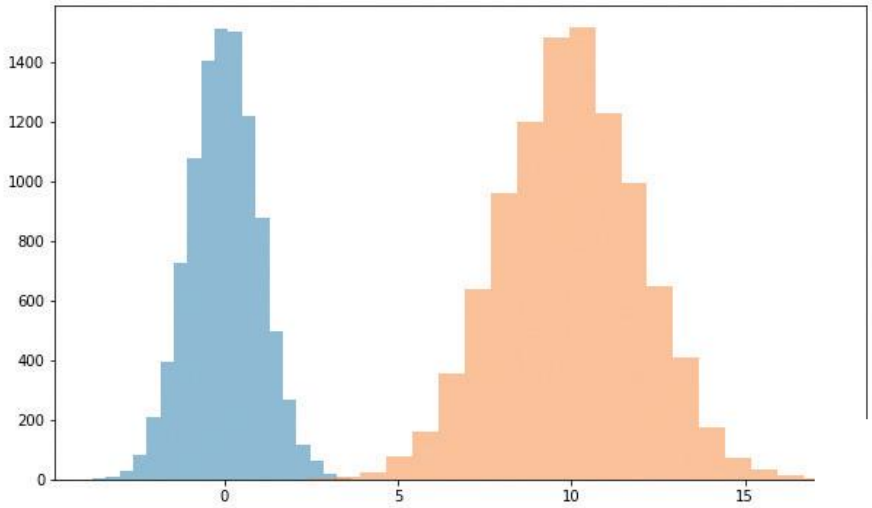
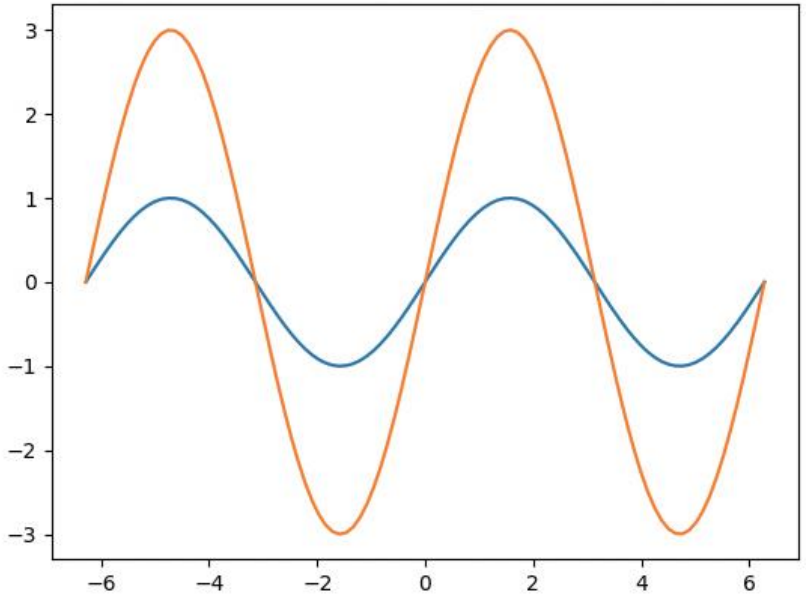
14장 넘파이(NUMPY)와 MATPLOT

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 라이브러리. 파이썬 모듈.
- MATLAB이 비싸고 상업용 제품인 반면에 MatPlot은 무료이고 오픈 소스이다.

