Matplotlib

김지성 강사

Matplotlib이란

1. Matplotlib은 파이썬에서 자료를 차트(Chart)나 플롯(plot, a graph showing the relation of two variables) 으로 시각화하는 패키지
2. Matplotlib은 다음과 같은 정형화된 차트나 플롯 등 다양한 시각화 기능 제공
    a. 라인 플롯(line plot)
    b. 스캐터 플롯(scatter plot)
    c. 컨투어 플롯(contour plot)
    d. 히스토그램(histogram)
    e. 박스 플롯(box plot)
3. 이외에도 다양한 시각화 기능을 제공

Matplotlib 및 matplotlib.pyplot 임포트

1. matplotlib 패키지에는 pyplot 이라는 서브 패키지가 존재하는데, 이 pyplot 서브패키지는 매트랩 (matlab) 이라는 수치 해석 소프트웨어의 시각화 명령을 제공
2. 간단한 시각화는 pyplot 서브패키지의 명령만으로도 충분
3. matplotlib 패키지를 임포트할 때, 별칭은 mpl / pyplot 서브 패키지의 별칭은 plt를 사용

[1]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Matplotlib이란

1. Matplotlib은 파이썬에서 자료를 차트(Chart)나 플롯(plot, a graph showing the relation of two variables) 으로 시각화하는 패키지
2. Matplotlib은 다음과 같은 정형화된 차트나 플롯 등 다양한 시각화 기능 제공
    a. 라인 플롯(line plot)
    b. 스캐터 플롯(scatter plot)
    c. 컨투어 플롯(contour plot)
    d. 히스토그램(histogram)
    e. 박스 플롯(box plot)
3. 이외에도 다양한 시각화 기능을 제공

Matplotlib 및 matplotlib.pyplot 임포트

1. matplotlib 패키지에는 pyplot 이라는 서브 패키지가 존재하는데, 이 pyplot 서브패키지는 매트랩 (matlab) 이라는 수치 해석 소프트웨어의 시각화 명령을 제공
2. 간단한 시각화는 pyplot 서브패키지의 명령만으로도 충분
3. matplotlib 패키지를 임포트할 때, 별칭은 mpl / pyplot 서브 패키지의 별칭은 plt를 사용

[1]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

기본 설정 - 그래프가 출력되지 않는다면 ?

- Jupyter 개발 환경이 아닌 non-interactive python의 경우, matplotlib을 활용하여 그래프를 그리면 결과로 별도의 Window를 띄웁니다.

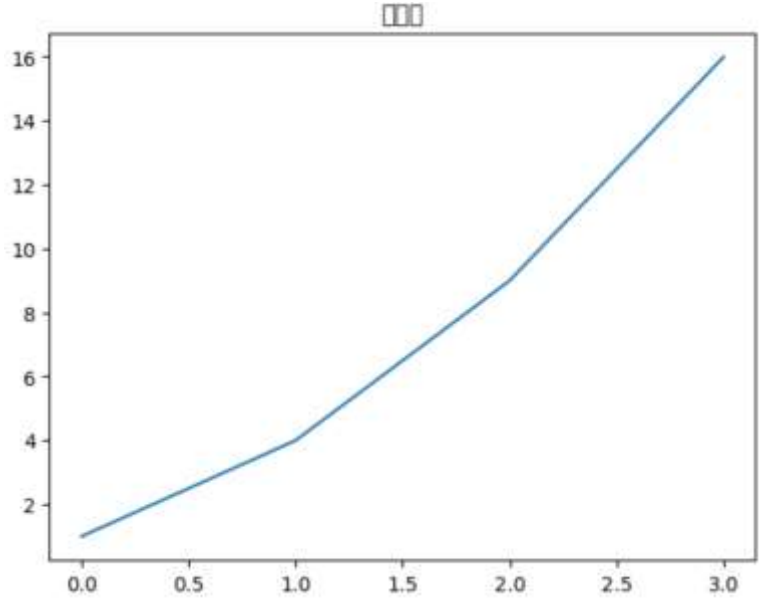
- 반대로 Jupyter(interactive python, ipython)에서는 웹페이지로 표현되기 때문에 별도의 윈도우를 띄워서 Figure를 표현할 수 없습니다. 이런 Jupyter의 특징 때문에, 경우에 따라서 시각화 결과물인 그래프가 출력되지 않는 경우가 발생됩니다. 그때는 다음처럼 **%matplotlib** 매직(magic) 명령으로 주피터 내부에 그림을 표시하도록 지정해야 합니다.

```
%matplotlib inline
```

기본 설정 - 한글 폰트 설정

- 한글 폰트를 설정하지 않고 그래프를 출력하면 아래 그림처럼 제대로 표현되지 않음

- 제목/축을 한글로 표현하고 싶다면 반드시 한글 폰트를 설정해줘야 함



기본 설정 - 한글 폰트 설정

- matplotlib.font_manager는 폰트를 관리, 사용하기 위한 모듈

- 폰트의 경로를 찾고 font_manager 패키지로 matplotlib의 글꼴을 설정

```
# 시스템 폰트 경로 리스트
font_list = mpl.font_manager.findSystemFonts()
print(font_list)

# 폰트 설정하기
import matplotlib.font_manager as fm # 폰트 매니저 임포트
font_path = '원하는 폰트 경로'
font = fm.FontProperties(fname=font_path).get_name()
mpl.rc('font', family=font)
```

