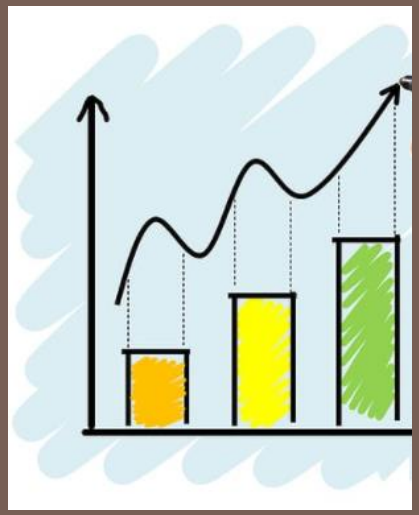
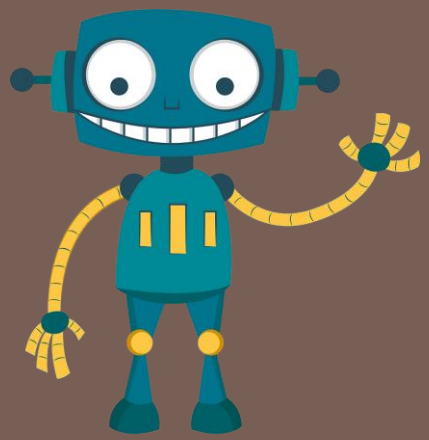
파이썬 익스프레스



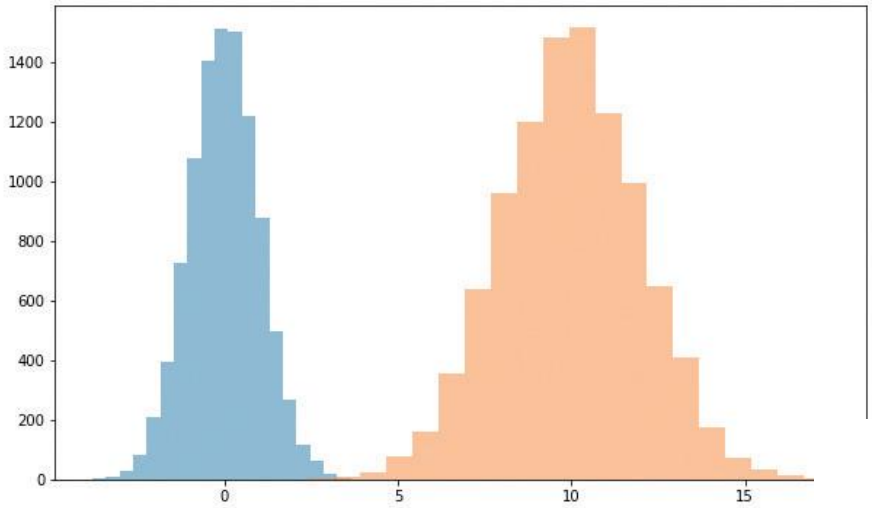


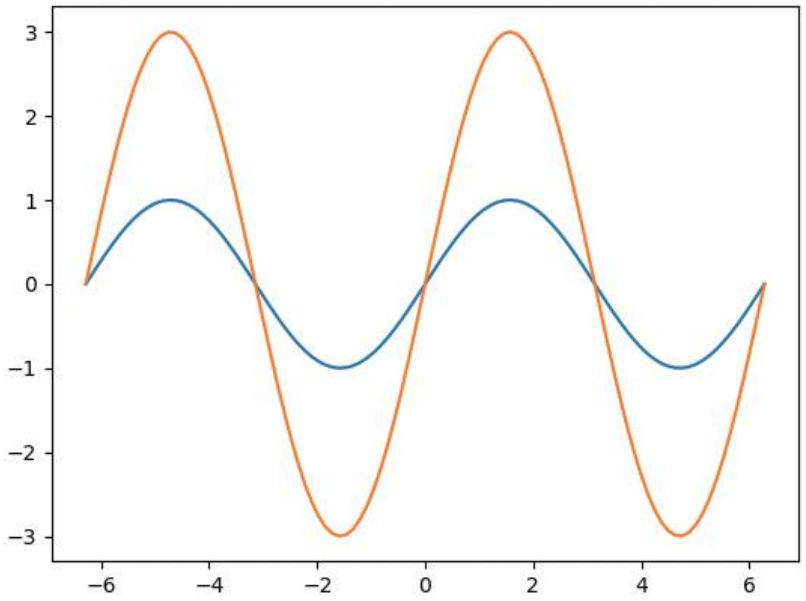
# 14장 넘파이(NUMPY)와 MATPLOT

# 학습 목표

- 넘파이가 제공하는 배열에 대하여 살펴본다.
- 넘파이가 제공하는 메소드를 살펴본다.
- 넘파이로 각종 확률 분포에서 난수를 생성해본다.
- 넘파이가 제공하는 데이터 분석 함수에 대하여 살펴본다.
- MatPlot을 이용하여 다양한 그래프를 그려본다.