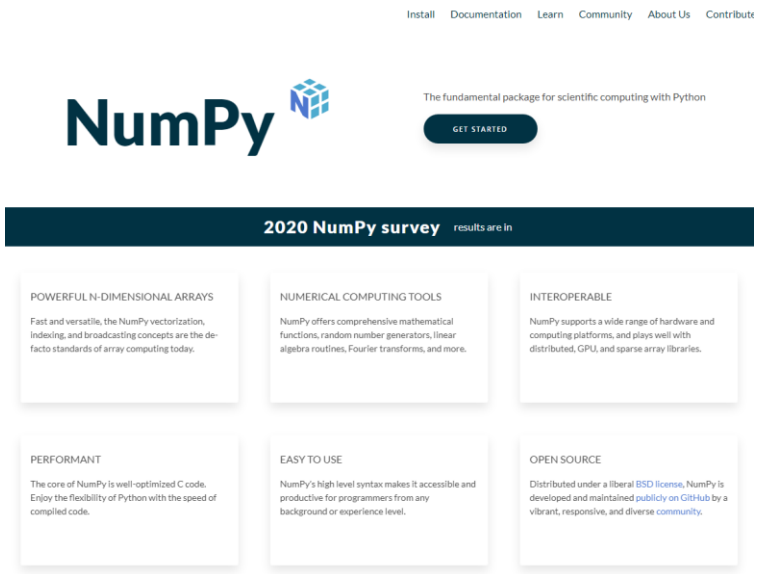
- matplotlib의 하위 모듈인 pyplot을 사용
- matplotlib.pyplot를 plt 이름으로 사용(import matplotlib.pyplot as plt) 하는 것이 거의 표준 관행

직선 그래프, 점선 그래프, 막대 그래프, 3차원 그래프

- p.609 – p.614
- 예제들을 통해 다양한 코딩 형식들을 알아두자.

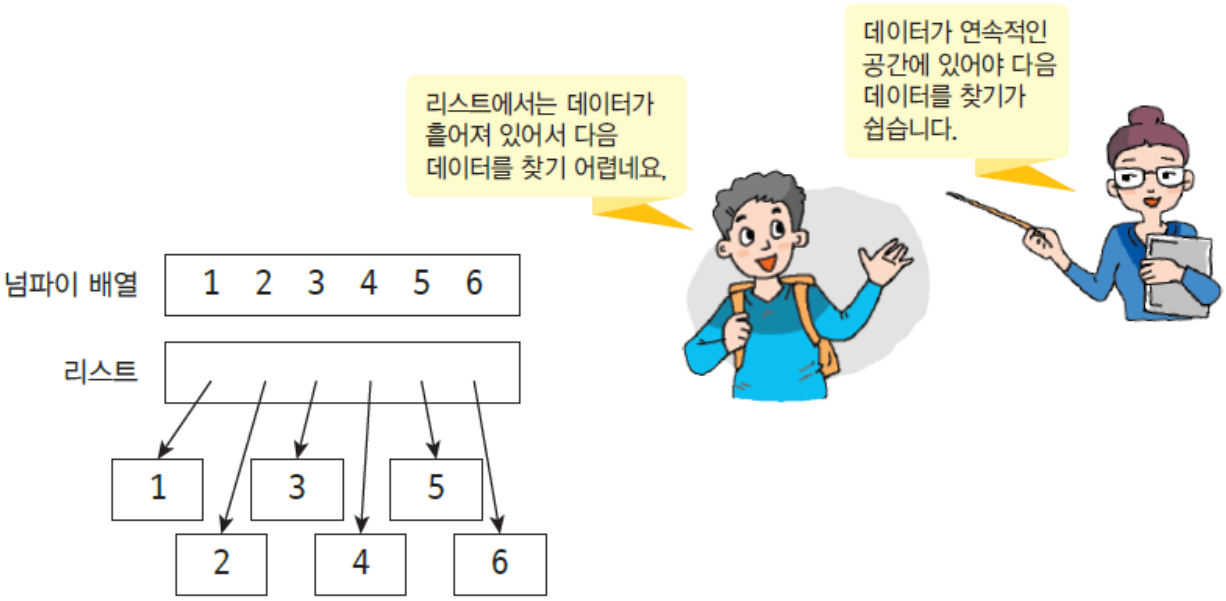
넘파이 기초

- 행렬 계산을 위한 파이썬 라이브러리 모듈.
- Numerical Python. 매우 빠른 속도
- 처리 속도가 중요한 인공지능이나 데이터 과학에서는 파이썬의 리스트 대신에 넘파이를 선호한다.
- 기계 학습 프로그램을 작성하는데 사용되는 scikit-learn이나 tensorflow 패키지도 모두 넘파이 위에서 작동



파이썬의 리스트(list) vs 넘파이

- 파이썬의 리스트에서는 데이터가 비연속적인 위치에 저장된다. 대량의 데이터를 처리할 때 상당히 불리하다. 느리다.
- 넘파이 2차원 배열은 데이터들이 연속적인 위치에 저장되어서 아주 효율적으로 데이터를 저장하고 처리할 수 있다.