기본 설정 - 마이너스(-) 설정

- matplotlib은 (-)를 표시할 때 unicode minus를 ascii hypen 보다 우선적으로 사용

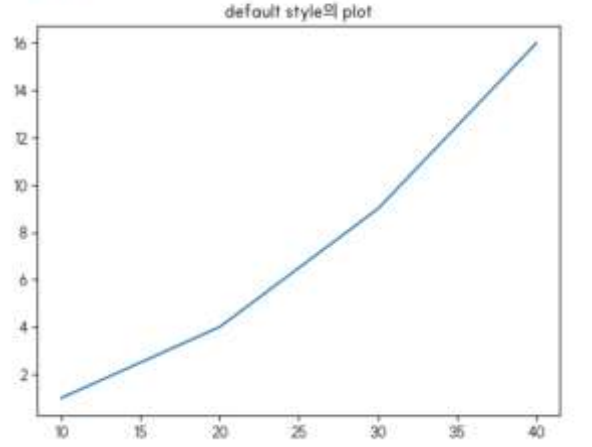
- 이때 (-) 부호가 깨지는 것을 방지하기 위해 다음과 같은 코드 적용

```
mpl.rc('axes', unicode_minus=False)
```

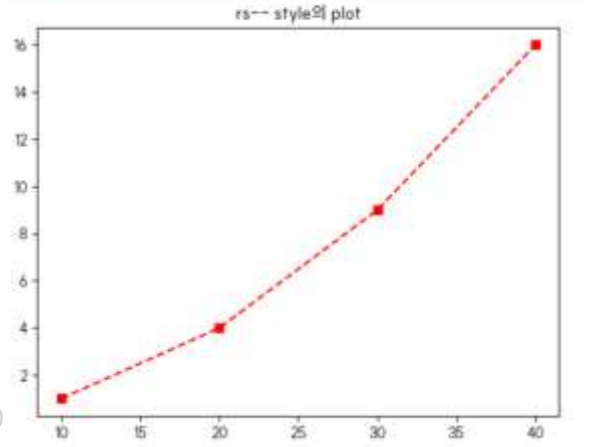
Style 설정

- 플롯 명령어는 보는 사람이 그림을 더 알아보기 쉽게 하기 위해 다양한 스타일(style)을 지원
- plot 명령어에서는 다음과 같이 추가 문자열 인수를 사용하여 스타일을 지원

```
plt.title("default style의 plot")
plt.plot([10,20,30,40],[1,4,9,16])
plt.show()
```




```
plt.title("rs-- style의 plot")
plt.plot([10,20,30,40],[1,4,9,16], 'rs--')
plt.show()
```



Style 설정 - style의 설정 순서

- 스타일 문자열은 색상(color), 마커(marker), 선 종류(line style)의 순서로 지정한다. 만약 이 중 일부가 생략되면 디폴트값이 적용

"색상 마커 선종류"

Style 설정 - 첫 번째, color

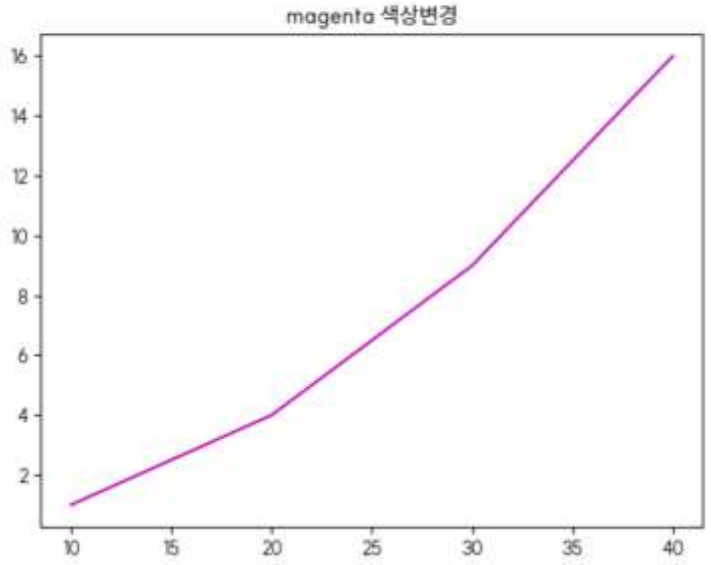
- 색깔을 지정하는 방법은 색 이름 혹은 약자를 사용하거나 # 문자로 시작되는 RGB코드를 사용

색 이름	약자
black	k
white	w
red	r
blue	b
green	g
cyan	c
magenta	m
yellow	y

Style 설정 - 첫 번째, color

- c=”m” 옵션을 설정하여 그래프 색상을 magenta 색상으로 변경

```
# 색상
plt.title("magenta 색상변경")
plt.plot([10,20,30,40],[1,4,9,16], c="m")
plt.show()
```



Style 설정 - 두 번째, marker

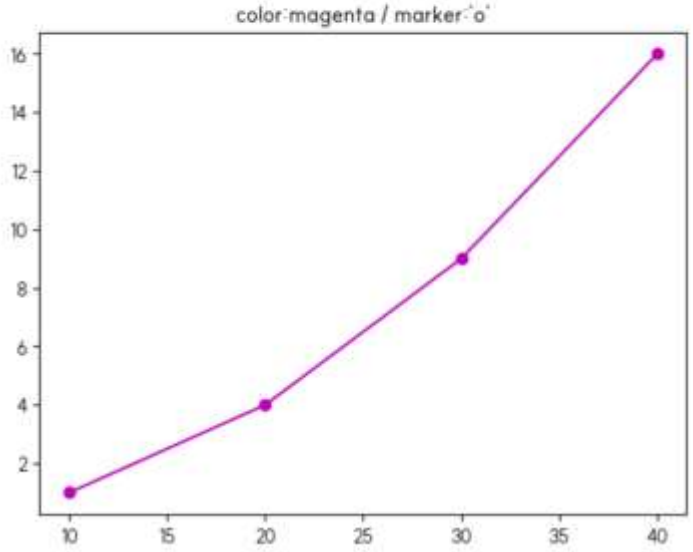
- 데이터 위치를 나타내는 기호를 마커(marker)라고 한다. 마커의 종류는 다음과 같습니다.