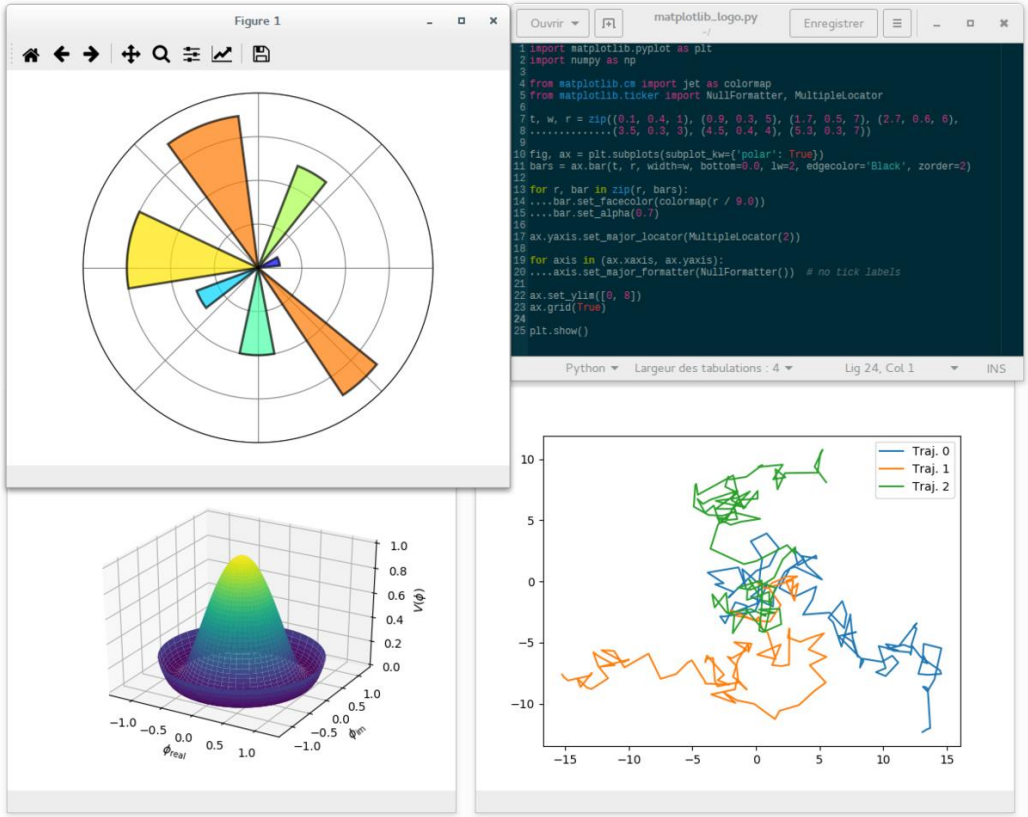
# 이번 장에서 만들 프로그램



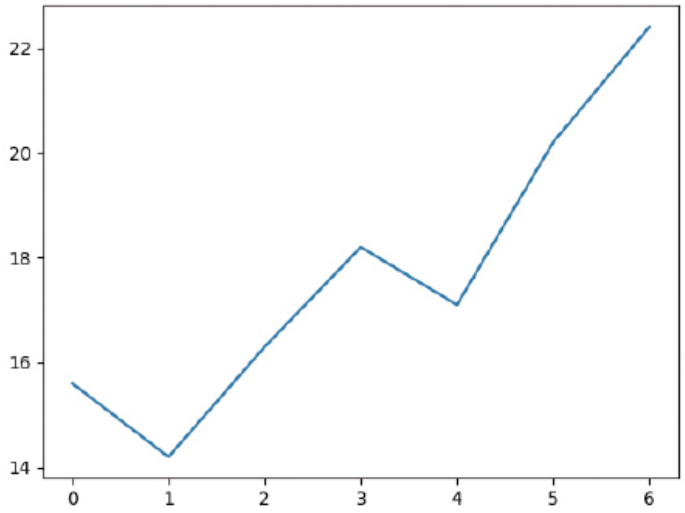
# MatPlot

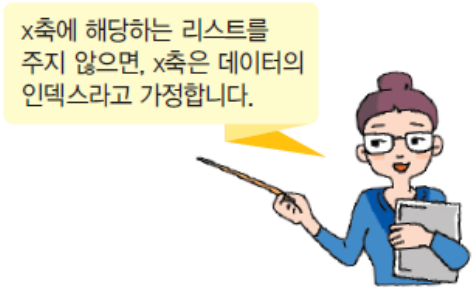
- MatPlot은 GNUplot처럼 그래프를 그리는 라이브러리이다.
- MatPlot이 MATLAB을 대신할 수 있다는 점도 장점이다. MATLAB이 비싸고 상업용 제품인 반면에 MatPlot은 무료이고 오픈 소스이다.



# 직선 그래프

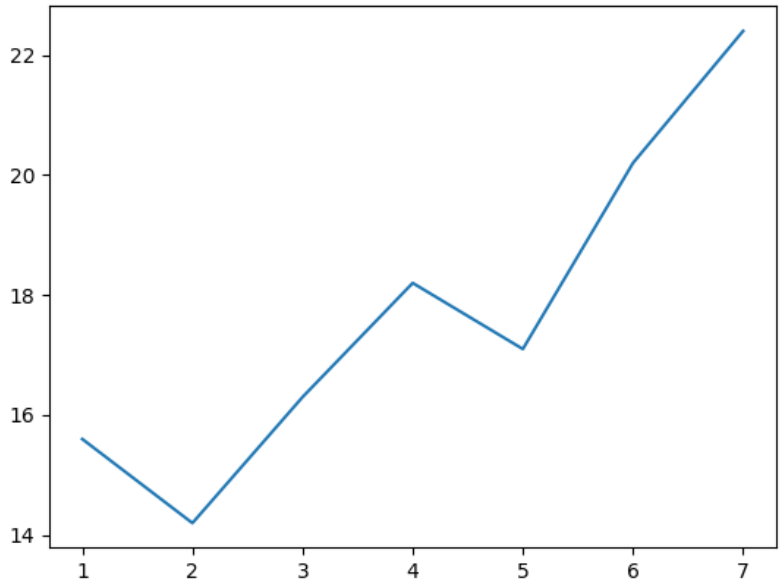
```
import matplotlib.pyplot as plt

plt.plot([15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4])
plt.show()
```



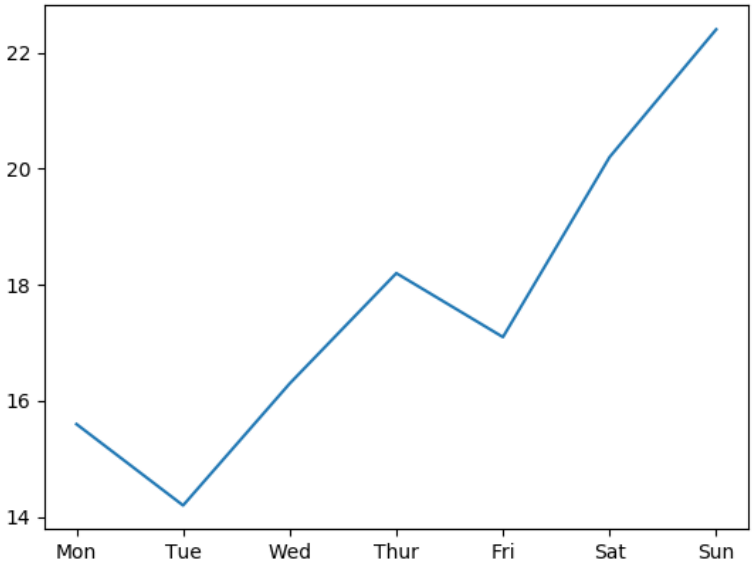
# 직선 그래프

```
X = [ 1, 2, 3, 4, 5, 6, 7]
Y = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]

plt.plot(X, Y)
plt.show()
```



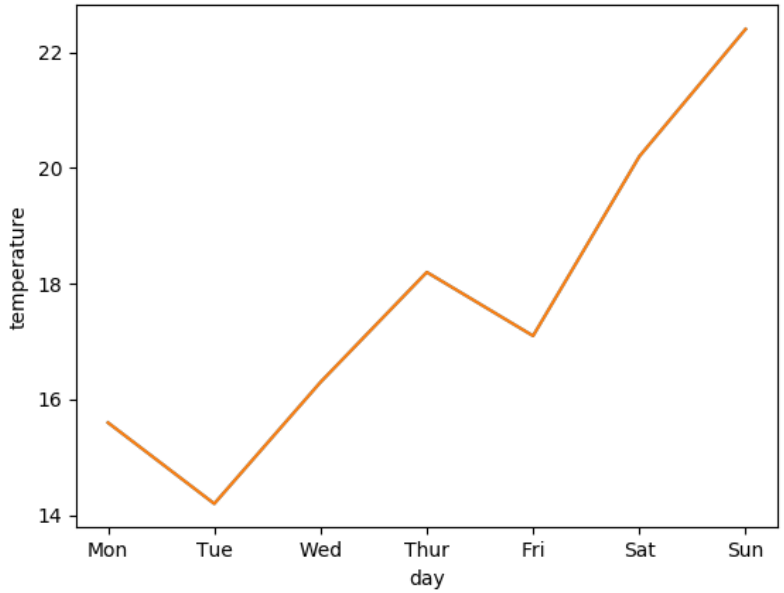
# 직선 그래프

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri",  "Sat", "Sun" ]
Y = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]

plt.plot(X, Y)
plt.show()
```



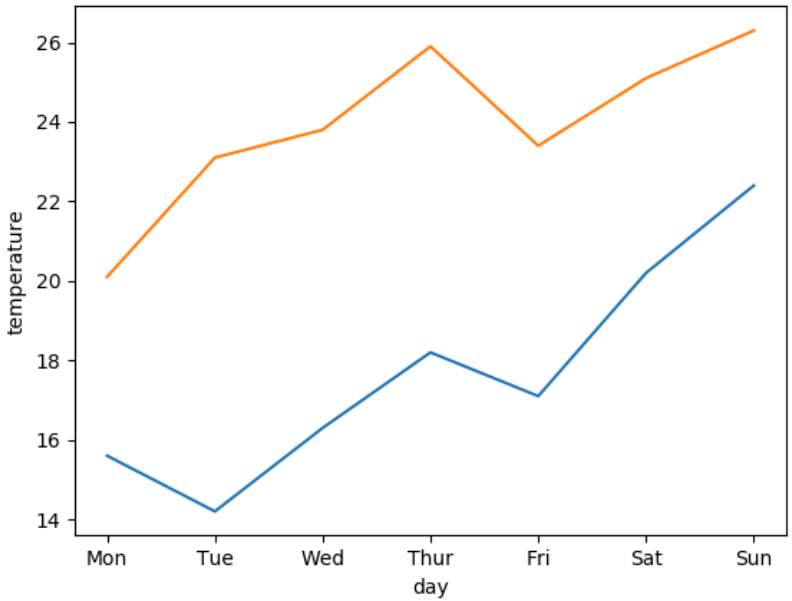
# 직선 그래프

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri",  "Sat", "Sun" ]
Y = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]

plt.plot(X, Y)
plt.xlabel("day")
plt.ylabel("temperature")
plt.show()
```