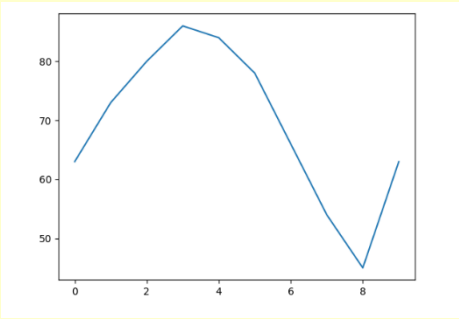


1차원 배열

- 화씨온도를 섭씨온도를 바꾸는 예

```
#(1) numpy를 사용한 코드
import numpy as np
ftemp = [ 63, 73, 80, 86, 84, 78, 66, 54, 45, 63 ]
F = np.array(ftemp)   #numpy 배열로 변환
print((F-32)*5/9)     #배열에 스칼라 값을 곱해주면 배열의 모든 요소에 스칼라 값이 곱해진다.

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

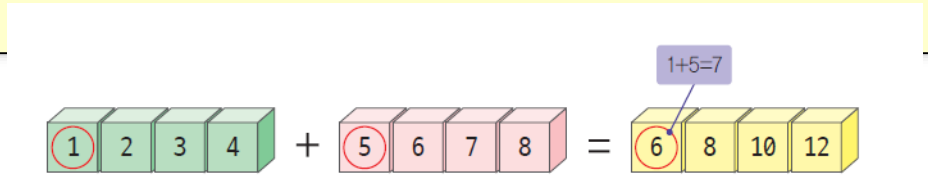
```
#(2) numpy를 사용하지 않는 코드
ftemp = [ 63, 73, 80, 86, 84, 78, 66, 54, 45, 63 ]
ctemp = [ (t-32)*5/9 for t in ftemp ]   #각 요소들을 반복문으로 계산
print(ctemp)

import matplotlib.pyplot as plt
plt.plot(ctemp)
plt.show()
```



배열 간의 연산

- 넘파이 배열에서는 + , -, *, / 등 모든 산술 연산자를 적용할 수 있으며 배열의 요소별로 적용된다.

```
>>> 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])
```

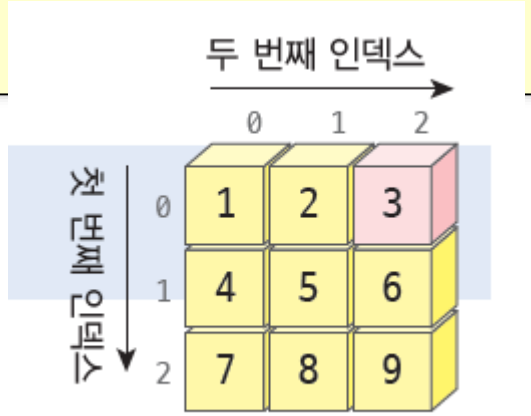



```
>>> a = np.array([0, 9, 21, 3])
>>> a < 10
array([ True, True, False, True])
```

2차원 배열

- np.array() 메소드에 이차원 리스트를 전달하여 생성

```
>>> 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
```



넘파이 배열이 훨씬 빠르다

- 동일 작업을 파이썬 리스트와 넘파이 배열로 각각 수행하여 속도를 비교해보자.