구분	설명	구분	설명
.	point marker	*	star marker
,	pixel marker	p	pentagon marker
o	circle marker	h	hexagon1 marker
v, 1	triangle_down marker	H	hexagon2 marker
^, 2	triange_up marker	+	plus marker
<, 3	triangle_left marker	x	x marker
>, 4	triangle_right marker	D	diamond marker
_, |	hline, vline	d	thin_diamond marker

Style 설정 - 두 번째, marker

- marker='o' 옵션을 사용하여 O 모양의 마커 표시

```
# 색상/마커
plt.title("color:magenta / marker:'o'")
plt.plot([10,20,30,40],[1,4,9,16], c="m", marker="o")
plt.show()
```



Style 설정 - 세 번째, linestyle

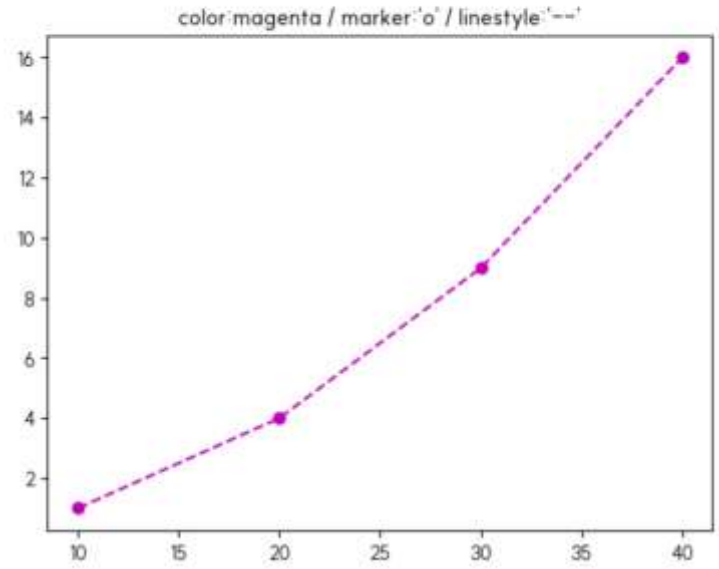
- 선 스타일의 지정 문자열은 다음과 같습니다.

구분	설명
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line
'None', ' ', ''	draw nothing

Style 설정 - 세 번째, linestyle

- ls="--" 옵션을 사용하여 점선 그래프로 변경

```
# 색상/마커/선 스타일
plt.title("color:magenta / marker:'o' / linestyle:'--'")
plt.plot([10,20,30,40],[1,4,9,16], c="m", marker="o", ls="--")
plt.show()
```



Style 설정 - 그 외

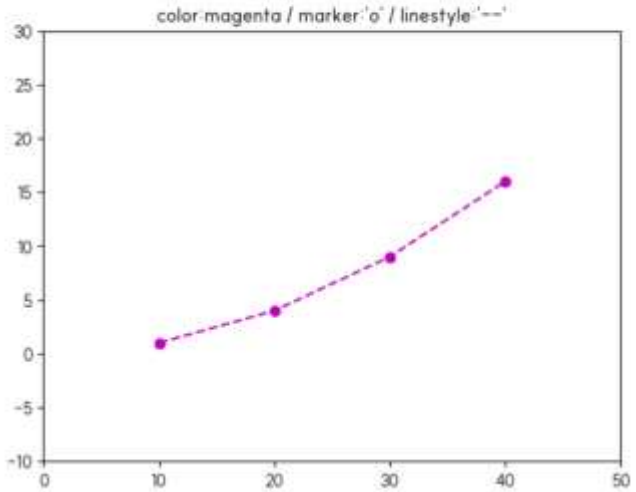
- 그 외 다양한 style 설정은 아래표를 참고

구분	약자	description
color	c	선 색상
linewidth	lw	선 굵기
linestyle	ls	선 스타일
marker		마커 종류
markersize	ms	마커 크기
markeredgecolor	mes	마커 선 색깔
markeredgewidth	mew	마커 선 굵기
markerfacecolor	mfc	마커 내부 색상

축 범위 설정

- plot의 특정 점들은 경계선에 있어서 잘 보이지 않는 경우가 존재 이럴때 x축과 y축의 범위를 지정해야 한다.
- x축의 범위를 지정하는 xlim(최소값, 최대값), y축의 범위를 지정하는 ylim(최소값, 최대값)

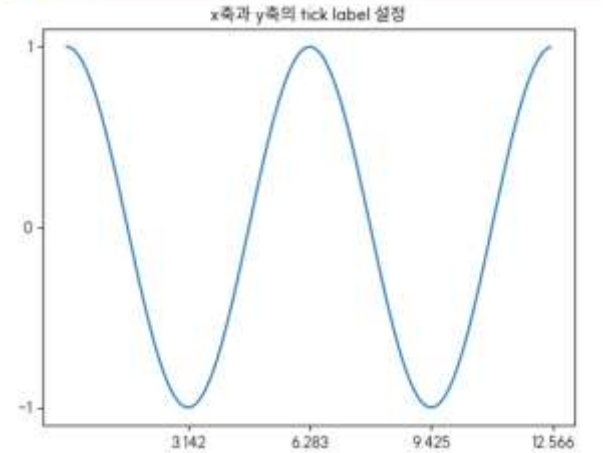
```
# 축 범위 설정
plt.title("color:magenta / marker:'o' / linestyle:'--'")
plt.plot([10,20,30,40],[1,4,9,16], c="m", marker="o", ls="--")
plt.xlim(0,50)
plt.ylim(-10,30)
plt.show()
```



tick 설정

- plot의 x축/y축 표시 지점을 tick이라고 하고, 이 tick에 써진 숫자/글자를 tick label이라고 표현
- 기본적으로 matplotlib에서 자동으로 설정하지만 수동으로 설정하고 싶다면 xticks() / yticks() 를 사용