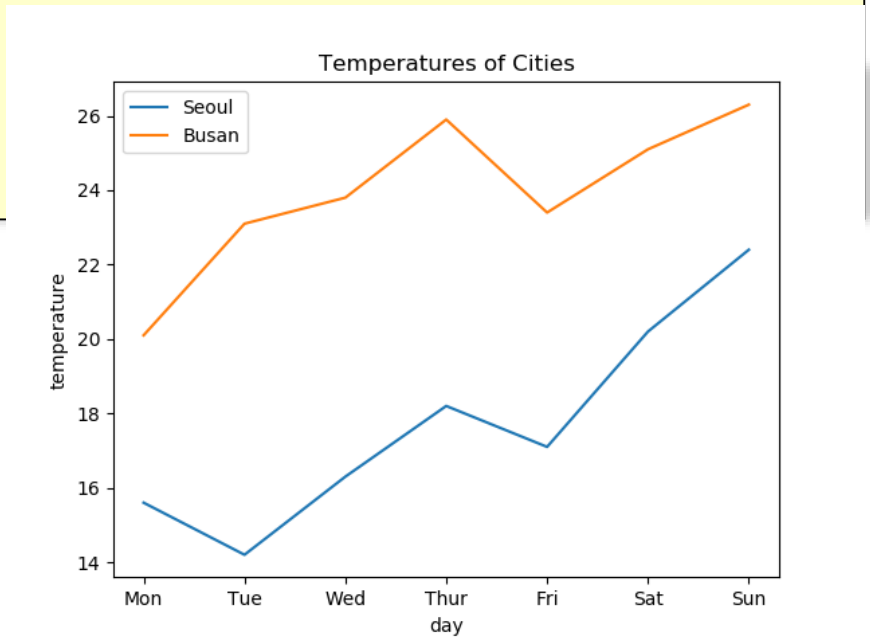
# 직선 그래프

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri",  "Sat", "Sun" ]
Y1 = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]
Y2 = [20.1, 23.1, 23.8, 25.9, 23.4, 25.1, 26.3]

plt.plot(X, Y1, X, Y2)                    # plot()에 2개의 쌍을 보낸다.
plt.xlabel("day")
plt.ylabel("temperature")
plt.show()
```



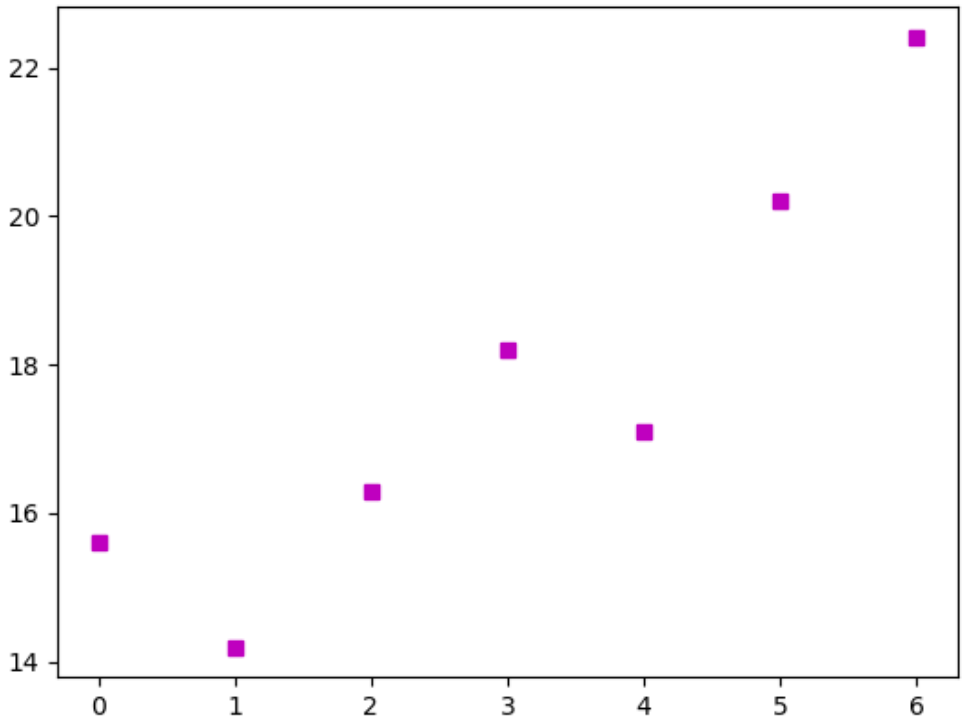
# 직선 그래프

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri",  "Sat", "Sun" ]
Y1 = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]
Y2 = [20.1, 23.1, 23.8, 25.9, 23.4, 25.1, 26.3]

plt.plot(X, Y1, label="Seoul")          # 붙이시켜서 그려도 됨
plt.plot(X, Y2, label="Busan")          # 붙이시켜서 그려도 됨
plt.xlabel("day")
plt.ylabel("temperature")
plt.legend(loc="upper left")
plt.title("Temperatures of Cities")
plt.show()
```



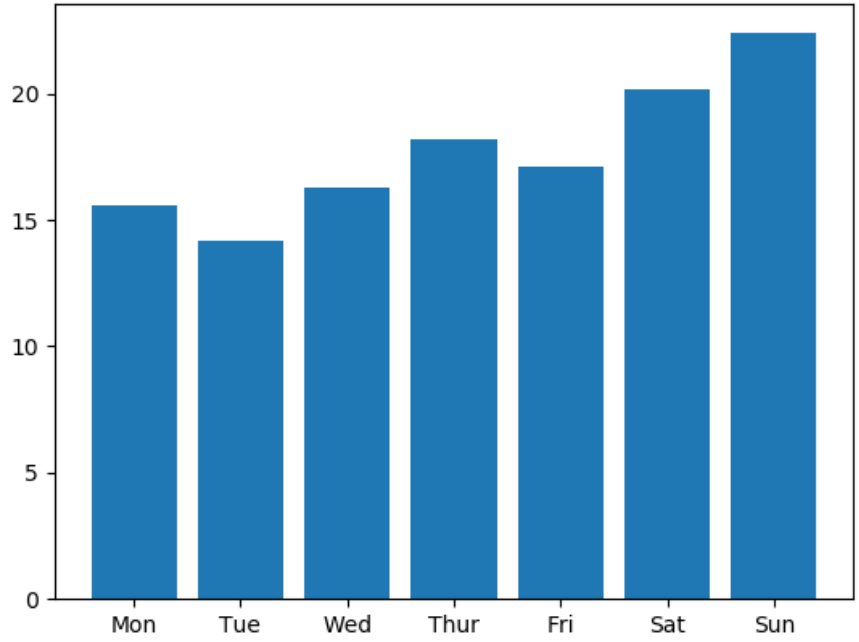
# 점선 그래프

```
plt.plot([15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4], "sm")
plt.show()
```



# 막대 그래프

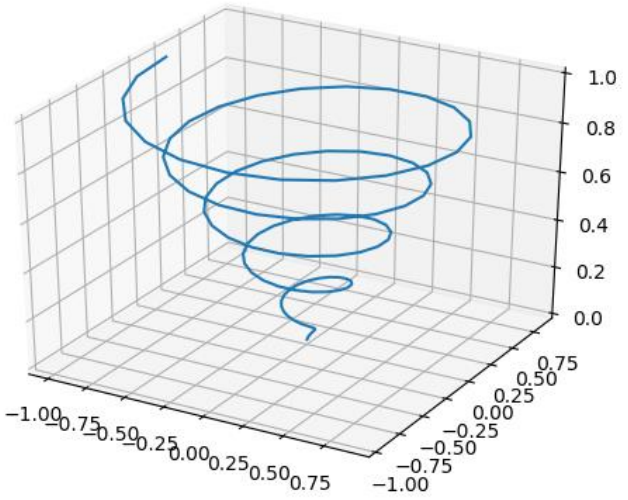
```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri",  "Sat", "Sun" ]
Y = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]
plt.bar(X, Y)
plt.show()
```



