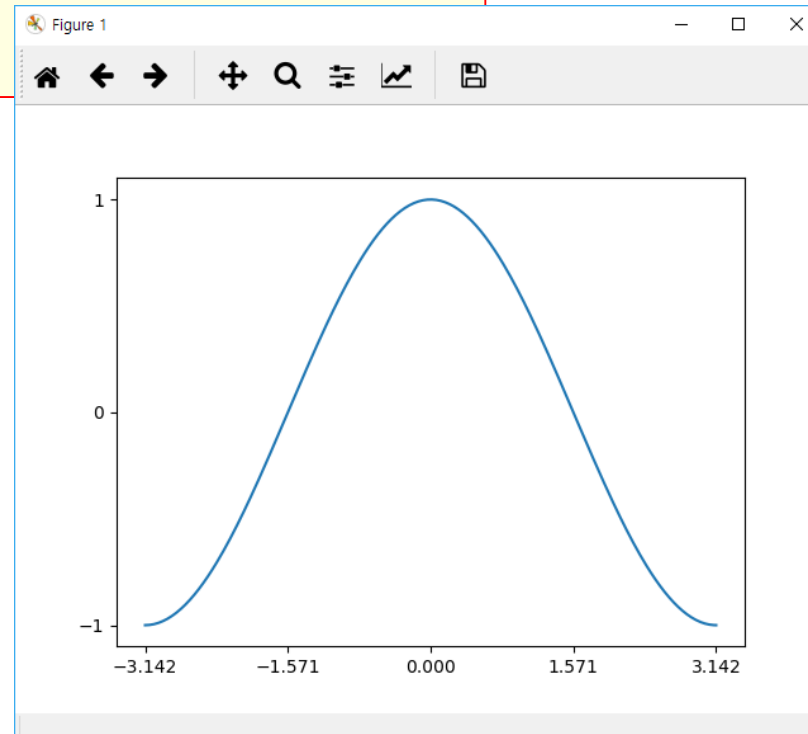
plt.plot([10, 20, 30, 40], [1, 4, 9, 16],
         c="b", lw=5, ls="--",
         marker="o", ms=15, mec="g", mew=5, mfc="r")
plt.show()
```



# Tick 설정하기

```
import numpy as np
import matplotlib.pyplot as plt

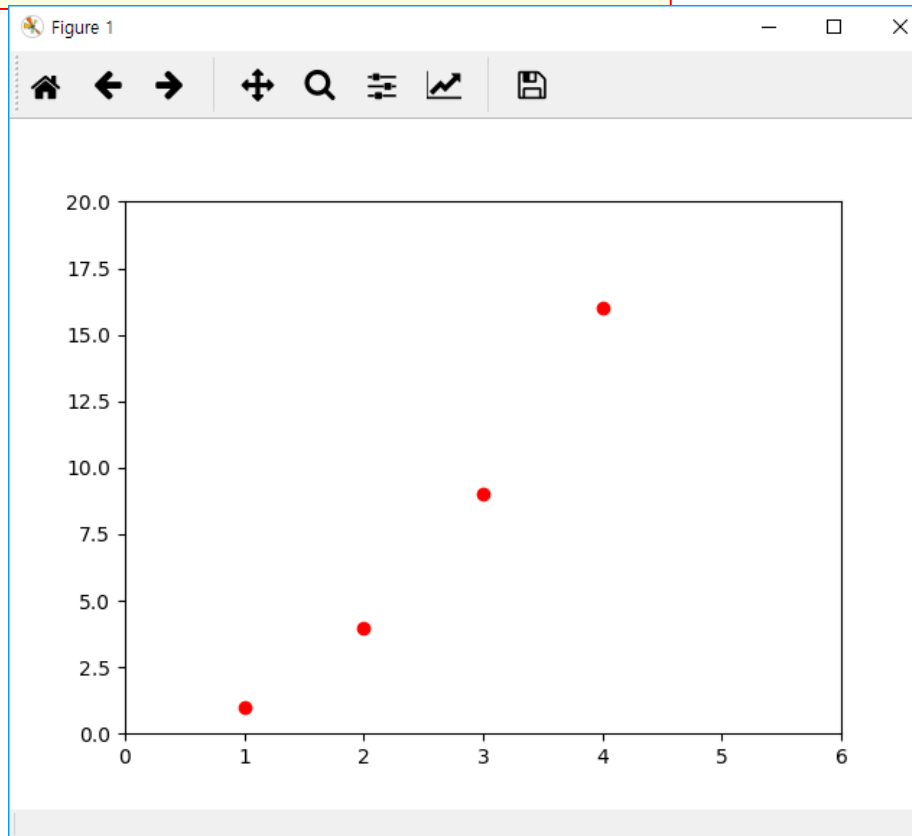
X = np.linspace(-np.pi, np.pi, 256)
C = np.cos(X)
plt.plot(X, C)
plt.xticks([-np.pi, -np.pi / 2, 0, np.pi / 2, np.pi])
plt.yticks([-1, 0, +1])
plt.show()
```



# Set min-max axes

```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20]) # [xmin, xmax, ymin, ymax]
plt.show()
```





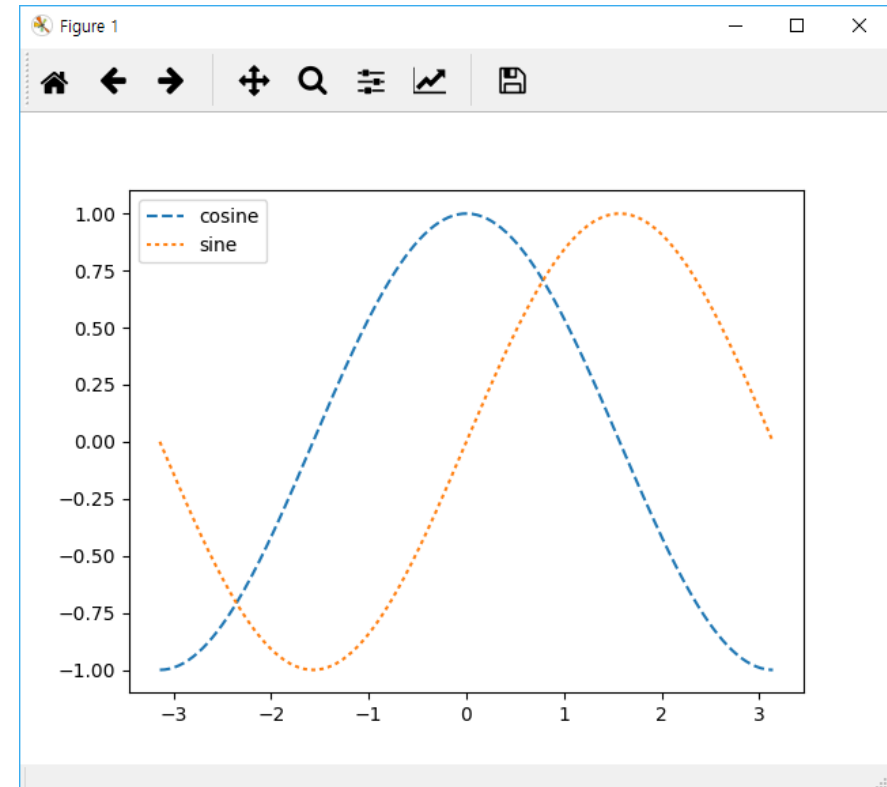
# 범례(legend)

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, ls="--", label="cosine")
plt.plot(X, S, ls=":", label="sine")
plt.legend(loc=2)
plt.show()
```