```
import numpy as np
import time

py_list = [i for i in range(10000000)]
start = time.time()                                    # 시작 시간
py_list = [i+2 for i in py_list]
end = time.time()                                      # 종료 시간
print("리스트=", end-start)

np_arr = np.array([i for i in range(10000000)])
start = time.time()                                    # 시작 시간
np_arr += 2
end = time.time()                                      # 종료 시간
print("넘파이=", end-start)
```

Lab: BMI 계산하기

- 병원에서는 실험 대상자들의 체질량 지수(BMI: Body Mass Index)를 계산하자.

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

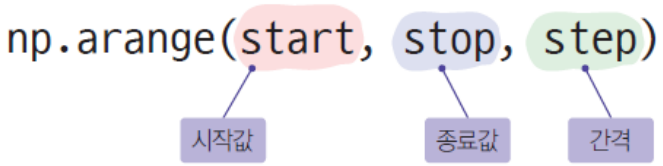
$$BMI = \frac{Weight(kg)}{[Height(m)]^2}$$

- 리스트로 하면 반복문 사용해서 여러 번 처리. 넘파이 배열을 사용하면 한번에 가능.

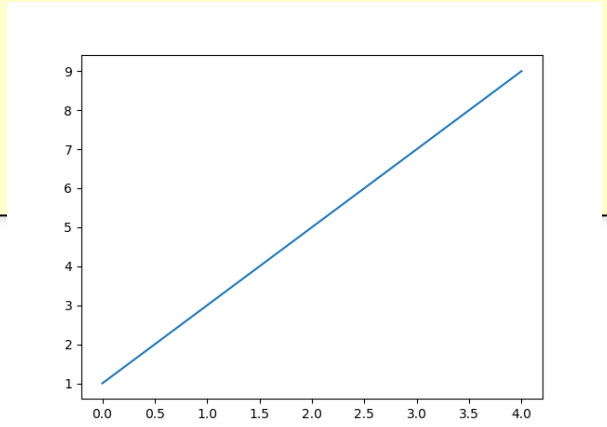
```
import numpy as np

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)
…
```

넘파이의 데이터 생성 함수: arange()

- 연속되는 정수를 가지는 넘파일 배열을 생성




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

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

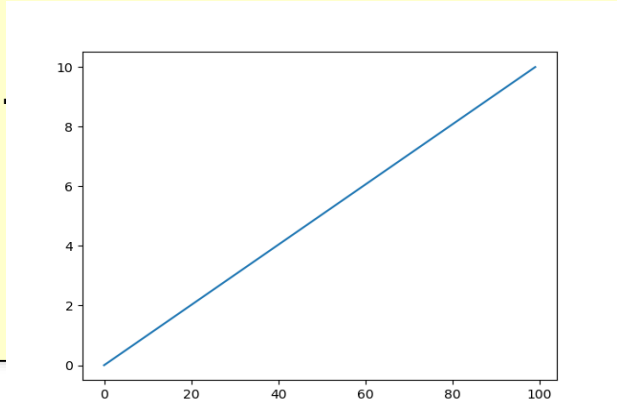


넘파이의 데이터 생성 함수: linspace()

- 시작값부터 종료값까지 균일한 간격으로 값을 생성하여 배열로 반환

```
np.linspace(start, stop, num)
```
시작값 종료값 개수

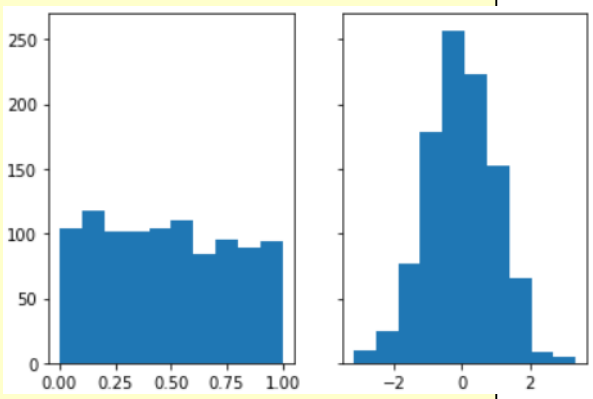
```
>>> A = np.linspace(0, 10, 100)
>>> A
array([ 0.        ,  0.1010101 ,  0.2020202 ,  0.3030303 ,  0.4040404 ,
        ...
        9.09090909,  9.19191919,  9.29292929,
        9.5959596 ,  9.6969697 ,  9.7979798 ,  9.