# 3차원 그래프

책의 소스에 추가

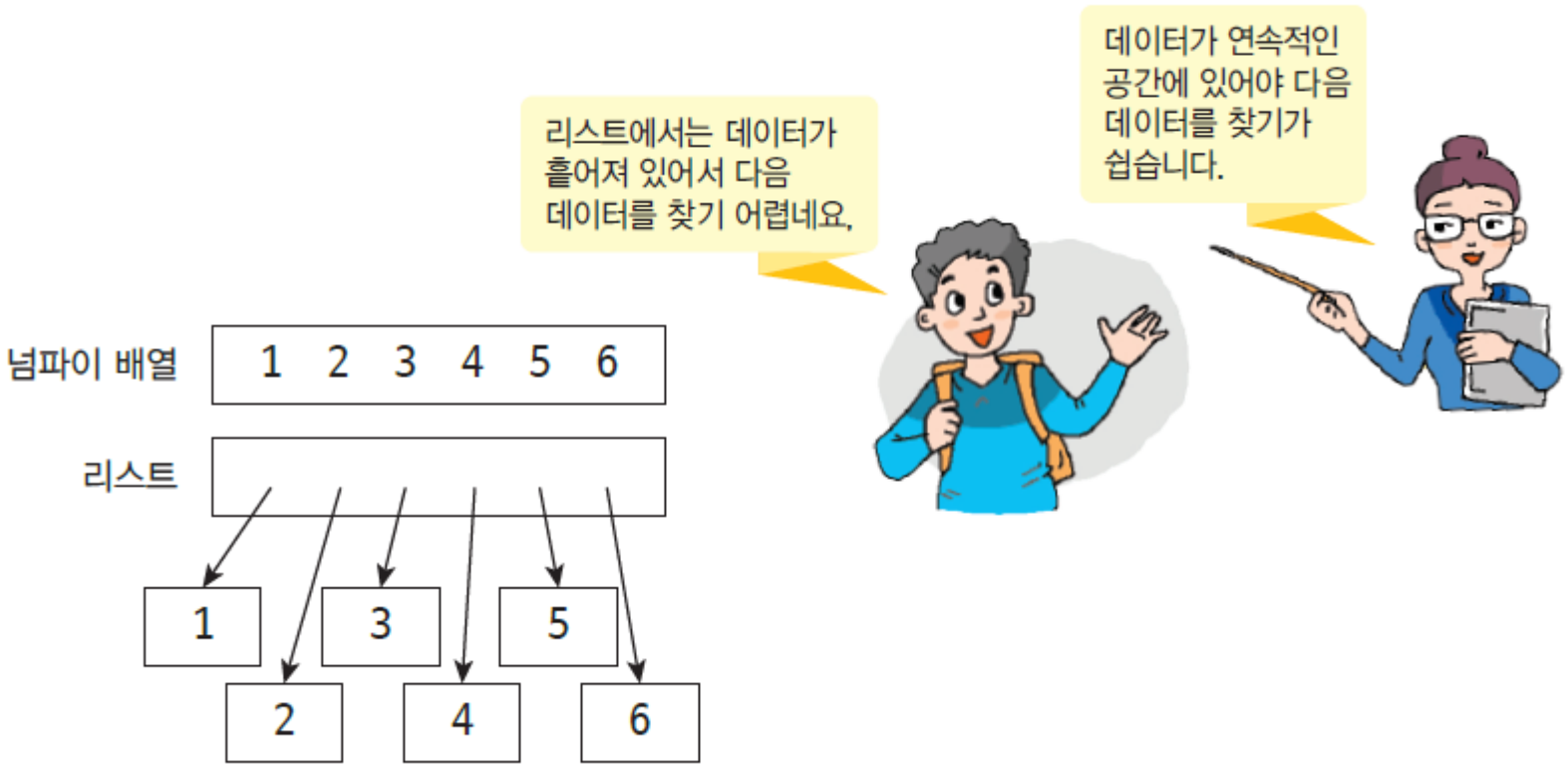
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# 3차원 축(axis)을 얻는다.
axis = plt.axes(projection='3d')
plt.show()

# 3차원 데이터를 넘파이 배열로 생성한다.
Z = np.linspace(0, 1, 100)
X = Z * np.sin(30 * Z)
Y = Z * np.cos(30 * Z)

# 3차원 그래프를 그린다.
axis.plot3D(X, Y, Z)
```



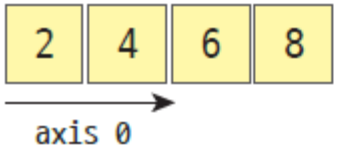
# 넘파이의 기초

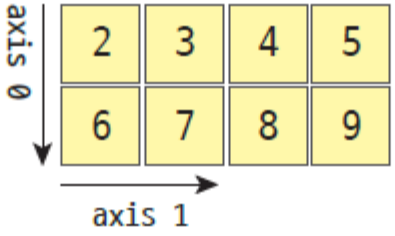
- 넘파이(NumPy)는 행렬(matrix) 계산을 위한 파이썬 라이브러리 모듈이다.
- 처리 속도가 중요한 인공지능이나 데이터 과학에서는 파이썬의 리스트 대신에 넘파이를 선호한다.
- scikit-learn이나 tensorflow 패키지도 모두 넘파이 위에서 작동

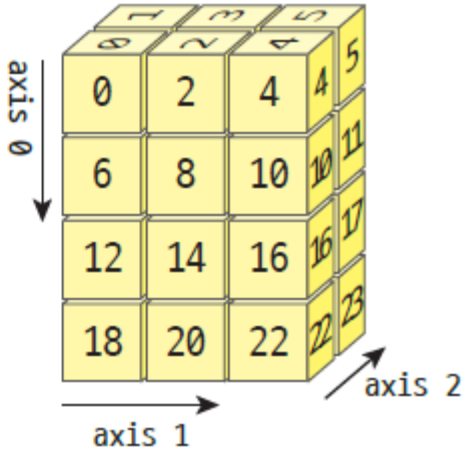
# 파이썬의 리스트(list) vs 넘파이



# 넘파이 배열의 종류

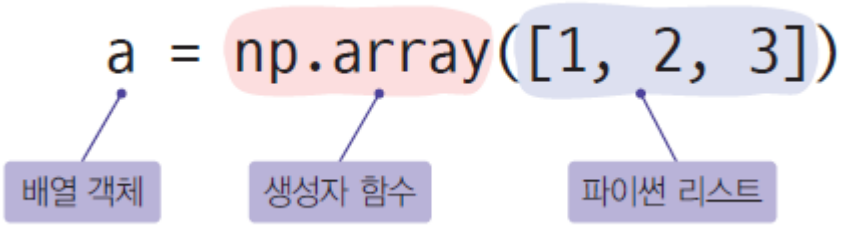

2
4
6
8
axis 0


2
3
4
5
6
7
8
9
axis 0
axis 1


0
2
4
6
8
10
12
14
16
18
20
22
axis 0
axis 1
axis 2

# 넘파이 배열

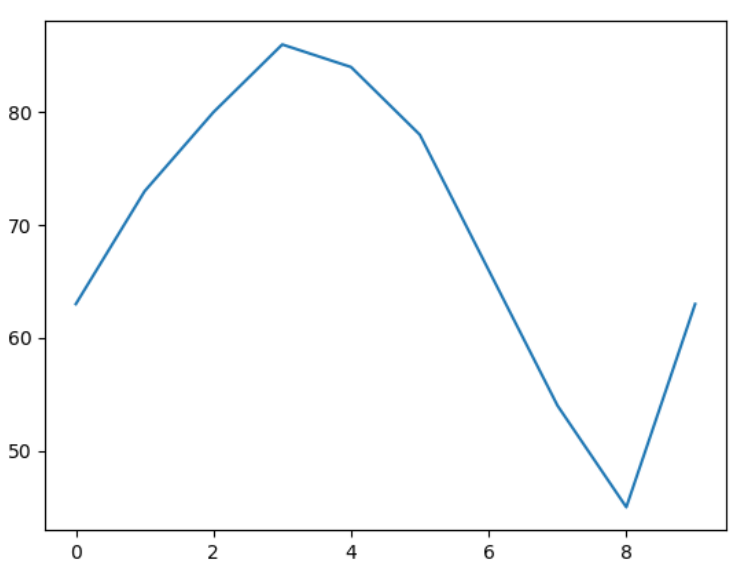
```
>>> import numpy as np