■ `loc=(0은 best, 5는 right)`

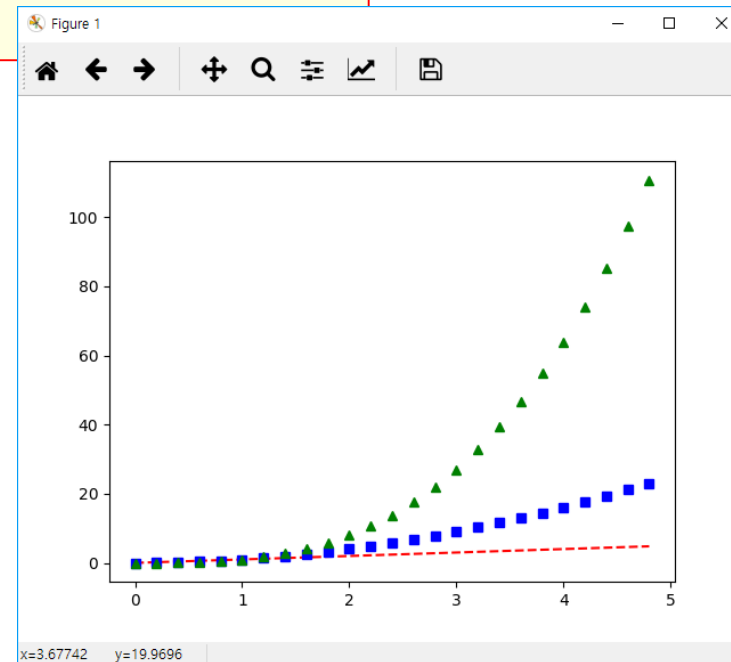
2	9	1
6	10	7
3	8	4



## ■ (예) 한꺼번에 plot하기(속성을 생략할 수 없음)

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

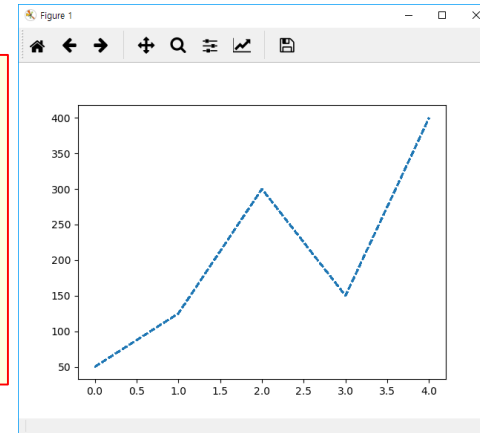


# Line의 속성 제어

## ■ anti-aliasing

```
import matplotlib.pyplot as plt

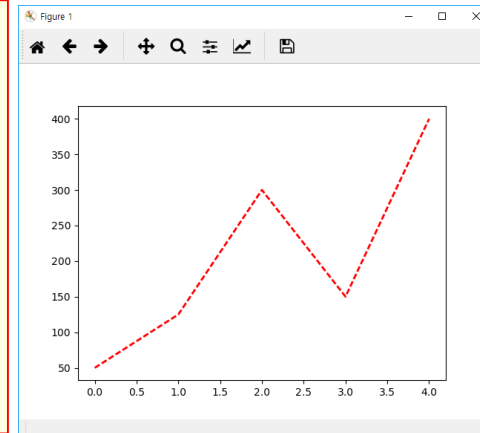
data = [50, 125, 300, 150, 400]
line, = plt.plot(range(len(data)), data, '--')
line.set_antialiased(False)
plt.show()
```



## ■ setp()

```
import matplotlib.pyplot as plt

data = [50, 125, 300, 150, 400]
line, = plt.plot(range(len(data)), data, '--')
plt.setp(line, color='r', linewidth=2.0)
# MATLAB style
# plt.setp(line, 'color', 'r', 'linewidth', 2.0)
plt.show()
```



# figure() & subplot()

- `plt.figure(n)`
  - 그림의 번호를 지정 (기본 1)
  - 기본그림에 새로운 그림을 추가
    - `plt.figure(2)`
- `plt.subplot(r, c, i)`
- `plt.subplot(rci)`
  - `r`: rows, `c`: columns, `i`: index(1부터 시작)
  - (예) 2행x3열, 첫번째 영역
    - `plt.subplot(2,3,1)`
    - `plt.subplot(231)`

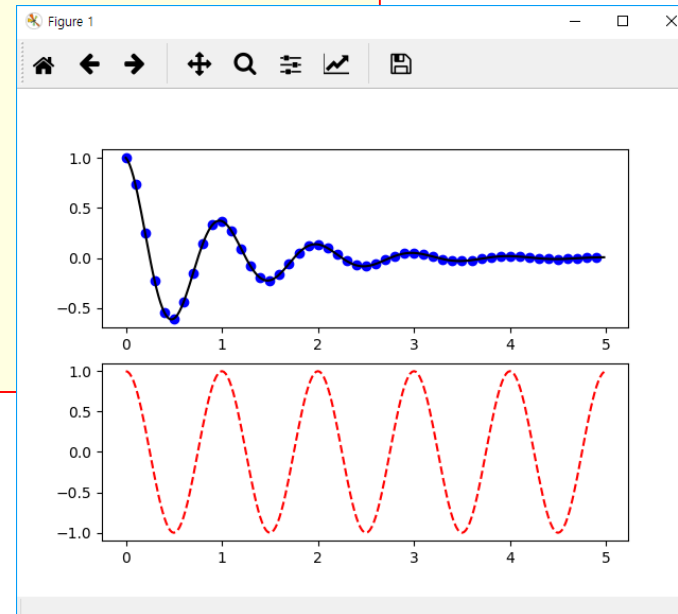
```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1) # 생략가능
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

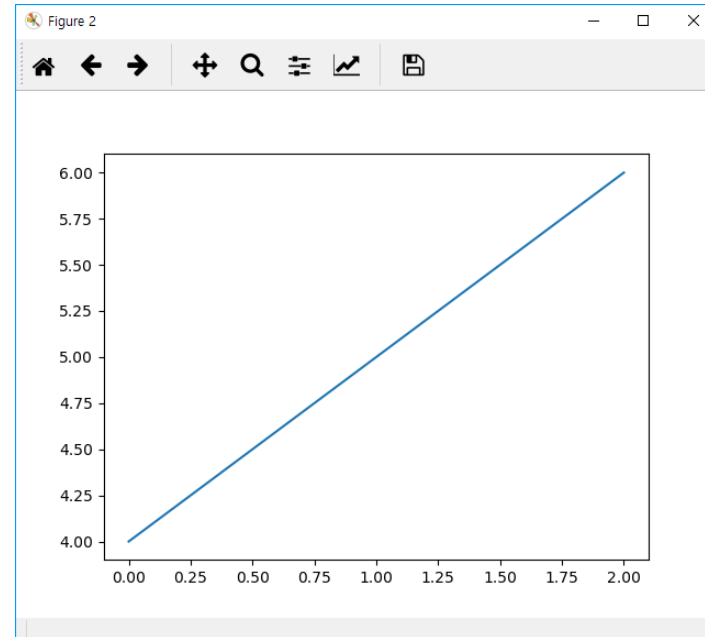
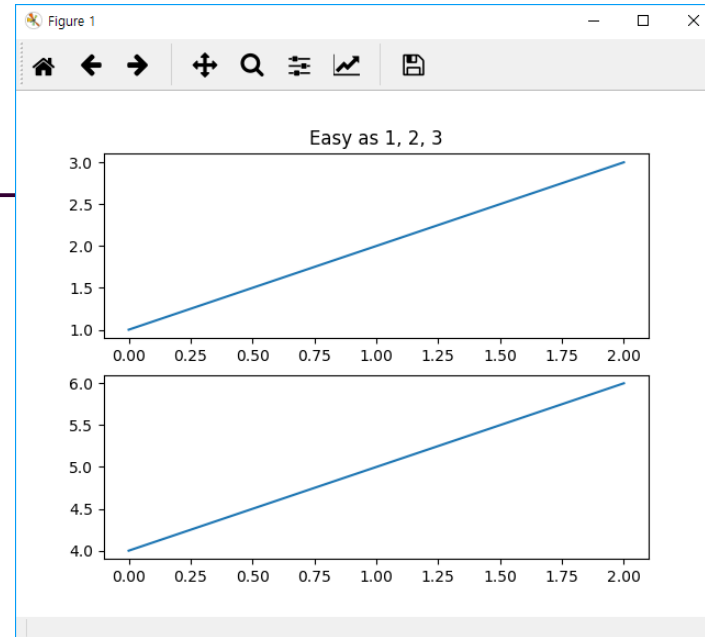