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



균일 분포와 정규 분포

- 균일 분포(uniform distribution) : 일정한 간격을 유지
- 정규 분포(Gaussian(normal) distribution) : 연속확률분포로 평균에서 좌우대칭하여 분포. 평균에서 멀어질수록 x축에 무한히 가까워 짐

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

uni = np.random.rand(1000)          # 균일분포에서 생성된 값들
normal = np.random.randn(1000)      #정규분포에서 생성된 값들

fig, axs = plt.subplots(1, 2, sharey=True)

# We can set the number of bins with the `bins` kwarg
axs[0].hist(uni)
axs[1].hist(normal)
```

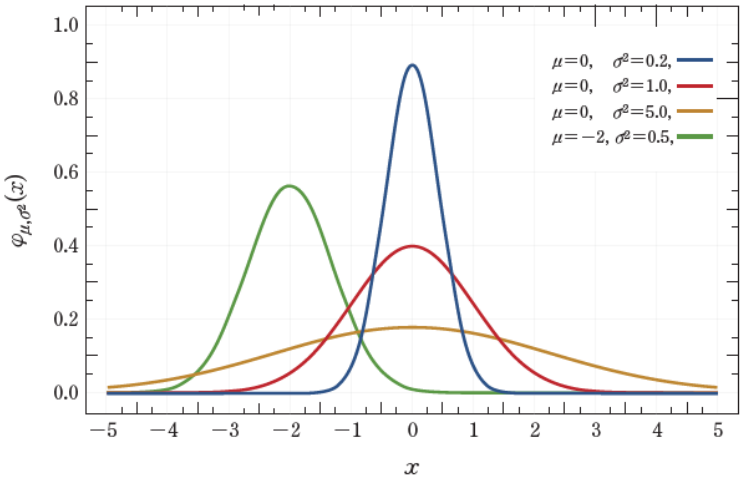


균일 분포 난수 생성

```
# seed 지정 : 난수를 생성할 때 마다 값이 달라지는 것이 아니라, 누가, 언제 하든지 간에 똑같은 난수 생성을 원한다면 seed를 지정해주면 된다.
>>> np.random.seed(100)

# 균일 분포로 5개의 난수 생성(0.0에서 1.0 사이의 값으로). rand()
>>> np.random.rand(5)
array([0.54340494, 0.27836939, 0.42451759, 0.84477613, 0.00471886])

# 2차원 배열(크기 5x3)로 난수 생성
>>> 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]])

# 어떤 범위(10에서 20사이)에 있는 난수 생성
a=10; b=20     # ;은 명령문의 끝을 의미한다.
(b-a)*np.random.rand(5)+a
```

정규 분포 난수 생성

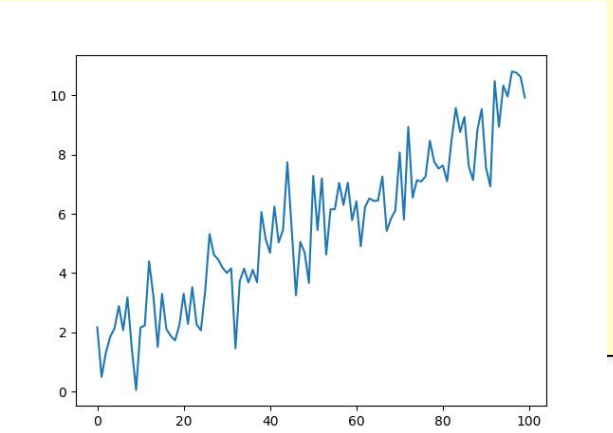
```
# 정규분포로 5개의 난수 생성. randn()
print(np.random.randn(5))

# 2차원 배열(크기 5x4)로 난수 생성
print(np.random.randn(5, 4))

# 평균값과 표준편차를 0, 1.0 외의 다른값으로 지정하는 경우
m, sigma = 10, 2      # 평균과 표준 편차
print(m + sigma*np.random.randn(5))

# 평균값과 표준편차를 지정하는 또 다른 방법. normal() 함수
mu, sigma = 0, 0.1      # 평균과 표준 편차
print(np.random.normal(mu, sigma, 5))
```



Lab: 잡음이 들어간 직선 그리기

- linespace() 함수를 이용하여 만든 직선 데이터에 정규 분포 잡음 np.random.normal() 을 추가한 차트 그리기.