# 우리가 화씨 온도로 저장된 뉴욕의 기온 데이터를 얻었다고 하자.
>>> ftemp = [ 63, 73, 80, 86, 84, 78, 66, 54, 45, 63 ]

#이것을 넘파이로 배열로 변환하려면 다음과 같이 넘파이 모듈의 array() 함수를 호출한다.
>>> F = np.array(ftemp)
>>> F
array([63, 73, 80, 86, 84, 78, 66, 54, 45, 63])
```



# 화씨온도를 섭씨온도로 바꾸고 싶으면

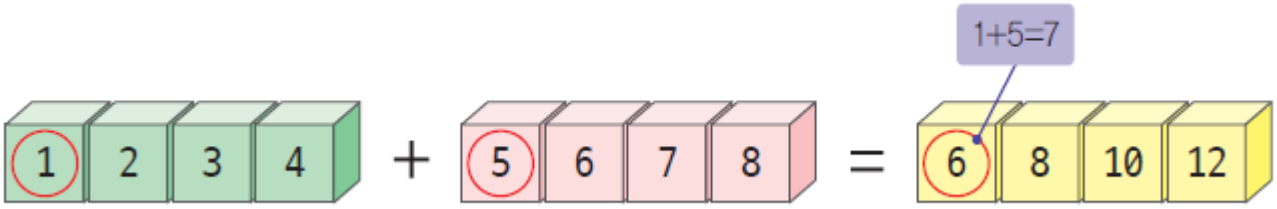
브로드캐스팅

```
>>> (F-32)*5/9                          # 배열의 모든 요소에 이 연산이 적용된다.

array([17.22222222, 22.77777778, 26.66666667, 30.        , 28.88888889,
       25.55555556, 18.88888889, 12.22222222,  7.22222222, 17.22222222])

>>> import matplotlib.pyplot as plt
>>> plt.plot(F)
>>> plt.show()
```



# 배열 간 연산

```
>>> A = np.array([1, 2, 3, 4])
>>> B = np.array([5, 6, 7, 8])
>>> result = A + B              # 넘파이 배열에 + 연산이 적용된다.
>>> result
array([ 6,  8, 10, 12])
```

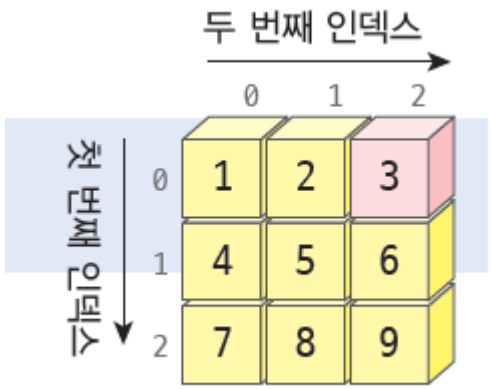


# 모든 연산자 가능

```
>>> a = np.array([0, 9, 21, 3])
>>> a < 10
array([ True, True, False, True])
```

# 2차원 배열

```
>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> b
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> b[0][2]
3
```



# Lab: BMI 계산하기

- 병원에서는 실험 대상자들의 체질량 지수(BMI: Body Mass Index)를 계산하고 싶다고 하자.

```
heights = [ 1.83, 1.76, 1.69, 1. 86, 1.77, 1.73 ]
weights = [ 86,    74,    59,   95,    80,   68  ]
```

$$BMI = \frac{Weight(kg)}{[Height(m)]^2}$$

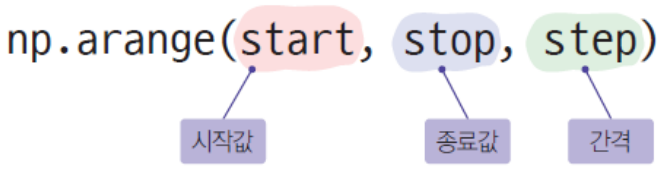
# Sol: BMI 계산하기

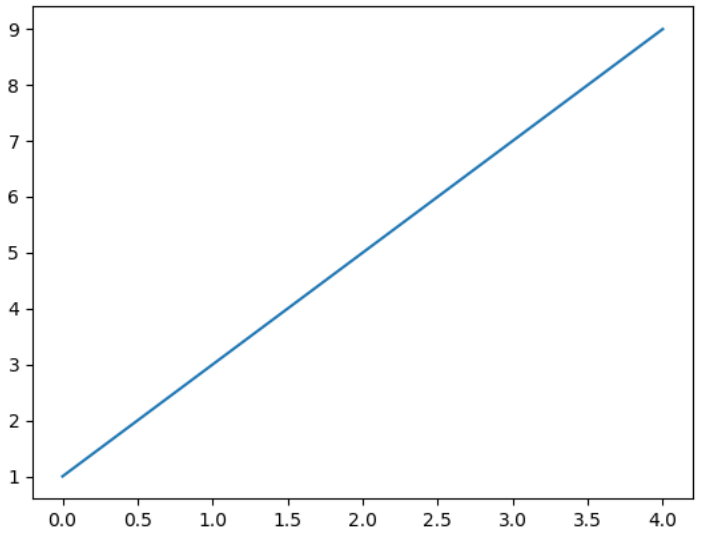
```
import numpy as np

heights = [ 1.83, 1.76, 1.69, 1.86, 1.77, 1.73 ]
weights = [ 86,    74,    59,   95,    80,   68  ]

np_heights = np.array(heights)
np_weights = np.array(weights)

bmi = np_weights/(np_heights**2)
print(bmi)
```

```
[25.68007405 23.88946281 20.65754 27.45982194 25.53544639
22.72043837]
```

# 넘파이의 데이터 생성 함수: arange()

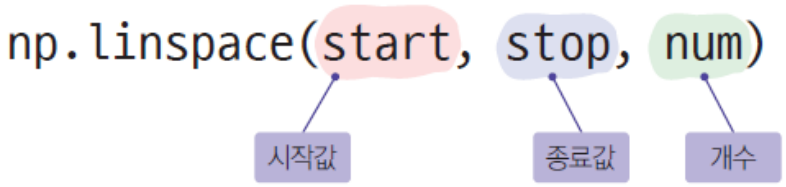
```
>>> A = np.arange(1, 10, 2)
>>> A
array([1, 3, 5, 7, 9])

>>> import matplotlib.pyplot as plt
>>> plt.plot(A)
>>> plt.show()
```

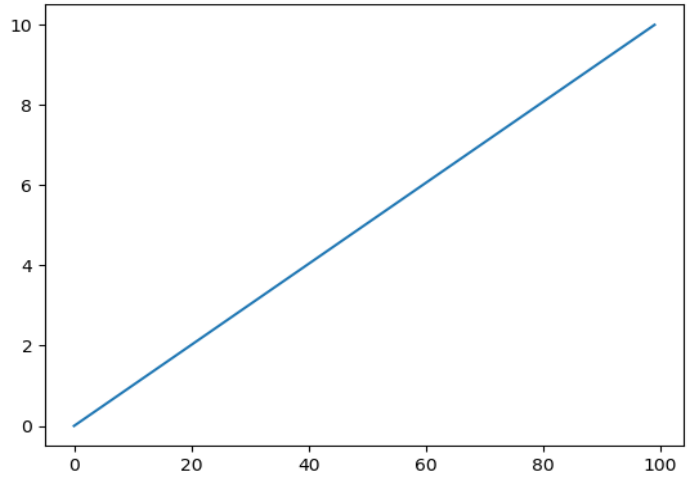


# 넘파이의 데이터 생성 함수: linspace()

```
>>> A = np.linspace(0, 10, 100)
>>> A
array([ 0.        ,  0.1010101 ,  0.2020202 ,  0.3030303 ,  0.4040404 ,
        ...
        9.09090909,  9.19191919,  9.29292929,  9.39393939,  9.49494949,
        9.5959596 ,  9.6969697 ,  9.7979798 ,  9.8989899 , 10.        ])