```
import matplotlib.pyplot as plt
plt.figure(1)                # first figure
plt.subplot(211)
plt.plot([1, 2, 3])
plt.subplot(212)
plt.plot([4, 5, 6])

plt.figure(2)                # second figure
plt.plot([4, 5, 6])

plt.figure(1)                # figure 1
plt.subplot(211)
plt.title('Easy as 1, 2, 3')

plt.show()
```



# 한글사용하기

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

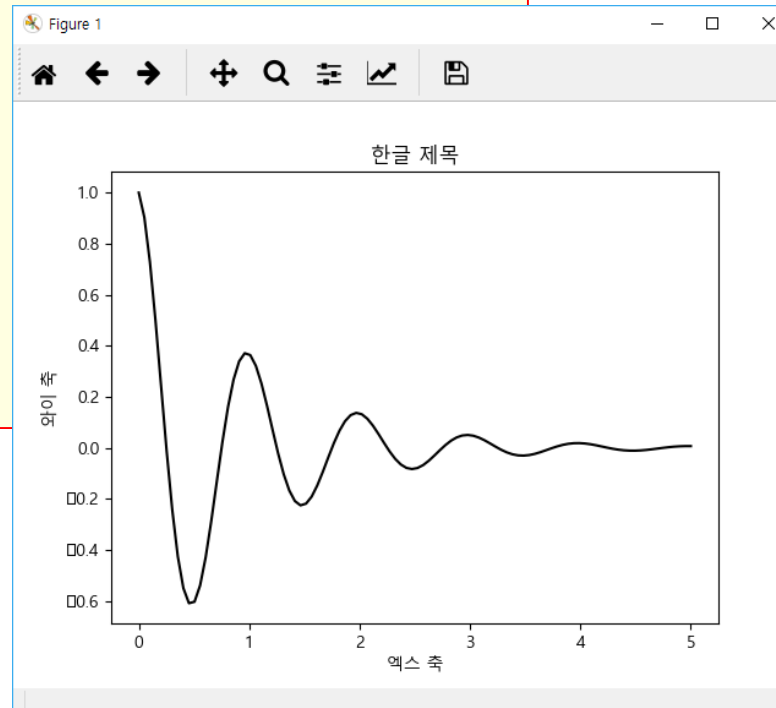
# matplotlib 한글 폰트 설정
mpl.rc('font', family='D2Coding')
mpl.rc('axes', unicode_minus=False) # 유니코드에서 음수 부호설정

x = np.linspace(0.0, 5.0, 100)
y = np.cos(2 * np.pi * x) * np.exp(-x)

plt.plot(x, y, 'k')
plt.title('한글 제목')
plt.xlabel('엑스 축')
plt.ylabel('와이 축')

plt.show()
```

맑은고딕체는  
'Malgun Gothic'



# 텍스트 작업

```
import numpy as np
import matplotlib.pyplot as plt

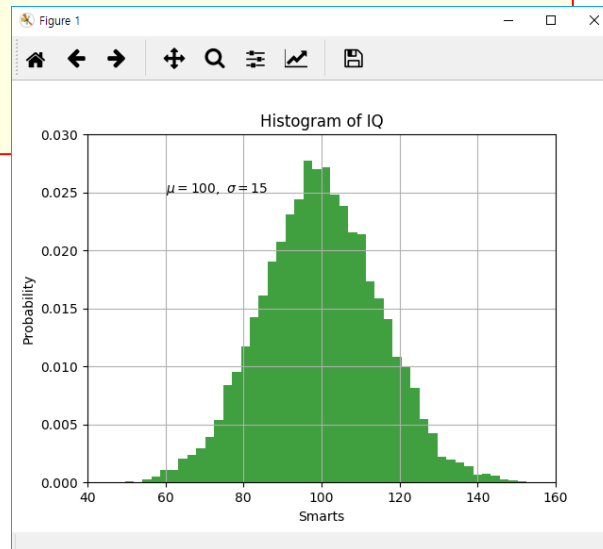
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100, \sigma=15$') # LaTeX문자열
plt.axis([40, 160, 0, 0.03])
plt.grid(True)

plt.show()
```

- **n** : 각 구간의 frequency
- **bins** : 구간
- **LaTeX**

<http://matplotlib.org/1.3.1/users/mathtext.html#mathtext-tutorial>





# 텍스트 주석 달기

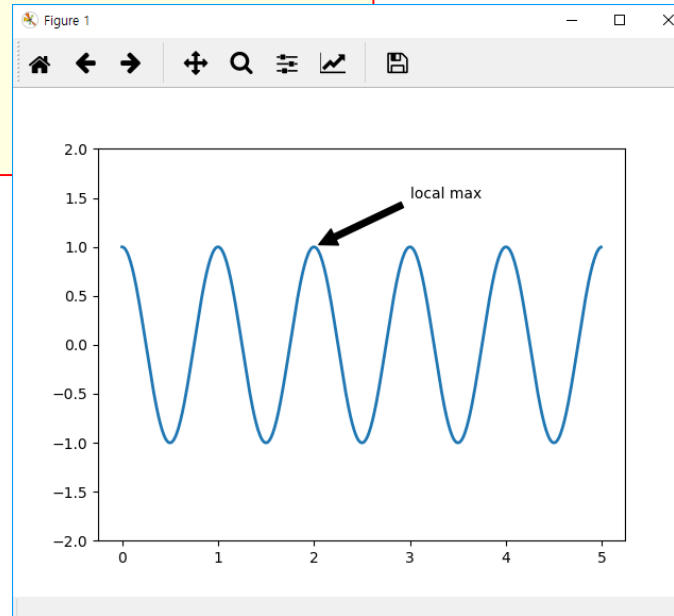
```
import numpy as np
import matplotlib.pyplot as plt

ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )

plt.ylim(-2, 2)
plt.show()
```