```
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(pure)
plt.plot(signal)
plt.show()
```



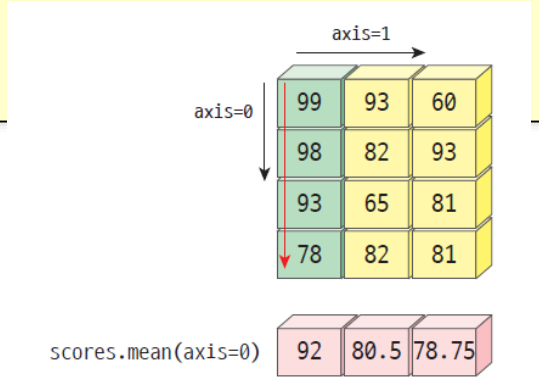
넘파이 내장 함수

- 넘파이에 함수를 적용하면 배열의 모든 요소에 함수가 적용된다.

```
scores = np.array([[99, 93, 60], [98, 82, 93], [93, 65, 81], [78, 82, 81]])
print(scores.sum())    # 합
print(scores.min())    # 최소값
print(scores.max())    # 최대값
print(scores.mean())   # 평균
print(scores.std())    # 표준편차
print(scores.var())    # 분산
```

- axis를 이용하여 특정 행(1)이나 열(0) 방향을 지정할 수 있다

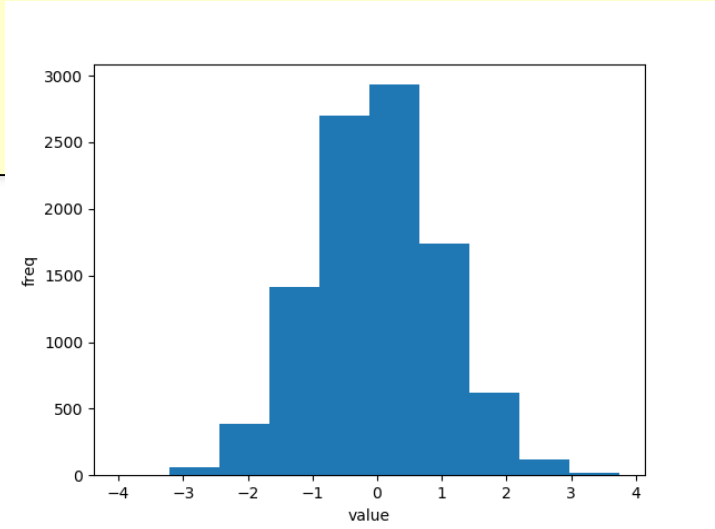
```
print(scores.mean(axis=0))
print(scores.mean(axis=1))
```



히스토그램

- 수치 데이터의 빈도를 표현

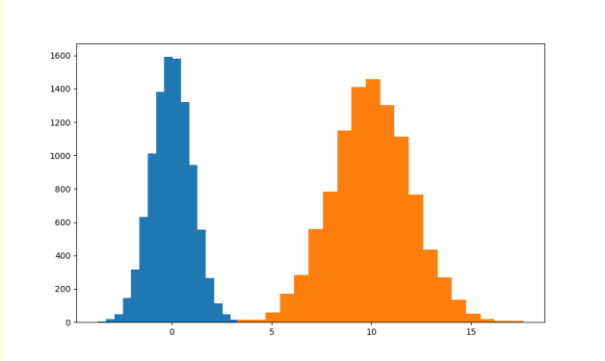
```
import matplotlib.pyplot as plt
import numpy as np

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.