>>> import matplotlib.pyplot as plt
>>> plt.plot(A)
>>> plt.show()
```



# 균일 분포 난수 생성

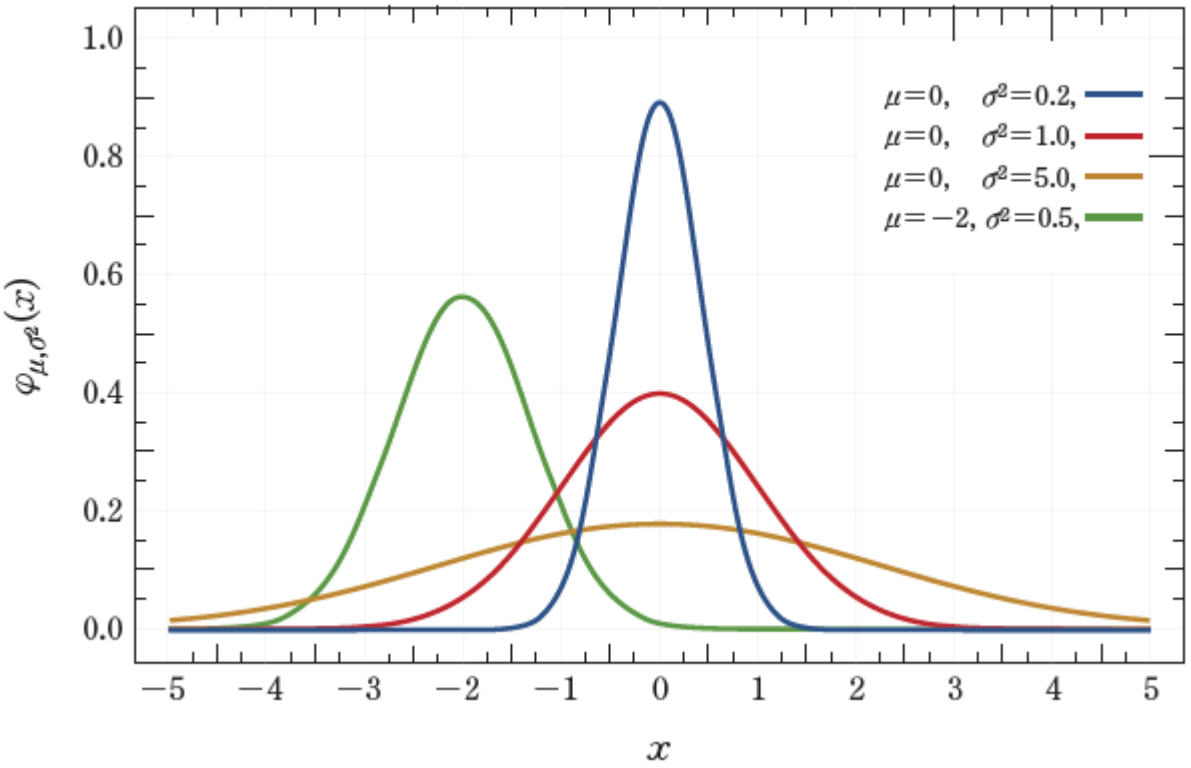
```
>>> np.random.seed(100)

# 시드가 설정되면 다음과 같은 문장을 수행하여 5개의 난수를 얻을 수 있다. 난수는 0.0에서 1.0 사이의 값으로 생성된다.
>>> np.random.rand(5)
array([0.54340494, 0.27836939, 0.42451759, 0.84477613, 0.00471886])

>>> np.random.rand(5, 3)
array([[0.12156912, 0.67074908, 0.82585276],
[0.13670659, 0.57509333, 0.89132195],
[0.20920212, 0.18532822, 0.10837689],
[0.21969749, 0.97862378, 0.81168315],
[0.17194101, 0.81622475, 0.27407375]])
```

# 정규 분포 난수 생성

```
>>> np.random.randn(5)
array([ 0.78148842, -0.65438103,  0.04117247, -0.20191691, -0.87081315])

# 난수로 채워진 5×4 크기의 2차원 배열을 생성하려면 다음과 같이 적어준다.
>>> np.random.randn(5, 4)
array([[ 0.22893207, -0.40803994, -0.10392514,  1.56717879],
       [ 0.49702472,  1.15587233,  1.83861168,  1.53572662],
       [ 0.25499773, -0.84415725, -0.98294346, -0.30609783],
       [ 0.83850061, -1.69084816,  1.15117366, -1.02933685],
       [-0.51099219, -2.36027053,  0.10359513,  1.73881773]])

# 위의 정규 분포는 평균값이 0이고 표준편차가 1.0이다. 만약 평균값과 표준편차를 다르게 하려면 다음과 같이 하면 된다
.
>>> m, sigma = 10, 2
>>> m + sigma*np.random.randn(5)
array([ 8.56778091, 10.84543531,  9.77559704,  9.09052469,  9.48651379])
```

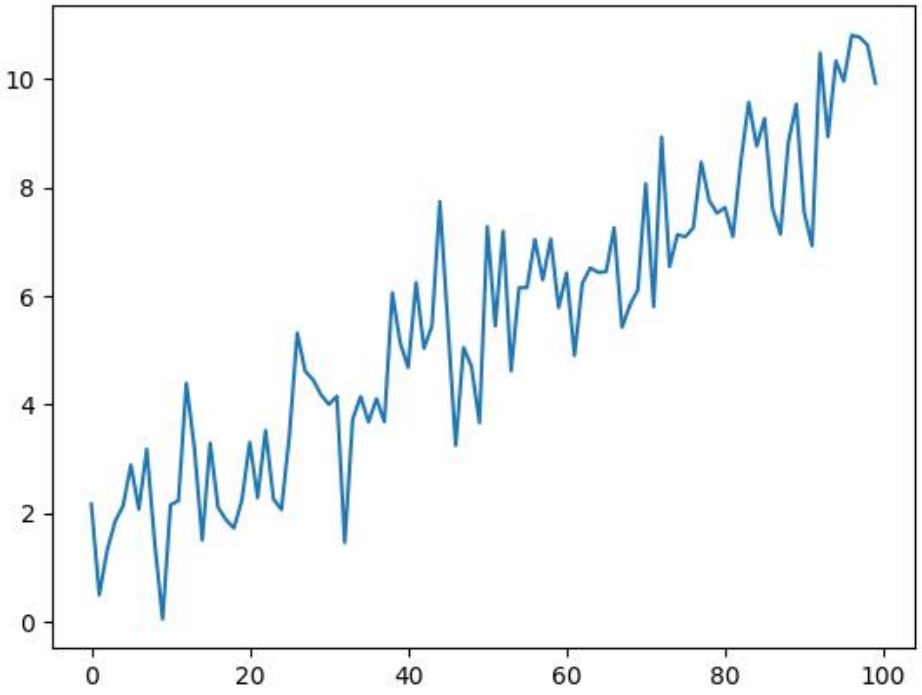
# 정규 분포 난수 생성

```
>>> mu, sigma = 0, 0.1      # 평균과 표준 편차
>>> np.random.normal(mu, sigma, 5)
array([ 0.15040638,  0.06857496, -0.01460342, -0.01868375, -0.1467971 ])
```



# Lab: 잡음이 들어간 직선 그리기

- 우리는 앞에서 linespace() 함수를 이용하여 직선을 그려보았다. 이번에는 직선 데이터에 약간의 정규 분포 잡음을 추가해보자. 즉 다음과 같이 잡음이 추가된 직선을 그려보자.



# Sol:

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

pure = np.linspace(1, 10, 100)                  # 1부터 10까지 100개의 데이터 생성
noise = np.random.normal(0, 1, 100)             # 평균이 0이고 표준편차가 1인 100개의 난수 생성

# 넘파이 배열 간 덧셈 연산, 요소별로 덧셈이 수행된다.
signal = pure + noise

# 선 그래프를 그린다.
plt.plot(signal)
plt.show()
```

# 넘파이 내장 함수

- 넘파이의 sin() 함수를 적용하면 배열의 요소에 모두 sin() 함수가 적용된다.

```
>>> A = np.array([0, 1, 2, 3])
>>> 10 * np.sin(A)
array([0. , 8.41470985, 9.09297427, 1.41120008])
```

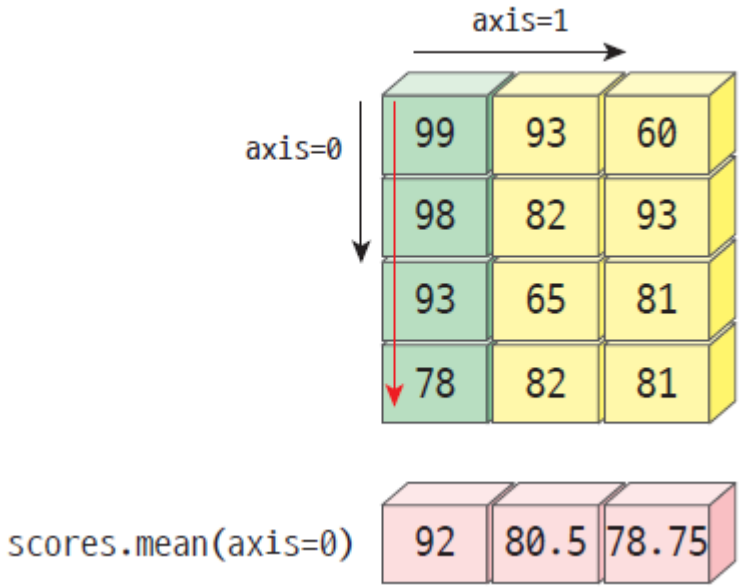
# 예제:

- 학생 4명의 3과목 성적(국어, 영어, 수학)이 넘파이 배열에 저장되었다고 가정하자.

```
>>> import numpy as np
>>> scores = np.array([[99, 93, 60], [98, 82, 93],
...:                [93, 65, 81], [78, 82, 81]])

>>> scores.sum()
1005
>>> scores.min()
60
>>> scores.max()
99
>>> scores.mean()
83.75
>>> scores.std()
11.769487386175038
>>> scores.var()
138.52083333333334
```

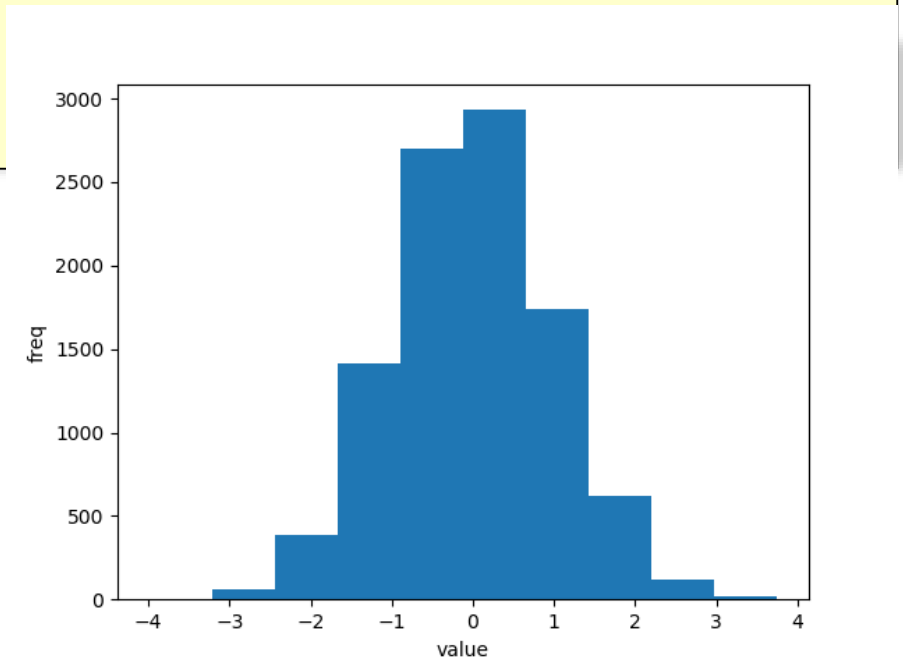
# 행이나 열 단위로 계산 가능

```
>>> scores.mean(axis=0)
array([92.  , 80.5 , 78.75])
```



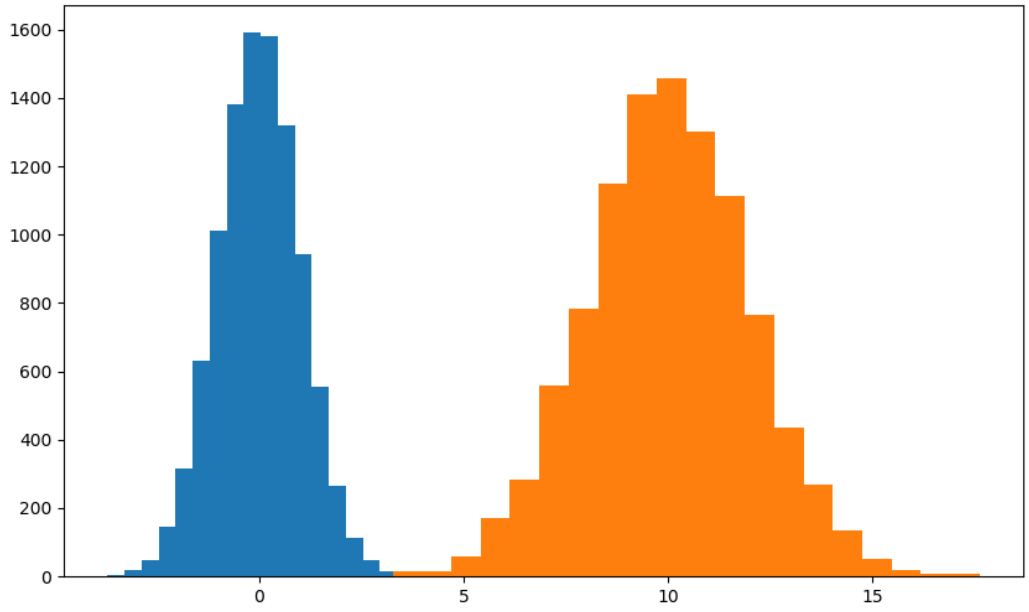
# 히스토그램

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

numbers = np.random.normal(size=10000)

plt.hist(numbers)
plt.xlabel("value")
plt.ylabel("freq")
plt.show()
```



# Lab: 정규 분포 그래프 그리기

- 다음과 같이 2개의 정규 분포를 그래프로 그려보자.

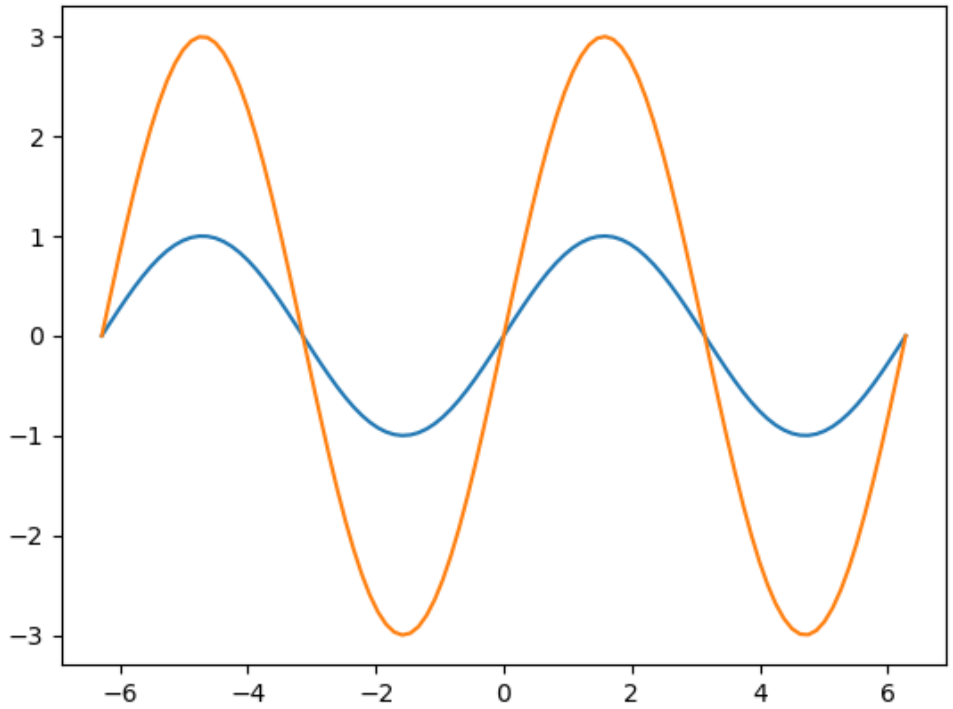


# Sol:

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

m, sigma = 10, 2
Y1 = np.random.randn(10000)
Y2 = m+sigma*np.random.randn(10000)

plt.figure(figsize=(10,6))          # 그래프의 크기 설정
plt.hist(Y1, bins=20)
plt.hist(Y2, bins=20)
plt.show()
```

# Lab: 싸인 함수 그리기

- linspace() 함수를 사용하여서 일정 간격의 데이터를 만들고 넘파이의 sin() 함수에 이 데이터를 전달하여서 싸인값을 얻는다.