# 등고선 그리기(contourf)

## ■ `contourf([X, Y,] Z, [levels])`

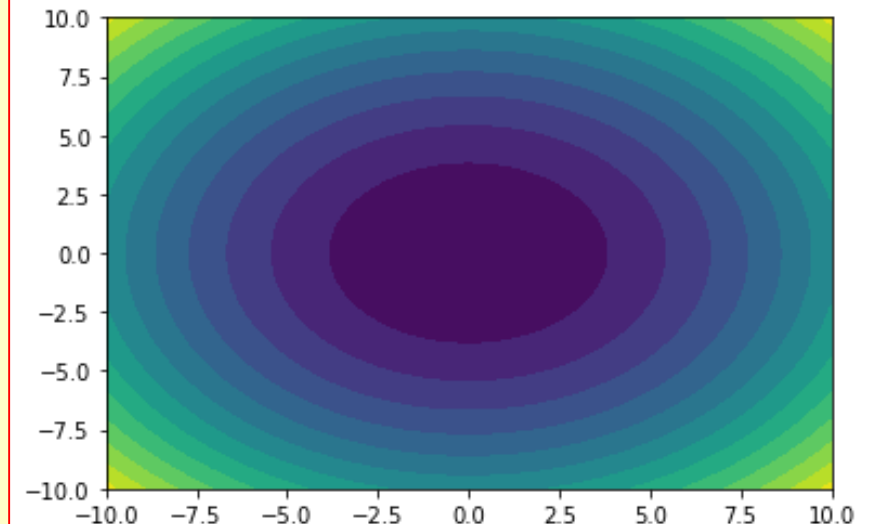
- `X, Y` : `Z`의 좌표를 나타내는 2차원 `array(shape: M, N)`
  - `X, Y`는 보통 `np.meshgrid()`로 생성됨
- `Z` : 높이(height)를 나타내는 2차원 `array(shape: M, N)`
- `levels` : 몇 단계로 나타낼지를 나타내는 것(int 또는 array)

```
import numpy as np
import matplotlib.pyplot as plt

def f(x, y):
    return x**2 + y**2

X, Y = np.meshgrid(np.linspace(-10, 10, 100),
                    np.linspace(-10, 10, 100))
Z = f(X, Y)

plt.contourf(X, Y, Z, levels=15)
plt.show()
```



## ■ 참고: meshgrid()

```
import numpy as np

x = np.arange(1, 10)
y = np.arange(1, 10)
print(f'x =\n{x}')
print(f'y =\n{y}')

xx, yy = np.meshgrid(x, y)
print(f'xx =\n{xx}')
print(f'yy =\n{yy}')
```

```
x =
[1 2 3 4 5 6 7 8 9]
y =
[1 2 3 4 5 6 7 8 9]
xx =
[[1 2 3 4 5 6 7 8 9]
 [1 2 3 4 5 6 7 8 9]
 [1 2 3 4 5 6 7 8 9]
 [1 2 3 4 5 6 7 8 9]
 [1 2 3 4 5 6 7 8 9]
 [1 2 3 4 5 6 7 8 9]
 [1 2 3 4 5 6 7 8 9]
 [1 2 3 4 5 6 7 8 9]
 [1 2 3 4 5 6 7 8 9]]
yy =
[[1 1 1 1 1 1 1 1 1]
 [2 2 2 2 2 2 2 2 2]
 [3 3 3 3 3 3 3 3 3]
 [4 4 4 4 4 4 4 4 4]
 [5 5 5 5 5 5 5 5 5]
 [6 6 6 6 6 6 6 6 6]
 [7 7 7 7 7 7 7 7 7]
 [8 8 8 8 8 8 8 8 8]
 [9 9 9 9 9 9 9 9 9]]
```

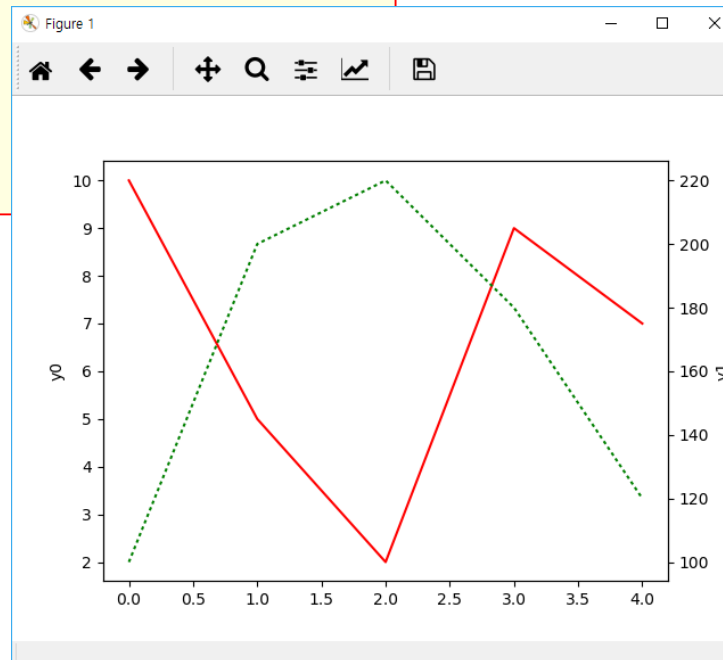
# 여러가지 plot를 하나로 표시

```
import matplotlib.pyplot as plt

fig, ax0 = plt.subplots()
ax1 = ax0.twinx()

ax0.plot([10, 5, 2, 9, 7], 'r-', label="y0")
ax0.set_ylabel("y0")
ax0.grid(False)
ax1.plot([100, 200, 220, 180, 120], 'g:', label="y1")
ax1.set_ylabel("y1")
ax1.grid(False)

plt.show()
```



# Axes3D scatterplot

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

# Fixing random state for reproducibility
np.random.seed(19680801)

def randrange(n, vmin, vmax):
    return (vmax - vmin)*np.random.rand(n) + vmin

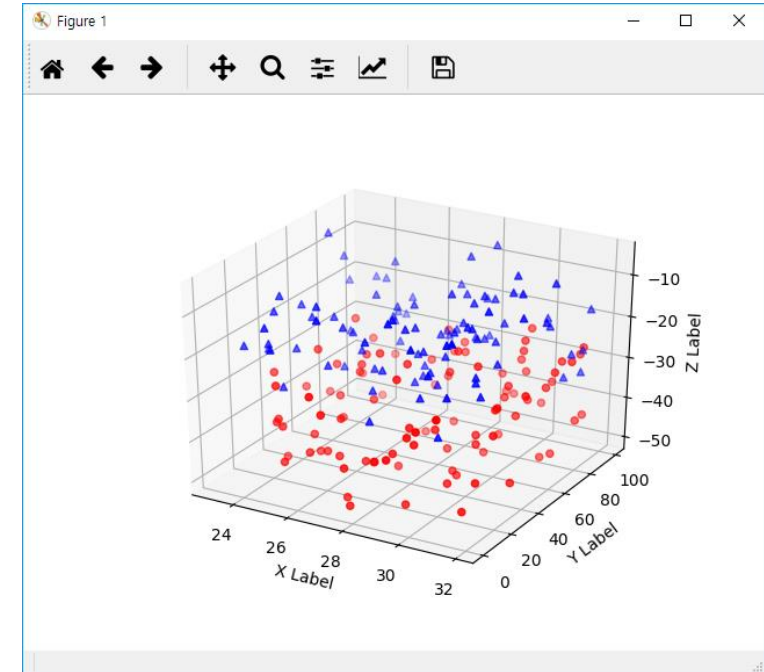
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

n = 100

# defined by x in [23, 32], y in [0, 100], z in [zlow, zhigh].
for c, m, zlow, zhigh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
    xs = randrange(n, 23, 32)
    ys = randrange(n, 0, 100)
    zs = randrange(n, zlow, zhigh)
    ax.scatter(xs, ys, zs, c=c, marker=m)