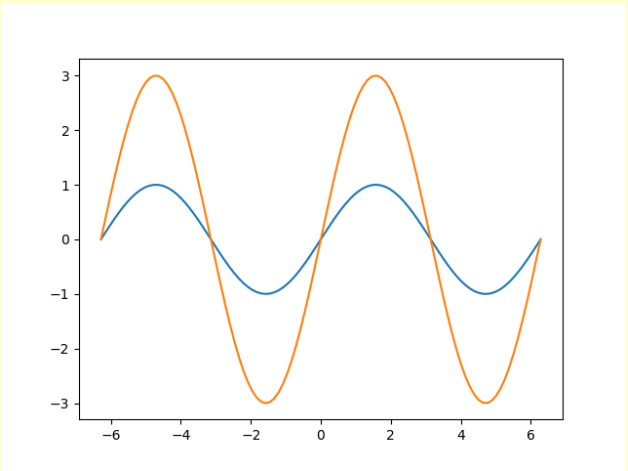
```
import numpy as np
import matplotlib.pyplot as plt


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


plt.figure(figsize=(10,6))          # 그래프의 크기 설정(단위:inch)
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.

```
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.

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

```
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)
```

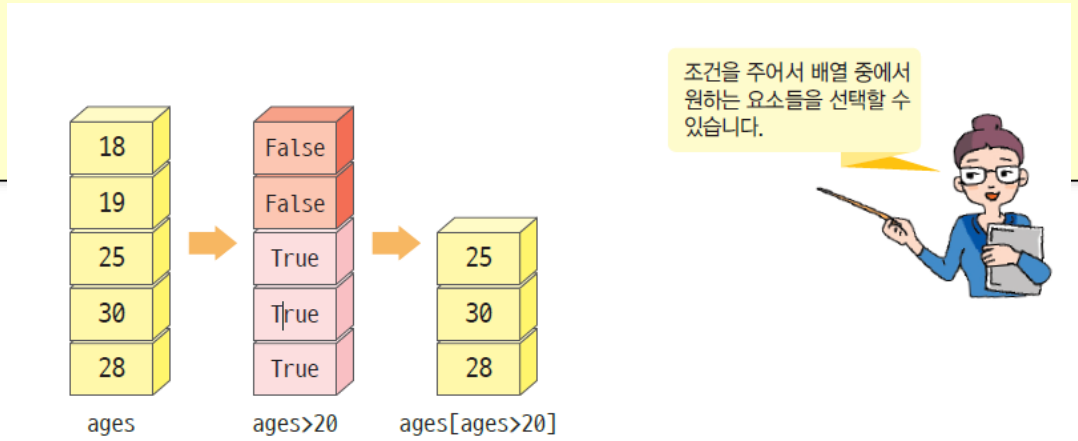
인덱싱과 슬라이싱

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

#  0에서 2까지의 슬라이스
>>> grades[1:3]
[72, 93]

# 시작 인덱스나 종료 인덱스는 생략 가능
>>> y[:2]
[88, 72]
```

논리적인 인덱싱

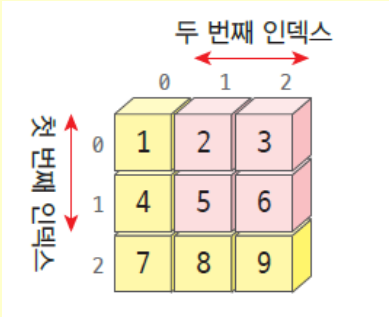
- 어떤 조건을 주어서 배열에서 원하는 값을 추려내는 것

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

>>> y = ages > 20
>>> y
array([False, False,  True,  True,  True])

>>> 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]


[[2, 3],
 [5, 6]]
```




```
>>> 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.

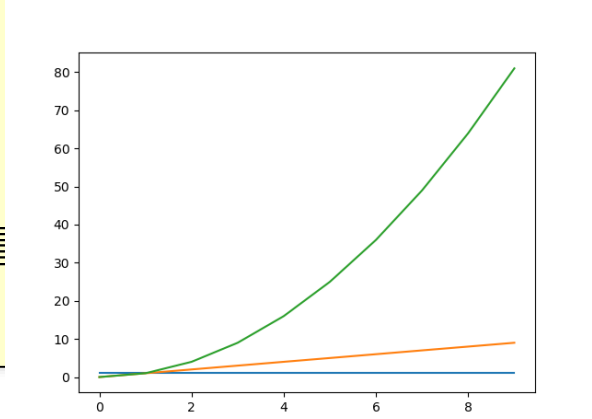
```
#월급을 100만원씩 올려주기
salary = np.array([220, 250, 230])
salary = salary + 100
print(salary)
```