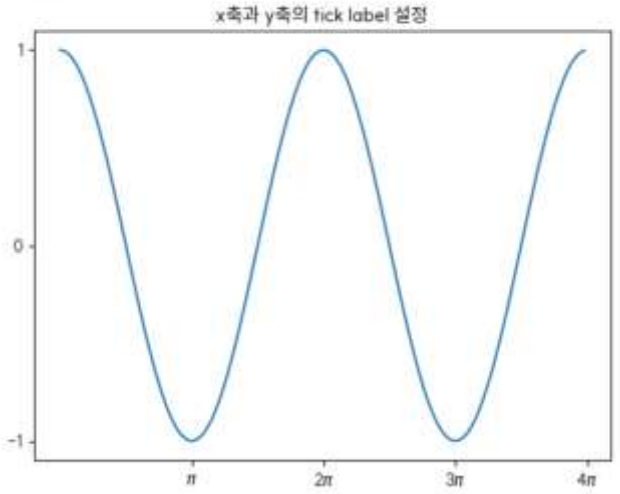
```
import numpy as np
x = np.arange(0, 4*np.pi, 0.1)
c = np.cos(x)
plt.plot(x,c)
plt.title("x축과 y축의 tick label 설정")
plt.xticks([np.pi, np.pi*2, np.pi*3, np.pi*4])
plt.yticks([-1,0,1])
plt.show()
```



tick 설정

- tick label 문자열에 $수학 문자식$을 넣으면 수식 표현 가능

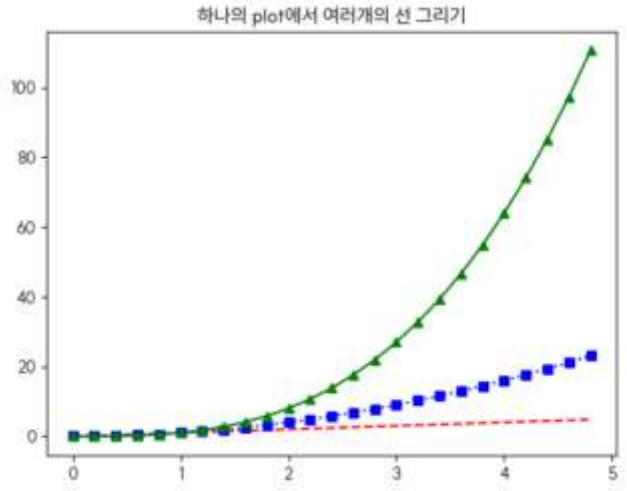
```
import numpy as np
x = np.arange(0, 4*np.pi, 0.1)
c = np.cos(x)
plt.plot(x,c)
plt.title("x축과 y축의 tick label 설정")
plt.xticks([np.pi, np.pi*2, np.pi*3, np.pi*4],
           [r'$\pi$',r'2$\pi$',r'3$\pi$',r'4$\pi$'])
plt.yticks([-1,0,1])
plt.show()
```



여러 개의 선 그리기 - 하나의 plot

- 하나의 plot에서 선을 하나가 아니라 여러 개를 그리고 싶은 경우에는 x 데이터, y 데이터, 스타일 문자열을 반복하여 인수로 넘김.
- 이 경우에는 하나의 선을 그릴 때 처럼 x 데이터나 스타일 문자열을 생략할 수 없음.

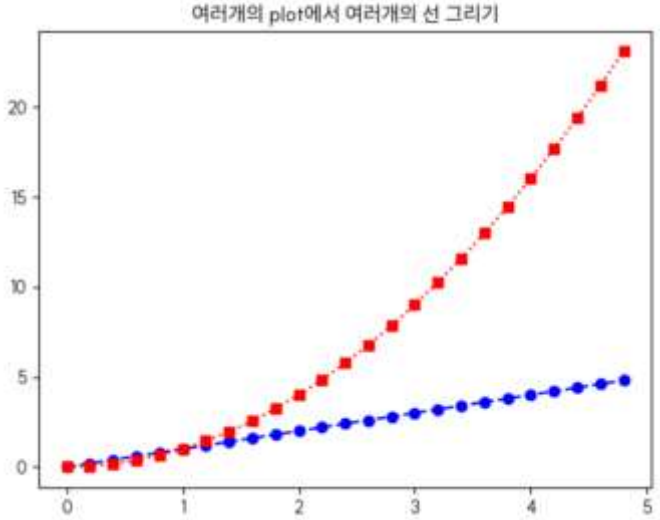
```
t = np.arange(0,5,0.2)
plt.title("하나의 plot에서 여러개의 선 그리기")
plt.plot(t, t, 'r--',
         t, t**2, 'bs:',
         t, t**3, 'g^-')
plt.show()
```



여러 개의 선 그리기 - 여러 개의 plot

- 여러 개의 plot을 그리게되면 하나의 plot에 합칠 수 있습니다.

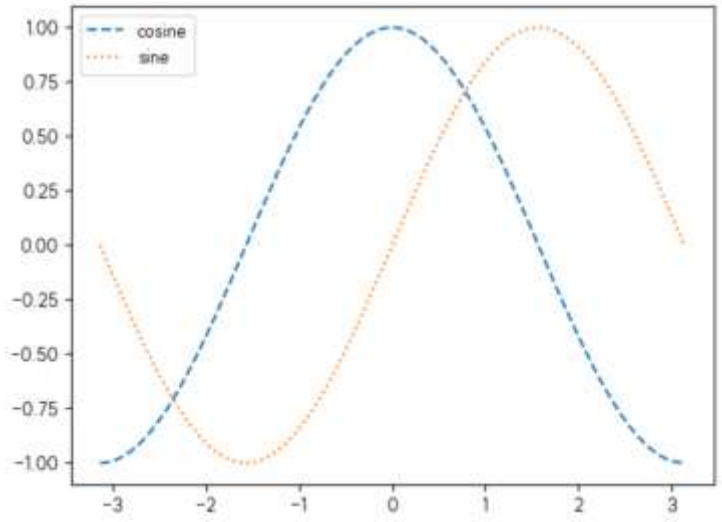
```
t = np.arange(0,5,0.2)
plt.title("여러개의 plot에서 여러개의 선 그리기")
style1 = {"c":"b","ls":"--","marker":"o"}
style2 = {"c":"r","ls":":","marker":"s"}
plt.plot(t, t, **style1)
plt.plot(t, t**2, **style2)
plt.show()
```