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
```



# 기타 참고 사이트

---

- 시각화 패키지 Matplotlib 소개

- <https://datascienceschool.net/view-notebook/d0b1637803754bb083b5722c9f2209d0/>

- 5 Quick and Easy Data Visualizations in Python with Code

- <https://towardsdatascience.com/5-quick-and-easy-data-visualizations-in-python-with-code-a2284bae952f>

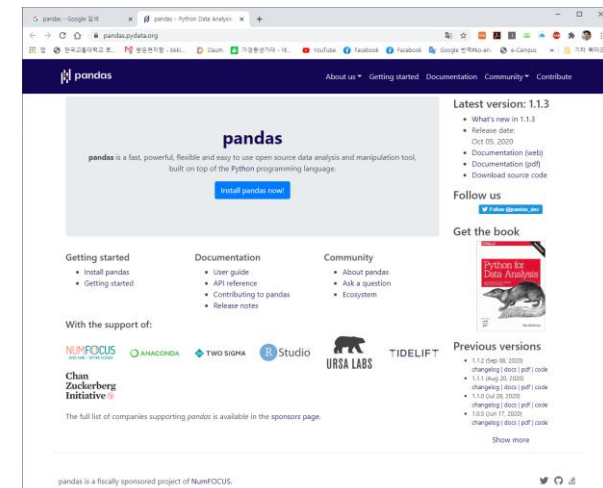
- The mplot3d Toolkit

- <https://matplotlib.org/tutorials/toolkits/mplot3d.html#toolkit-mplot3d-tutorial>

## (4) Pandas

# Pandas 라이브러리

- 효과적인 데이터 분석을 위한 고수준의 자료구조와 데이터 분석 도구를 제공
  - Python기반 데이터 과학자와 분석가에게 가장 중요한 도구
  - 모델링 또는 복잡한 시각화를 수행하기 전에 데이터 셋의 특성을 잘 이해하는데 유용함
  - numpy의 상위 패키지로 통계분석을 위한 scipy, 시각화를 위한 matplotlib, 기계학습을 위한 scikit-learn에 자주 사용됨
- 홈페이지
  - <https://pandas.pydata.org/>
- 설치방법
  - `pip install pandas`





## ■ 데이터 자료구조의 종류

### ■ Series

- 1차원 데이터를 다루는 데 효과적인 자료구조

### ■ DataFrame

- 행과 열로 구성된 2차원 데이터를 다루는 데 효과적인 자료구조

### ■ Panel

Series			Series			DataFrame		
	apples			oranges			apples	oranges
0	3		0	0		0	3	0
1	2	+	1	3	=	1	2	3
2	0		2	7		2	0	7
3	1		3	2		3	1	2

출처 : <https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/>

# Series(1차원)

## ■ Series 객체 생성(1) - by list

```
import pandas as pd

# Series 정의하기
obj = pd.Series([3, 6, -7, 2])
print(obj)
```

```
0    3
1    6
2   -7
3    2
dtype: int64
```

- Series 객체는 1차원 배열과 달리 값 뿐만 아니라 각 값에 연결된 인덱스 값도 동시에 저장
  - index는 별도로 지정 가능

## ■ Series 객체 생성(2) - by list

```
import numpy as np
import pandas as pd

# Series 정의하기
s = pd.Series([1, 3, 5, np.nan, 6, 8])
print(s)
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

★ NaN : Not a Number

## ■ Series의 속성

- values : 값
- index : 인덱스
- dtypes : 자료형

```
import numpy as np
import pandas as pd
```

```
# Series 정의하기
```

```
s = pd.Series([1, 3, 5, np.nan, 6, 8])
print(s.values)
print(s.index)
print(s.dtypes)
```

```
[ 1.  3.  5. nan  6.  8.]
RangeIndex(start=0, stop=6, step=1)
float64
```

## ■ Series index의 변경

- 기본적인 index를 원하는 index로 변경 가능

```
import numpy as np
import pandas as pd

# Series 정의하기(index 변경)
obj = pd.Series([3, 6, -7, 2], index=['a', 'd', 'c', 'b'])
print(obj)
```

```
a    3
d    6
c   -7
b    2
dtype: int64
```

## ■ Series 객체 생성(3) - by dictionary

```
import numpy as np
import pandas as pd

# Series 정의하기(dictionary 이용)
obj = pd.Series({'Kim': 32_000,
                 'Lee': 7_000,
                 'Park': 12_000,
                 'Choi': 14_000})

print(obj)
```

```
Kim      32000
Lee       7000
Park     12000
Choi     14000
dtype: int64
```

## ■ Series name과 index name의 변경

### ■ series와 index의 이름 변경 가능

```
import numpy as np
import pandas as pd

# Series 정의하기
obj = pd.Series([3, 6, -7, 2])
obj.name = 'Test data'
obj.index.name = 'No.'
print(obj)
```

```
No.
0    3
1    6
2   -7
3    2
Name: Test data, dtype: int64
```



```
0    3
1    6
2   -7
3    2
dtype: int64
```

# Series Examples

## ■ index를 이용하여 Series객체 생성

```
import pandas as pd

kakao = pd.Series([92600, 92400, 92100, 94300, 92300],
                  index=['2016-02-19', '2016-02-18',
                        '2016-02-17', '2016-02-16',
                        '2016-02-15'])

print(kakao)
```

```
2016-02-19    92600
2016-02-18    92400
2016-02-17    92100
2016-02-16    94300
2016-02-15    92300
dtype: int64
```



## ■ Series Indexing

```
import pandas as pd

a = pd.Series([926, 924, 921, 943, 923])
print(a[0])
print(a[1])
print(a[2])
print()

kakao = pd.Series([92600, 92400, 92100, 94300, 92300],
                  index=['2016-02-19', '2016-02-18', '2016-02-17',
                        '2016-02-16', '2016-02-15'])
print(kakao['2016-02-17'])
```

```
926
924
921

92100
```

## ■ Series operation by index

```
import pandas as pd

mine = pd.Series([10, 20, 30], index=['naver', 'sk', 'kt'])
friend = pd.Series([10, 30, 20], index=['kt', 'naver', 'sk'])

merge = mine + friend

print(merge)
```

```
kt      40
naver   40
sk      40
dtype: int64
```

mine

Series	
Index	Value
'naver'	10
'sk'	20
'kt'	30

friend

Series	
Index	Value
'kt'	10
'naver'	30
'sk'	20

# Series 연습문제

- 다음의 방법으로 3개의 Series를 자유롭게 생성하시오.
  - 리스트만으로 생성
  - 딕셔너리만으로 생성
  - 리스트와 index를 이용하여 생성
- 각 Series에 대해
  - 각 Series의 name을 SeriesA, SeriesB, SeriesC로 수정하시오.
  - 각 Series의 index name을 IndexA, IndexB, IndexC로 수정하시오
  - 각 Series에 대해 indexing을 통해 원하는 값을 선택하시오.
  - 각 Series에 2개씩 새로운 데이터를 추가하시오.
  - 각 Series의 index를 다르게 부여하시오.
    - 참고: `obj=obj.reindex([3, 1, 0, 2])` # index의 순서바꾸기
  - 각 Series에서 1개씩 데이터를 삭제하시오.(참고: `del`)
    - 참고: `obj=obj.drop('row1')` # index 'row1'에 해당하는 데이터 삭제

# DataFrame(2차원)

## ■ DataFrame 만들기(1) - by dictionary

```
import numpy as np
import pandas as pd

# DataFrame 정의하기(dictionary 이용)
data = {'Apples': [3, 2, 0, 1],
        'Oranges': [0, 3, 7, 2]}
df = pd.DataFrame(data)
print(df)
```

	Apples	Oranges
0	3	0
1	2	3
2	0	7
3	1	2

## ■ DataFrame 만들기(2) - by dictionary

```
import numpy as np
import pandas as pd

# DataFrame 정의하기(dictionary 이용)
data = {'Name': ['Lee', 'Lee', 'Choi', 'Kim', 'Park'],
        'Year': [2013, 2014, 2015, 2016, 2015],
        'Points': [1.5, 1.8, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data)
print(df)
```

	Name	Year	Points
0	Lee	2013	1.5
1	Lee	2014	1.8
2	Choi	2015	3.6
3	Kim	2016	2.4
4	Park	2015	2.9