```
# 월급을 2배 올려주기
salary = np.array([220, 250, 230])
salary = salary * 2
print(salary)
# 월급이 450만원 이상인 직원이 있는지 찾기
print(salary > 450)
```

Lab: 그래프 그리기

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

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

X = np.arange(0, 10)

Y1 = np.ones(10)          # ones()는 1로 이루어진 넘파이 배열 생성
Y2 = X
Y3 = X**2

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



약간의 수치해석 - 전치 행렬 계산하기

- transpose() 또는 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]])
print(x.transpose())
# 또는
print(x.T)
```

역행렬 계산하기

```
x = np.array([[1,2],[3,4]])
y = np.linalg.inv(x)          # 역행렬을 구한다.
print(y)
```

```
[[-2.    1. ]
 [ 1.5 -0.5]]
```

```
x = np.array([[1,2],[3,4]])
y = np.linalg.inv(x)
z = np.dot(x, y)              # 원래 행렬과 역 행렬을 곱하면 단위 행렬이 된다.
print(z)
```

```
[[1.00000000e+00 1.11022302e-16]
 [0.00000000e+00 1.00000000e+00]]
```

선형방정식 풀기

- 방정식을 행렬로 변환하여 해를 구한다.

$$
\begin{aligned}
a_{11}x_1 &+ a_{12}x_2 &+\cdots+ a_{1M}x_M &= b_1 \\
a_{21}x_1 &+ a_{22}x_2 &+\cdots+ a_{2M}x_M &= b_2 \\
\vdots & \quad \vdots & \quad \vdots & \quad \vdots \\
a_{N1}x_1 &+ a_{N2}x_2 &+\cdots+ a_{NM}x_M &= b_N
\end{aligned}
$$

$$
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1M} \\
a_{21} & a_{22} & \cdots & a_{2M} \\
\vdots & \vdots & \ddots & \vdots \\
a_{N1} & a_{N2} & \cdots & a_{NM}
\end{bmatrix}
\begin{bmatrix}
x_1 \\ x_2 \\ \vdots \\ x_M
\end{bmatrix}
=
\begin{bmatrix}
b_1 \\ b_2 \\ \vdots \\ b_N
\end{bmatrix}
$$

$$
A = \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1M} \\
a_{21} & a_{22} & \cdots & a_{2M} \\
\vdots & \vdots & \ddots & \vdots \\
a_{N1} & a_{N2} & \cdots & a_{NM}
\end{bmatrix},
\quad x = \begin{bmatrix}
x_1 \\ x_2 \\ \vdots \\ x_M
\end{bmatrix},
\quad b = \begin{bmatrix}
b_1 \\ b_2 \\ \vdots \\ b_N
\end{bmatrix}
$$

Ax=b

A: 계수행렬(coefficient matrix)

x : 미지수 벡터

b : 상수벡터(constant vector)

선형방정식 풀기

- 3*x1 + x2 = 9와 x1 + 2*x2 = 8가 주어졌을 때, 이들 방정식을 만족하는 해는 다음과 같이 계산한다.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NM} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$$

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

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

이번 장에서 배운 것

- 넘파이가 제공하는 배열에 대하여 공부하였다.
- 넘파이가 제공하는 메소드를 살펴보았다.
- 넘파이로 각종 확률 분포에서 난수를 생성하였다.
- 넘파이가 제공하는 데이터 분석 함수에 대하여 공부하였다.
- MatPlot을 이용하여 다양한 그래프를 그려보았다.