# Sol:

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

# -2π에서 +2π까지 100개의 데이터를 균일하게 생성한다.
X = np.linspace(-2 * np.pi, 2 * np.pi, 100)

# 넘파이 배열에 sin() 함수를 적용한다.
Y1 = np.sin(X)
Y2 = 3 * np.sin(X)

plt.plot(X, Y1, X, Y2)
plt.show()
```

# Lab: MSE 오차 계산하기

- 회귀 문제나 분류 문제에서 실제 출력과 우리가 원하는 출력 간의 오차를 계산하기 위하여 MSE를 많이 계산한다.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (ypred_i - y_i)^2$$

# Sol:

```
import numpy as np


ypred = np.array([1, 0, 0, 0, 0])
y = np.array([0, 1, 0, 0, 0])
n = 5
MSE = (1/n) * np.sum(np.square(ypred-y))
print(MSE)
```

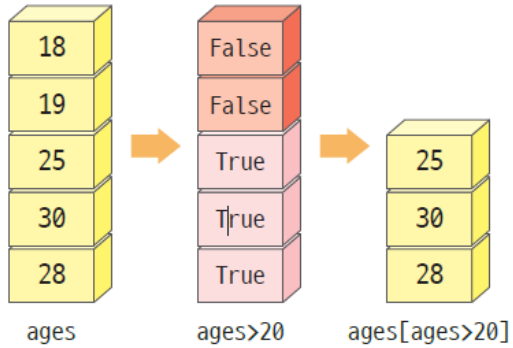
```
0.4
```

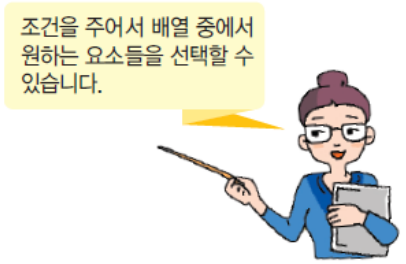
# 인덱싱과 슬라이싱

```
>>> grades = np.array([ 88, 72, 93, 94])

#  예를 들어서 0에서 2까지의 슬라이스는 다음과 같이 얻을 수 있다.
>>> grades[1:3]
array([72, 93])

# 다음과 같이 시작 인덱스나 종료 인덱스는 생략이 가능하다.
>>> y[:2]
array([88, 72])
```

# 논리적인 인덱싱

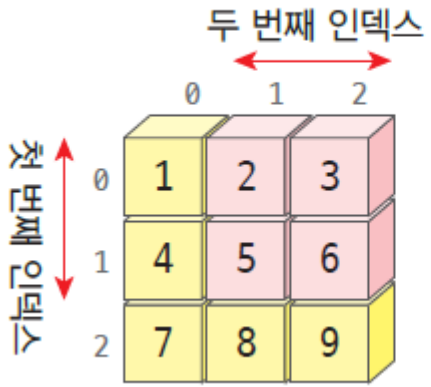
```
>>> ages = np.array([18, 19, 25, 30, 28])

# ages에서 20살 이상인 사람만 고르려고 하면 다음과 같은 조건식을 써준다.
>>> y = ages > 20
>>> y
array([False, False,  True,  True,  True])

# 논리적인 인덱싱
>>> ages[ ages > 20 ]
array([25, 30, 28])
```

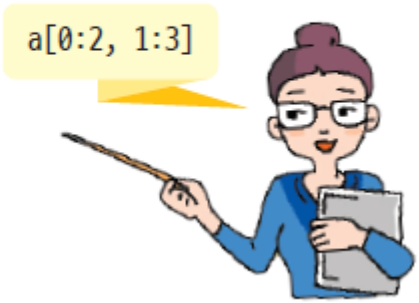


# 2차원 배열의 슬라이싱

```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> a[0:2, 1:3]


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



# 2차원 배열의 논리적인 인덱싱

```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> a > 5
array([[False, False, False],
       [False, False,  True],
       [ True,  True,  True]])

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

# Lab: 직원들의 월급 인상하기

- 현재 직원들의 월급이 [220, 250, 230]이라고 하자. 사장님이 월급을 100만원씩 올려주기로 하셨다. 넘파이를 이용하여 계산해보자.

```
>>> import numpy as np
>>> salary = np.array([220, 250, 230])


>>> salary = salary + 100
>>> salary
array([320, 350, 330])
```

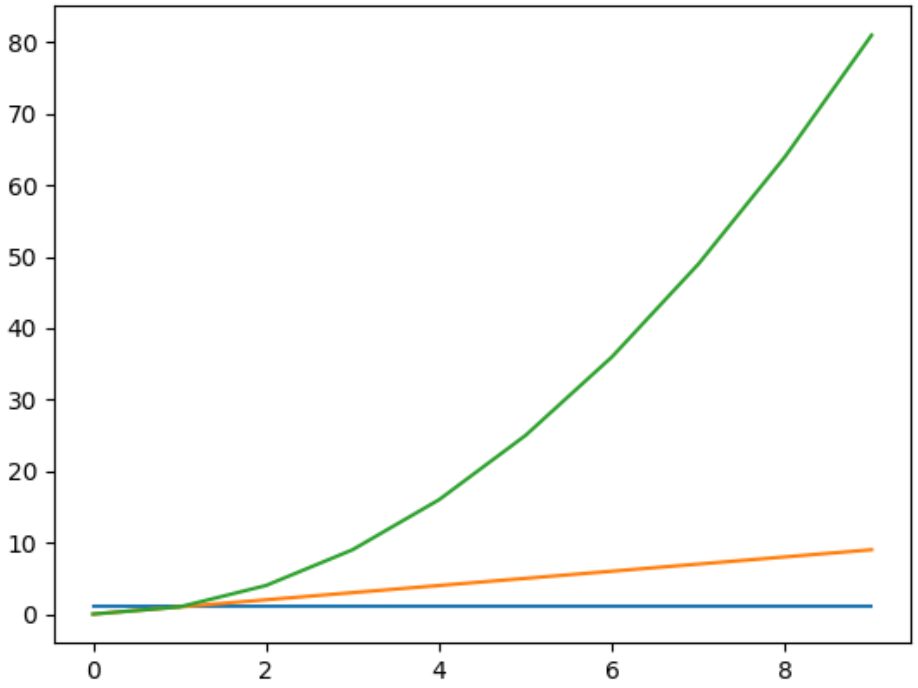
# Lab: 직원들의 월급 인상하기

- 이것을 들은 다른 사장님은 모든 직원들의 월급을 2배 올려주기로 하셨다. 어떻게 하면 될까? 월급이 450만원 이상인 직원을 찾고 싶으면 어떻게 하면 될까?

```
>>> salary = np.array([220, 250, 230])
>>> salary = salary * 2
>>> salary
array([440, 500, 460])

>>> salary > 450
array([False,  True,  True])
```

# Lab: 그래프 그리기

- linspace()로 x축값을 생성하고 f(x) = 1, f(x) = x, $f(x) = x^2$의 그래프를 함께 그려보자.



# Sol:

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

X = np.arange(0, 10)

Y1 = np.ones(10)                # ones()는 0으로 이루어진 넘파이 배열 생성
Y2 = X
Y3 = X**2

# 3개의 그래프를 하나의 축에 그린다.
plt.plot(X, Y1, X, Y2, X, Y3)
plt.show()
```

# 전치 행렬 계산하기

- 넘파이의 transpose()를 호출해도 되고, 아니면 속성 T를 참조하면 된다.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
>>> x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> x.transpose()
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
>>> x.T
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

# 역행렬 계산하기

- 넘파이 안에는 LAPACK이 내장되어 있다. np.linalg.inv(x) 와 같이 역행렬을 계산한다.

```
>>> x = np.array([[1,2],[3,4]])
>>> y = np.linalg.inv(x)
>>> y
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
>>> np.dot(x, y)                              # 행렬의 내적 계산
array([[1.00000000e+00, 1.11022302e-16],
       [0.00000000e+00, 1.00000000e+00]])
```

# 선형방정식 풀기

- 3*x0 + x1 = 9와 x0 + 2*x1 = 8가 주어졌을 때, 이들 방정식을 만족하는 해는 다음과 같이 계산한다.

```
>>> a = np.array([[3, 1], [1, 2]])
>>> b = np.array([9, 8])
>>> x = np.linalg.solve(a, b)
>>> x
array([2.,  3.])
```

# Q & A