## ■ DataFrame 만들기(3) - by dictionary

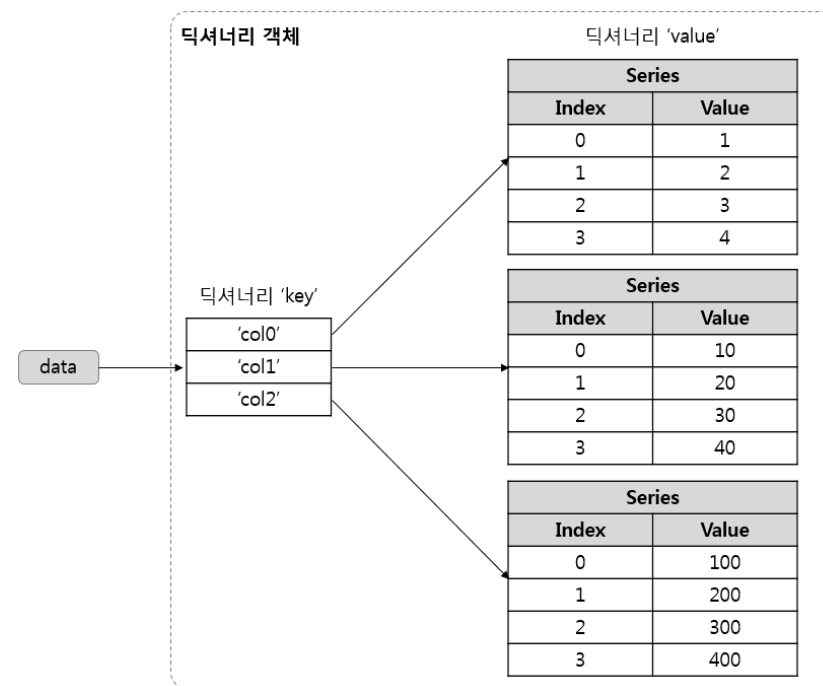
```
import pandas as pd

raw_data = {'col0': [1, 2, 3, 4],
            'col1': [10, 20, 30, 40],
            'col2': [100, 200, 300, 400]}

data = pd.DataFrame(raw_data)
print(data)
```

```
   col0  col1  col2
0     1    10   100
1     2    20   200
2     3    30   300
3     4    40   400
```

★주의: Series를 단순히 list로 묶으면 행(row)이 됨



DataFrame 구조

# 시계열 데이터 생성

- `pd.date_range(start=None, end=None, periods=None, freq=None, ...)`
  - `start` : 생성할 `DateTime`의 왼쪽 경계(문자열 또는 유사한 형식)
  - `end` : 생성할 `DateTime`의 오른쪽 경계
  - `periods` : 생성할 주기(날짜, 시간 등)의 개수
  - `freq` : `D(Day)`, `W(Week)`, `M(Month)`, `Y(Year)`, `H(Hour)`, `T` or `min(minute)`, `S(Second)`
- `pd.Timestamp(ts_input=<object>, year=None, month=None, day=None, hour=None, minute=None, second=None, ...)`
  - `ts_input` : `Timestamp`로 변환될 값(예: `'20201225T092030'`)

```
import pandas as pd
```

```
dndx1 = pd.date_range(start='1/1/2020', end='1/08/2020')
```

```
dndx2 = pd.date_range(start='20200301', periods=3)
```

```
dndx3 = pd.date_range(start='20201001', periods=3, freq='2H20min')
```

```
dndx4 = pd.date_range(start='1/1/2020', periods=4, freq='2M')
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',  
               '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08'],  
              dtype='datetime64[ns]', freq='D')
```

```
DatetimeIndex(['2020-03-01', '2020-03-02', '2020-03-03'],  
              dtype='datetime64[ns]', freq='D')
```

```
DatetimeIndex(['2020-10-01 00:00:00', '2020-10-01 02:20:00',  
               '2020-10-01 04:40:00'],  
              dtype='datetime64[ns]', freq='140T')
```

```
DatetimeIndex(['2020-01-31', '2020-03-31', '2020-05-31', '2020-07-31'],  
              dtype='datetime64[ns]', freq='2M')
```



```
import pandas as pd
```

```
ts1 = pd.Timestamp(2020, 1, 1, 10)
```

```
ts2 = pd.Timestamp('20200101T0920')
```

```
ts3 = pd.Timestamp(year=2020, month=1, day=1, hour=12)
```

```
ts4 = pd.Timestamp(year=2020, month=1, day=1)
```

```
2020-01-01 10:00:00
```

```
2020-01-01 09:20:00
```

```
2020-01-01 12:00:00
```

```
2020-01-01 00:00:00
```

## ■ DataFrame 만들기(4) - by list

```
import numpy as np
import pandas as pd

# DatetimeIndex 생성하기
dates = pd.date_range('20180521', periods=6)
print(dates, '\n')
# 생성한 index로 DataFrame 생성하기
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
print(df)
```

```
DatetimeIndex(['2018-05-21', '2018-05-22', '2018-05-23', '2018-05-24', '2018-05-25', '2018-05-26'],
              dtype='datetime64[ns]', freq='D')
```

	A	B	C	D
2018-05-21	1.103142	-0.621130	0.450134	-1.964967
2018-05-22	0.711677	0.492436	-1.089973	-0.901384
2018-05-23	-0.166606	0.311157	-0.732189	-0.120666
2018-05-24	2.086948	0.290059	-0.584881	0.127703
2018-05-25	0.588907	2.080505	1.780577	-1.571741
2018-05-26	0.884827	0.070771	-1.417167	-0.260650

## ■ DataFrame 만들기(5)

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

df = pd.DataFrame({'A':1.0,
                   'B':pd.Timestamp('20180521'),
                   'C':pd.Series(1, index=list(range(4)), dtype='float32'),
                   'D':np.array([3] * 4, dtype='int32'),
                   'E':pd.Categorical(['test', 'train', 'test', 'train']),
                   'F':'foo'})

print(df)
```

← Categorical 자료형

	A	B	C	D	E	F
0	1.0	2018-05-21	1.0	3	test	foo
1	1.0	2018-05-21	1.0	3	train	foo
2	1.0	2018-05-21	1.0	3	test	foo
3	1.0	2018-05-21	1.0	3	train	foo

## ■ DataFrame의 속성

- index : 행 인덱스
- columns : 열 인덱스
- values : 값

```
import numpy as np
import pandas as pd
```

```
# DataFrame 정의하기(dictionary 이용)
data = {'Name': ['Lee', 'Lee', 'Choi', 'Kim', 'Park'],
        'Year': [2013, 2014, 2015, 2016, 2015],
        'Points': [1.5, 1.8, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data)
print(df.index)
print(df.columns)
print(df.values)
```

```
RangeIndex(start=0, stop=5, step=1)
Index(['Name', 'Year', 'Points'], dtype='object')
[['Lee' 2013 1.5]
 ['Lee' 2014 1.8]
 ['Choi' 2015 3.6]
 ['Kim' 2016 2.4]
 ['Park' 2015 2.9]]
```

## ■ DataFrame index의 변경

### ■ 기본적인 index를 원하는 index로 변경 가능

```
import numpy as np
import pandas as pd

# DataFrame 정의하기(dictionary 이용)
data = {'Name': ['Lee', 'Lee', 'Choi', 'Kim', 'Park'],
        'Year': [2013, 2014, 2015, 2016, 2015],
        'Points': [1.5, 1.8, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data, index=['one', 'two', 'three', 'four', 'five'])
print(df)
```

	Name	Year	Points
one	Lee	2013	1.5
two	Lee	2014	1.8
three	Choi	2015	3.6
four	Kim	2016	2.4
five	Park	2015	2.9

## ■ DataFrame column의 (순서)변경

- 기본적인 column을 원하는 column으로 변경 가능(순서변경 등)