범례(Legend)

- 범례를 지정하기 위해서는 각 plot에 label을 적어주고 plt.legend() 함수를 사용

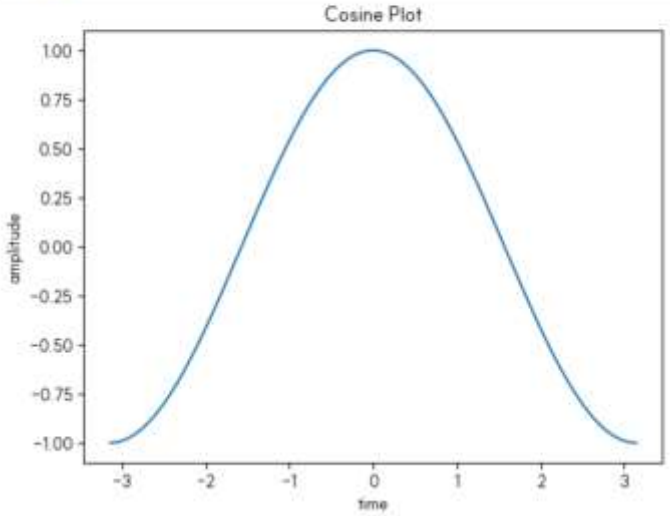
```
x = np.linspace(-np.pi, np.pi, 256)
c, s = np.cos(x), np.sin(x)
plt.plot(x, c, ls="--", label="cosine")
plt.plot(x, s, ls=":", label="sine")
plt.legend(loc=0) # loc=0 좌측 상단에 범례를 표시
plt.show()
```



x/y 축 label

- x/y축의 label을 작성하기 위해서는 plt.xlabel() / plt.ylabel() 메소드를 사용

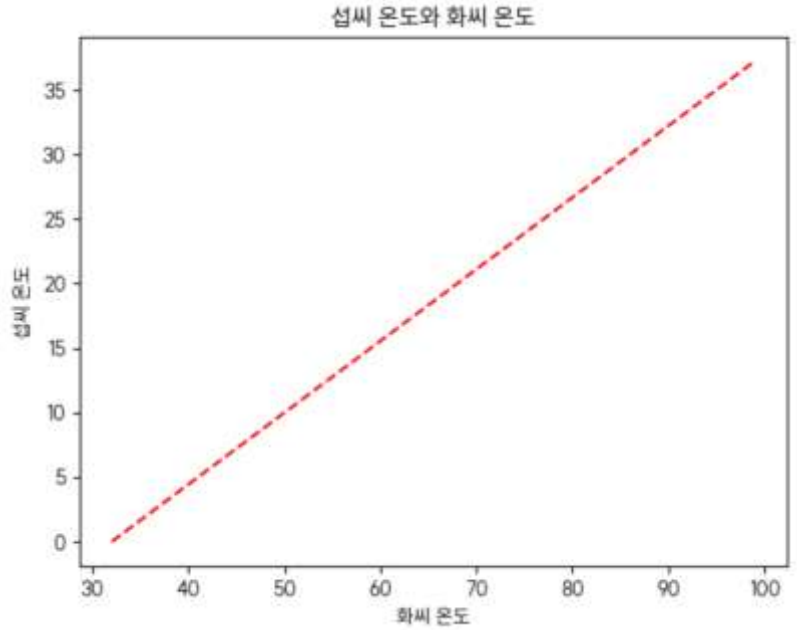
```
x = np.linspace(-np.pi, np.pi, 256)
c, s = np.cos(x), np.sin(x)
plt.title("Cosine Plot")
plt.xlabel("time")
plt.ylabel("amplitude")
plt.plot(x, c, ls="-")
plt.show()
```



연습 문제

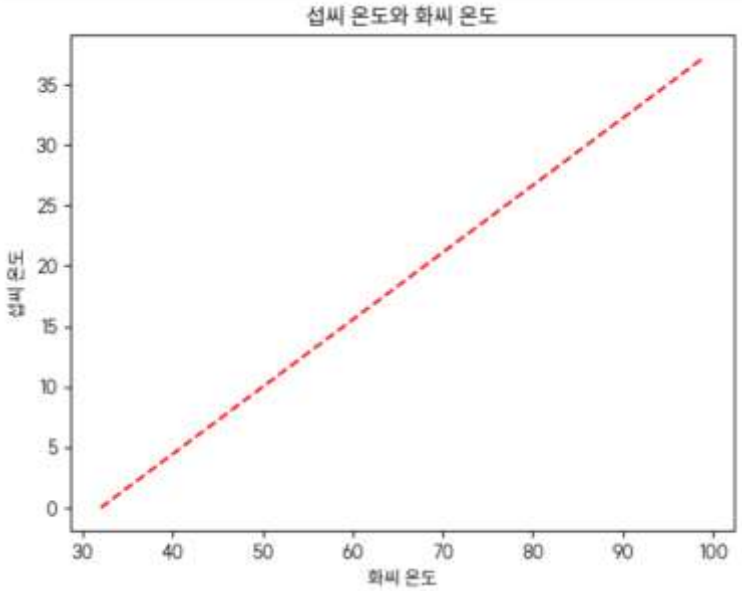
다음과 같은 그래프를 그려보세요.