```
import numpy as np
import pandas as pd

# DataFrame 정의하기(dictionary 이용)
data = {'Name': ['Lee', 'Lee', 'Choi', 'Kim', 'Park'],
        'Year': [2013, 2014, 2015, 2016, 2015],
        'Points': [1.5, 1.8, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data, columns=['Year', 'Name', 'Points', 'Penalty'])
print(df)
```

	Year	Name	Points	Penalty
0	2013	Lee	1.5	NaN
1	2014	Lee	1.8	NaN
2	2015	Choi	3.6	NaN
3	2016	Kim	2.4	NaN
4	2015	Park	2.9	NaN

## ■ DataFrame index와 column name의 변경

### ■ index와 column의 이름 변경 가능

```
import numpy as np
import pandas as pd

# DataFrame 정의하기(dictionary 이용)
data = {'Name': ['Lee', 'Lee', 'Choi', 'Kim', 'Park'],
        'Year': [2013, 2014, 2015, 2016, 2015],
        'Points': [1.5, 1.8, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data)
df.index.name = 'No.'
df.columns.name = 'Information'
print(df)
```



Information	Name	Year	Points
No.			
0	Lee	2013	1.5
1	Lee	2014	1.8
2	Choi	2015	3.6
3	Kim	2016	2.4
4	Park	2015	2.9

# DataFrame의 Indexing과 Slicing

## ■ 기본 인덱싱

### ■ DataFrame Column indexing

- `df['col_name']` 또는 `df.col_name`
- `df[[col1, col2]]`와 같이 여러 컬럼 indexing 가능

### ■ DataFrame Row indexing

- `df[0:3]` : 행(index) 번호의 범위를 이용하여 indexing
- `df.iloc[0:3]` : 행(index) **번호를 이용하여** indexing
- `df.loc['one':'three']` : 행(index) **이름을 이용하여** indexing
- `df.ix[0:3]` : 권장되지 않음
- `df.ix['one':'three']` : 권장되지 않음



```

import numpy as np
import pandas as pd

# DataFrame 정의하기(dictionary 이용)
data = {'Name': ['Lee', 'Lee', 'Choi', 'Kim', 'Park'],
        'Year': [2013, 2014, 2015, 2016, 2015],
        'Points': [1.5, 1.8, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data)
#print(df, '\n')

print(df['Year'], '\n')

print(df[['Name', 'Points']], '\n')
# 주의(list)
print(df.iloc[1:3], '\n')

```

	Name	Year	Points
0	Lee	2013	1.5
1	Lee	2014	1.8
2	Choi	2015	3.6
3	Kim	2016	2.4
4	Park	2015	2.9

```

0    2013
1    2014
2    2015
3    2016
4    2015
Name: Year, dtype: int64

```

	Name	Points
0	Lee	1.5
1	Lee	1.8
2	Choi	3.6
3	Kim	2.4
4	Park	2.9

	Name	Year	Points
1	Lee	2014	1.8
2	Choi	2015	3.6

```

import pandas as pd

data = {'open': [11650, 11100, 11200, 11100, 11000],
        'high': [12100, 11800, 11200, 11100, 11150],
        'low' : [11600, 11050, 10900, 10950, 10900],
        'close': [11900, 11600, 11000, 11100, 11050]}

data_day = pd.DataFrame(data)
print(data_day, '\n')

# column으로 컬럼순서 변경
data_day = pd.DataFrame(data,
                        columns=['open', 'high', 'low', 'close'])
print(data_day, '\n')

# index의 지정
date = ['16.02.29', '16.02.26', '16.02.25',
        '16.02.24', '16.02.23']
data_day = pd.DataFrame(data,
                        columns=['open', 'high', 'low', 'close'],
                        index=date)
print(data_day, '\n')

# index로 행 데이터 가져오기
day_data = data_day.loc['16.02.24']
print(day_data)

```

	close	high	low	open
0	11900	12100	11600	11650
1	11600	11800	11050	11100
2	11000	11200	10900	11200
3	11100	11100	10950	11100
4	11050	11150	10900	11000

	open	high	low	close
0	11650	12100	11600	11900
1	11100	11800	11050	11600
2	11200	11200	10900	11000
3	11100	11100	10950	11100
4	11000	11150	10900	11050

	open	high	low	close
16.02.29	11650	12100	11600	11900
16.02.26	11100	11800	11050	11600
16.02.25	11200	11200	10900	11000
16.02.24	11100	11100	10950	11100
16.02.23	11000	11150	10900	11050

open	11100
high	11100
low	10950
close	11100

Name: 16.02.24, dtype: int64

## ■ Selection by label(loc[])

```
import numpy as np
import pandas as pd
```

```
dates = pd.date_range('20180521', periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
print(df, '\n')
print(df.loc[dates[0]], '\n')
print(df.loc['20180523':'20180525', ['A', 'B']], '\n')
```

	A	B	C	D
2018-05-21	0.358264	-2.195800	-0.787492	-2.150825
2018-05-22	-0.136014	-1.250828	0.599481	0.909023
2018-05-23	-0.748285	0.150766	0.196016	-0.630450
2018-05-24	-1.377841	-1.114354	-1.164417	0.891487
2018-05-25	1.239171	-1.292015	1.674718	1.134030
2018-05-26	-0.048574	-0.247509	-0.409093	0.539743

```
A    0.358264
B    -2.195800
C    -0.787492
D    -2.150825
Name: 2018-05-21 00:00:00, dtype: float64
```

	A	B
2018-05-23	-0.748285	0.150766
2018-05-24	-1.377841	-1.114354
2018-05-25	1.239171	-1.292015

## ■ Selection by position(iloc[])

```
import numpy as np
import pandas as pd

dates = pd.date_range('20180521', periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
print(df, '\n')
print(df.iloc[3:5, 2:4], '\n')
print(df.iloc[[1,2,4],[0,2]], '\n')
```

	A	B	C	D
2018-05-21	0.655354	-1.773629	-0.211734	1.549659
2018-05-22	-0.604032	0.480937	-0.362119	-0.444556
2018-05-23	-0.570404	0.015221	0.452421	0.837963
2018-05-24	0.715199	1.054076	-0.956249	-0.522862
2018-05-25	-0.859092	1.162902	0.805129	-0.706284
2018-05-26	1.191285	1.161897	0.052587	-0.778644

	C	D
2018-05-24	-0.956249	-0.522862
2018-05-25	0.805129	-0.706284

	A	C
2018-05-22	-0.604032	-0.362119
2018-05-23	-0.570404	0.452421
2018-05-25	-0.859092	0.805129

## ❖ Boolean Indexing

```
import numpy as np
import pandas as pd

dates = pd.date_range('20180521',
                      periods=6)
df = pd.DataFrame(np.random.randn(6, 4),
                  index=dates, columns=list('ABCD'))
print(df, '\n')