- title: 섭씨 온도와 화씨 온도
- xlabel: 화씨 온도
- ylabel: 섭씨 온도
- x 값은 화씨로 32이상~100미만의 정수,
- y 값은 섭씨로 $℃ = (℉ - 32) \times 5/9$



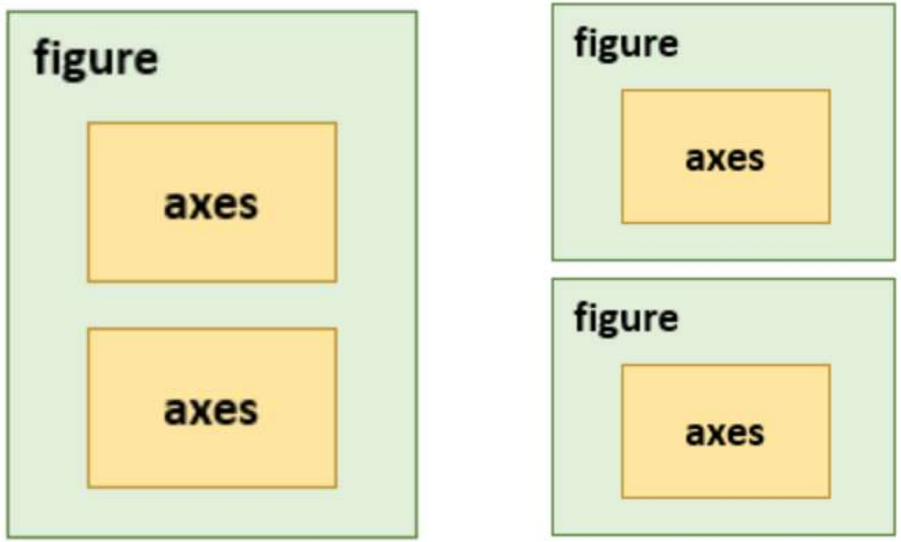
연습 문제 해답

```
x = np.arange(32,100)
y = (x-32) * (5/9)
plt.title("섭씨 온도와 화씨 온도")
plt.xlabel("화씨 온도")
plt.ylabel("섭씨 온도")
plt.plot(x,y,c="r",ls="--")
plt.show()
```



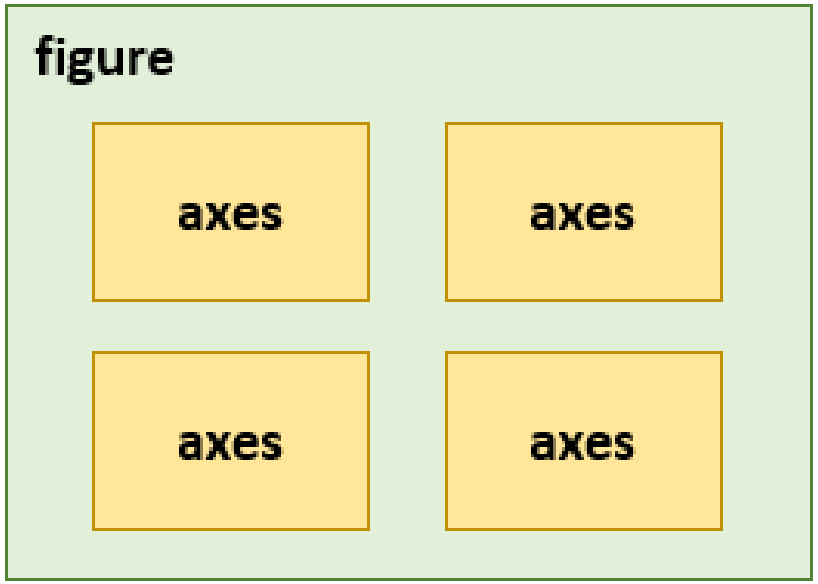
Figure, Axes를 사용하는 이유

- 앞서 여러개의 plot을 그리는 방법을 설명했다. figure와 axes를 사용하면 더 유연하게 여러 개의plot을 그릴 수 있는 장점이 있다.
- 또한 여러개의 plot을 그릴 때 코드를 훨씬 간결하게 사용할 수 있다.



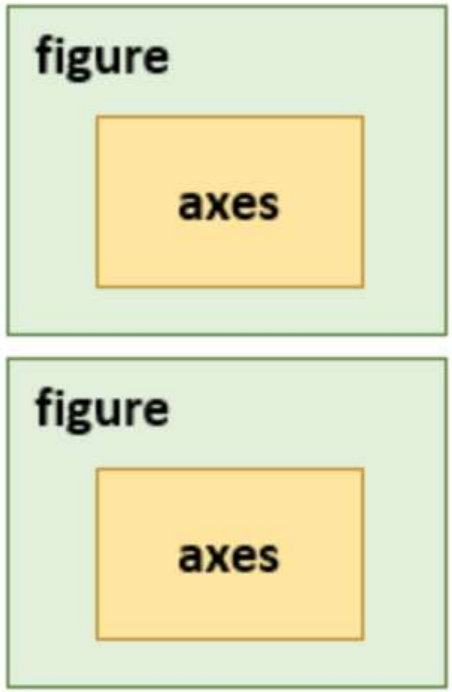
Figure, Axes

-figure는 그림을 그리기 위한 전체 프레임이고, axes는 그래프가 그려지는 캔버스

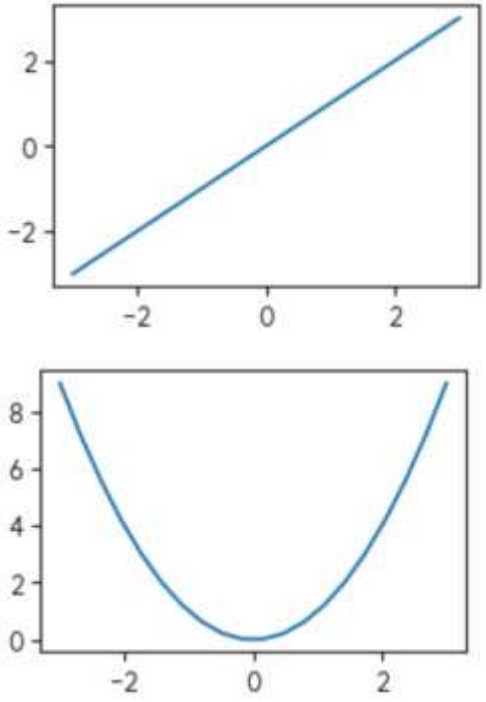


Figure, Axes를 사용하는 이유

- 2개의 프레임에 각각의 그래프를 그리려면 2개의 figure가 필요



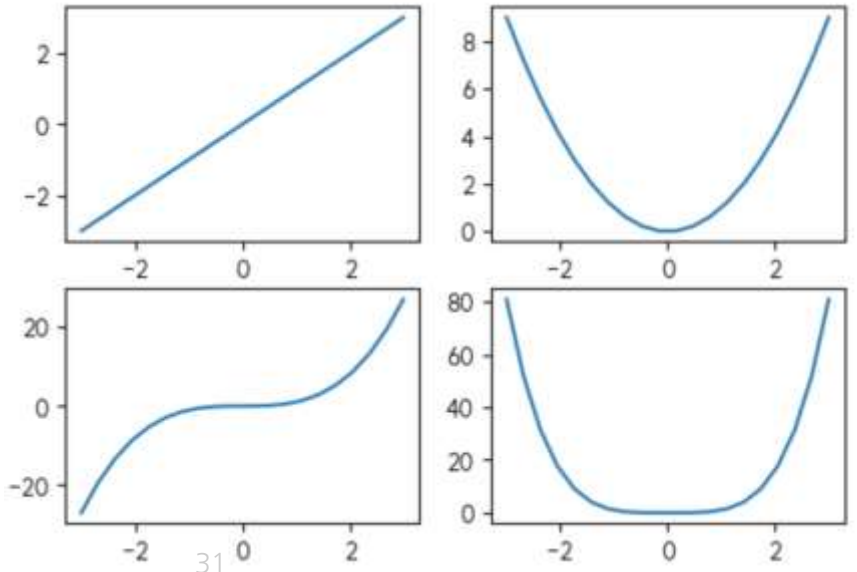

```
x = np.linspace(-3, 3, 20)
y1 = x
y2 = x ** 2

# 첫 번째 figure
plt.figure(figsize=(3, 2))
plt.plot(x, y1)

# 두 번째 figure
plt.figure(figsize=(3, 2))
plt.plot(x, y2)

plt.show()
```



Figure, Axes를 사용하는 이유 - subplot

- 하나의 figure에 여러개의 그래프를 그리려면 axes를 원하는 위치에 배치하기만 하면 된다.
- 이때 필요한 것이 subplot 이고, subplot은 grid의 형태를 가진다.
- plt.subplot(행의 수, 열의 수, 플롯 번호)로 plot을 배치할 수 있다.

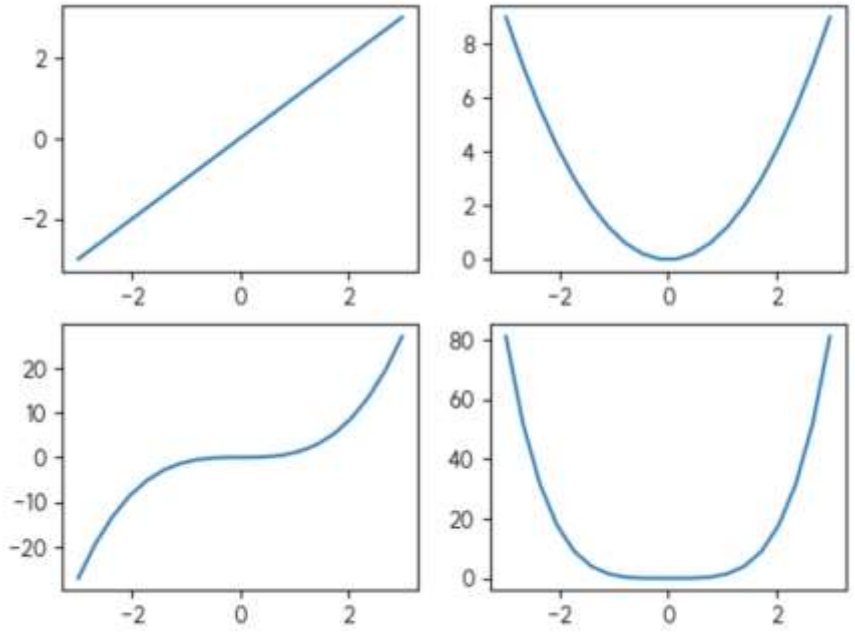
```
x = np.linspace(-3, 3, 20)
y1 = x
y2 = x ** 2
y3 = x ** 3
y4 = x ** 4

plt.figure(figsize=(6,4))

# 왼쪽 위
plt.subplot(2,2,1)
plt.plot(x, y1)

# 오른쪽 위
plt.subplot(2,2,2)
plt.plot(x, y2)

# 왼쪽 아래
plt.subplot(2,2,3)
plt.plot(x, y3)

# 오른쪽 아래
plt.subplot(2,2,4)
plt.plot(x, y4)

plt.show()
```



Figure, Axes를 사용하는 이유 - subplot

- Figure, Axes와 subplot을 사용하면 여러 개의 그래프를 반복문으로 쉽게 표현 가능

```
x = np.linspace(-3, 3, 20)
y_list = [x, x**2, x**3, x**4]


figure, axes = plt.subplots(2,2)

plot_number = 0
for i in range(2):
    for j in range(2):
        axes[i][j].plot(x,y_list[plot_number])
        plot_number += 1

plt.figure(figsize=(6,4))
plt.show()
```