# A열이 0보다 큰 것만...
print(df[df.A > 0], '\n')

# 0보다 큰 항목만...
print(df[df > 0], '\n')

# 모든 항목이 0보다 큰지 여부 판단
print(df > 0, '\n')
```

	A	B	C	D
2018-05-21	2.513938	-0.069216	-1.269969	-0.000224
2018-05-22	0.543928	0.722747	-0.752938	-0.050311
2018-05-23	-0.164831	0.221810	0.203558	0.409441
2018-05-24	0.901690	-0.979920	-0.285339	1.004979
2018-05-25	-1.140251	0.308924	-0.298360	-0.276837
2018-05-26	0.504930	0.587304	0.558582	0.107541

	A	B	C	D
2018-05-21	2.513938	-0.069216	-1.269969	-0.000224
2018-05-22	0.543928	0.722747	-0.752938	-0.050311
2018-05-24	0.901690	-0.979920	-0.285339	1.004979
2018-05-26	0.504930	0.587304	0.558582	0.107541

	A	B	C	D
2018-05-21	2.513938	NaN	NaN	NaN
2018-05-22	0.543928	0.722747	NaN	NaN
2018-05-23	NaN	0.221810	0.203558	0.409441
2018-05-24	0.901690	NaN	NaN	1.004979
2018-05-25	NaN	0.308924	NaN	NaN
2018-05-26	0.504930	0.587304	0.558582	0.107541

	A	B	C	D
2018-05-21	True	False	False	False
2018-05-22	True	True	False	False
2018-05-23	False	True	True	True
2018-05-24	True	False	False	True
2018-05-25	False	True	False	False
2018-05-26	True	True	True	True

## ■ Deep copy & Append column

```
import numpy as np
import pandas as pd
```

```
dates = pd.date_range('20180521', periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
print(df, '\n')
df2 = df.copy()
df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three'] # append
print(df2)
```

	A	B	C	D	
2018-05-21	1.467631	-0.703174	-0.019962	-0.254769	
2018-05-22	-1.126177	-0.164954	-0.647567	-0.058103	
2018-05-23	0.340183	0.255191	-0.296328	0.368004	
2018-05-24	1.143418	-0.662765	0.340919	-1.026001	
2018-05-25	-1.149688	1.366978	1.323565	-1.157979	
2018-05-26	0.905588	-0.765605	-0.723773	0.815026	

	A	B	C	D	E
2018-05-21	1.467631	-0.703174	-0.019962	-0.254769	one
2018-05-22	-1.126177	-0.164954	-0.647567	-0.058103	one
2018-05-23	0.340183	0.255191	-0.296328	0.368004	two
2018-05-24	1.143418	-0.662765	0.340919	-1.026001	three
2018-05-25	-1.149688	1.366978	1.323565	-1.157979	four
2018-05-26	0.905588	-0.765605	-0.723773	0.815026	three

## ■ Boolean Indexing

```
import numpy as np
import pandas as pd
```

```
dates = pd.date_range('20180521', periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
df['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
print(df, '\n')
print(df['E'].isin(['two', 'four']), '\n')
print(df[df['E'].isin(['two', 'four'])]) # 값이 포함된 것만...
```

	A	B	C	D	E
2018-05-21	-1.028879	-0.140622	0.484885	2.098001	one
2018-05-22	1.922399	-1.108201	1.195509	3.688045	one
2018-05-23	-0.277140	-0.450373	-0.366301	0.693126	two
2018-05-24	-0.495127	-1.103904	0.404049	0.331411	three
2018-05-25	0.725936	0.379741	-0.362155	0.385754	four
2018-05-26	-0.383170	0.600350	-0.839287	-0.406119	three

2018-05-21	False
2018-05-22	False
2018-05-23	True
2018-05-24	False
2018-05-25	True
2018-05-26	False

Freq: D, Name: E, dtype: bool

	A	B	C	D	E
2018-05-23	-0.277140	-0.450373	-0.366301	0.693126	two
2018-05-25	0.725936	0.379741	-0.362155	0.385754	four

## ■ 값의 변경(Setting)

```
import numpy as np
import pandas as pd
```

```
dates = pd.date_range('20180521', periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
print(df, '\n')
s1 = pd.Series([1,2,3,4,5,6], index=dates)
df['F'] = s1 # Series 추가
df.at[dates[0], 'A'] = 0 # 특정위치 값 변경
df.iat[1,2] = 0 # 특정위치 값 변경
df.loc[:, 'D'] = np.array([5] * len(df)) # numpy array
df[df < 0] = -df # where(조건변경)
print(df, '\n')
```

	A	B	C	D
2018-05-21	-0.105252	1.248612	0.454534	-0.363001
2018-05-22	-0.982168	-1.228725	-0.002424	0.278496
2018-05-23	-1.791725	1.394449	0.353793	0.334340
2018-05-24	-0.067901	0.567677	0.236593	1.148338
2018-05-25	1.022912	-0.952381	0.510225	0.018540
2018-05-26	0.641900	-1.210912	-0.077962	-1.447788

	A	B	C	D	F
2018-05-21	0.000000	1.248612	0.454534	5	1
2018-05-22	0.982168	1.228725	0.000000	5	2
2018-05-23	1.791725	1.394449	0.353793	5	3
2018-05-24	0.067901	0.567677	0.236593	5	4
2018-05-25	1.022912	0.952381	0.510225	5	5
2018-05-26	0.641900	1.210912	0.077962	5	6



## ■ Reindexing - 순서변경 등

```
import numpy as np
import pandas as pd

dates = pd.date_range('20180521', periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
print(df, '\n')
df = df.reindex(index=dates[:4], columns=list(df.columns) + ['E'])
df.loc[:dates[1], 'E'] = 1.0
print(df)
```

	A	B	C	D
2018-05-21	0.799838	1.469235	0.872812	-0.817230
2018-05-22	0.395092	-0.456556	0.540121	-1.625846
2018-05-23	-2.336693	0.199340	-1.400223	1.011659
2018-05-24	0.174324	-1.355532	0.019341	-1.758408
2018-05-25	0.973889	0.887610	-0.808811	1.247085
2018-05-26	0.613006	-0.316288	-0.382984	-1.136906

	A	B	C	D	E
2018-05-21	0.814663	0.111710	1.039941	0.467407	1.0
2018-05-22	0.711100	0.127389	-0.801204	-0.064625	1.0
2018-05-23	0.334612	-0.505426	-0.338143	-0.564670	NaN
2018-05-24	-0.621112	0.642088	-3.338910	-1.382158	NaN

# DataFrame 연습문제(1)

- seaborn으로 tips 데이터셋을 data라는 이름으로 로드하시오.
- data를 이용하여 DataFrame을 df라는 이름으로 생성하시오.
- df을 화면에 출력하시오.
- df에서 100번째 부터 130번째 행까지 30개만 출력하시오.
- df에서 total\_bill, tip 열만 출력하시오.
- df에서 100번째 부터 130번째 행까지 30개만 출력하되, total\_bill, tip 열만 출력하시오.
- df의 특정 열을 추가, 삭제하시오.(참고: indexing 이용)
  - 참고
- df의 특정 행을 추가, 삭제하시오.(참고: loc와 indexing 이용)
  - 참고

# 참고

## ■ DataFrame

### ■ DataFrame 열 추가/삭제

#### ■ 열 추가

- `df['new_col'] = values...`

#### ■ 열 삭제

- `df.drop('column', axis=1, inplace=True)`

- `df.drop(columns='column', inplace=True)`

### ■ DataFrame 행 추가/삭제

#### ■ 행 추가

- `df.loc['new_row'] = values...`

#### ■ 행 삭제

- `df.drop('row', inplace=True)`

- `df.drop(index='row', inplace=True)`

## DataFrame 연습문제(2)

- tips, titanic, iris 데이터셋을 csv파일로 제공한 경우에는 다음과 같이 pandas로 읽어 DataFrame을 생성할 수 있다.
  - `df = pd.read_csv('filename.csv')`
- 각 csv파일을 로드하여 DataFrame을 생성한 후, 간단히 조작해 보세요.

# 참고사이트

---

- **Pandas Tutorials**

- <http://pandas.pydata.org/pandas-docs/stable/tutorials.html>

- **Python Pandas Tutorial: A Complete Introduction for Beginners**

- <https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/>