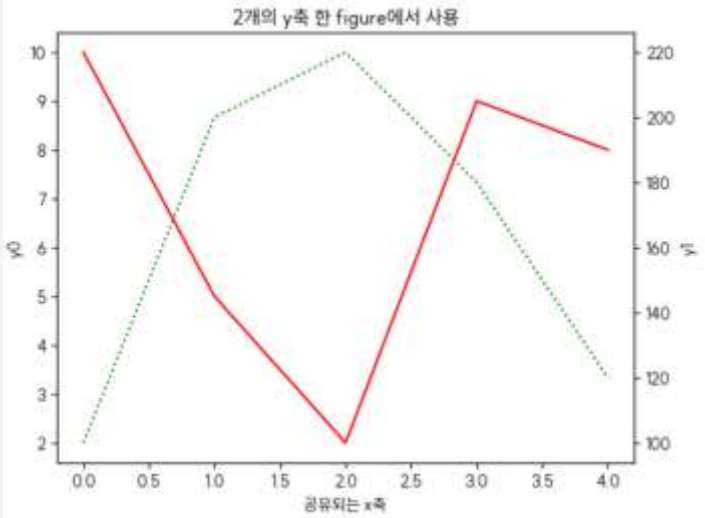


```
<Figure size 600x400 with 0 Axes>
```

Figure, Axes를 사용하는 이유 - subplot

- 각 축을 공유 하거나 분리하여 표시할 수 있다.

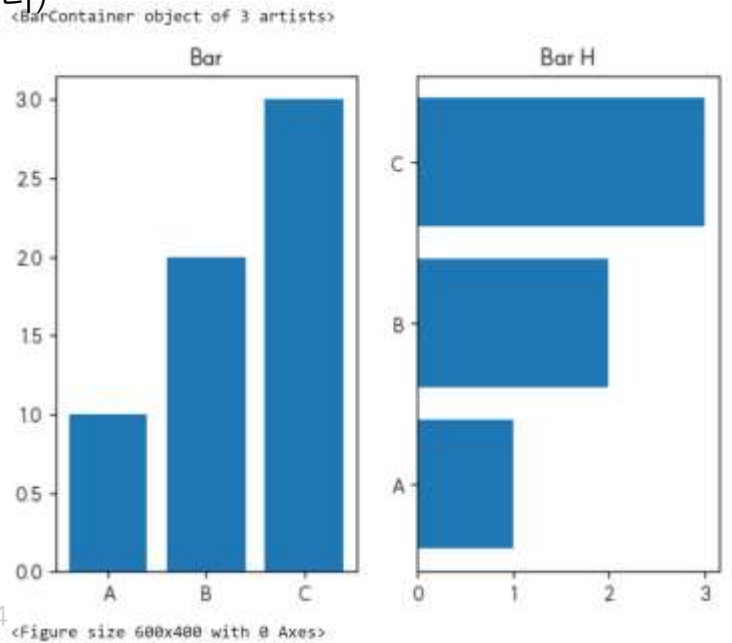
```
fig, ax0 = plt.subplots()
ax1 = ax0.twinx() # 공유되는 x축 설정

ax0.set_title("2개의 y축 한 figure에서 사용")

ax0.plot([10, 5, 2, 9, 8], 'r-', label="y0")
ax0.set_ylabel("y0")


ax1.plot([100,200, 220, 180, 120], 'g:', label="y1")
ax1.set_ylabel("y1")

ax0.set_xlabel("공유되는 x축")
plt.show()
```



바 차트(bar chart)

- x 데이터가 카테고리 값인 경우 바 차트(bar chart)를 그릴 수 있다.
- 세로 방향으로 바 차트를 그리려면 bar(x데이터, y데이터)
- 가로 방향으로 바 차트를 그리려면 barh(x데이터, y데이터)

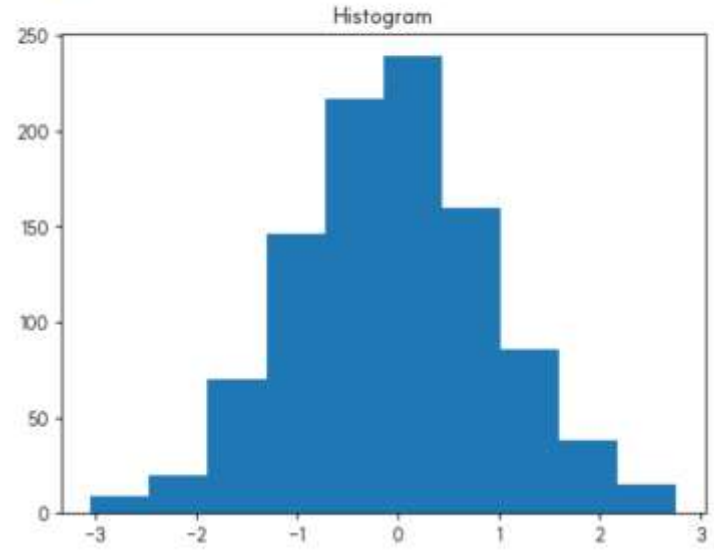
```
y = [1,2,3]
x = ['A', 'B', 'C']
fig, ax = plt.subplots(1,2)

plt.figure(figsize=(6,4))
ax[0].set_title("Bar")
ax[0].bar(x,y)

ax[1].set_title("Bar H")
ax[1].barh(x,y)
```



히스토그램(histogram)

- hist()는 히스토그램을 그리기위한 메소드
- bins 인수로 데이터를 집계 할 구간 정보를 받고, 반환값으로 데이터 집계 결과를 반환

```
np.random.seed(0)
x = np.random.randn(1000)
plt.title("Histogram")
arrays, bins, patches = plt.hist(x, bins=10)
plt.show()
```



bins = 히스토그램의 가로축 구간의 개수

arrays = 그래프에 표현된 데이터

patches = 히스토그램의 관측치

산점도(scatter)

- scatter 명령은 산점도를 그리는 메소드
- scatter plot의 점 하나의 위치는 데이터 하나의 x v 값

```
np.random.seed(0)
x = np.random.normal(0, 1, 100)
y = np.random.normal(0, 1, 100)
plt.title("Scatter Plot")
plt.scatter(x,y)
plt.show()
```