# 기초빅데이터프로그래밍

## MATPLOTLIB
## Numpy
## Pandas


서강대학교
SOGANG UNIVERSITY

# matplotlib

- 자료를 **chart나 plot으로 시각화(visualization)하는 패키지**
- Platform을 초월해서 다양한 포맷과 인터렉티브한 환경에서 publication quality figure를 제공할 수 있는 Python 2D plotting library
- matplotlib는 다음과 같은 정형화된 차트나 플롯 이외에도 저수준 API를 사용한 다양한 시각화 기능을 제공한다.
  - line plot
  - scatter plot
  - contour plot
  - surface plot
  - bar chart
  - histogram
  - box plot

- matplotlib를 사용한 시각화 예제들을 다음 웹사이트에서 볼 수 있다.

  http://matplotlib.org/gallery.html
- Seaborn : matplotlib 을 바탕으로 통계 분석 결과의 시각화 라이브러리, 기본적인 시각화 기능은 Matplotlib 패키지에 의존하며 통계 기능은 Statsmodels 패키지에 의존한다.

  http://seaborn.pydata.org/
- Bokeh: 임의의 데이터 시각화를 지원, interactive plots 을 그릴 수 있다
- Folium: 지도 위에 데이터를 interactive하게 표현해 주는 대표적인 파이썬 **지도 시각화** 라이브러리

# pyplot 서브패키지

- matplotlib 패키지에는 pyplot 라는 서브패키지가 존재한다.
- **MATLAB**의 수치해석 소프트웨어의 시각화 명령을 거의 그대로 사용할 수 있도록 matplotlib 의 하위 API를 포장(wrapping)한 명령어 집합을 제공한다.
- 간단한 시각화 프로그램은 pyplot 서브패키지의 명령만으로도 충분하다.

- matplotlib 패키지를 사용할 때는 보통 주 패키지는 **mpl** 이라는 alias로 임포트하고,
- pyplot 서브패키지는 **plt** 라는 alias로 별도 임포트하여 사용하는 것이 관례이다.

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

# Line plot

- 가장 간단한 플롯은 선을 그리는 line plot이다.
- 라인 플롯은 데이터가 시간, 순서 등에 따라 어떻게 변화하는지 보여준다.

- 명령은 pyplot 서브패키지의 **plot** 명령을 사용한다.

- #matplotlib.pyplot.plot

만약 데이터가 1, 4, 9, 16 으로 변화하였다면, plot 명령에 데이터 리스트 혹은 ndarray 객체를 넘긴다.

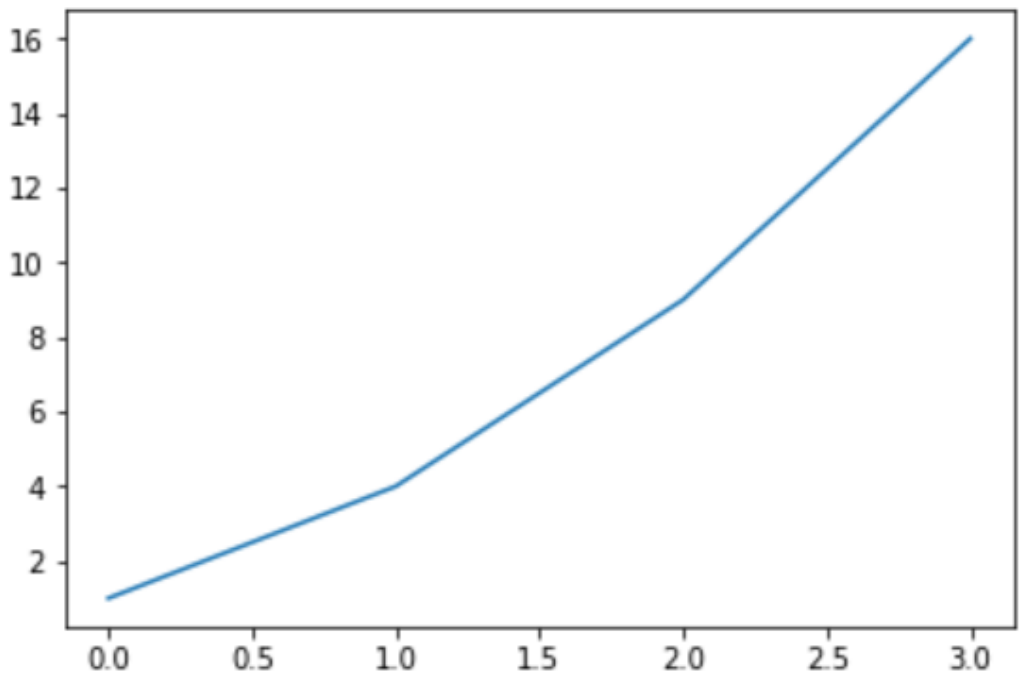
```
plt.plot([1, 4, 9, 16])
plt.show()
```

- 만약 이 x tick 위치를 별도로 명시하고 싶다면 다음과 같이 두 개의 같은 길이의 리스트 혹은 배열 자료를 넣는다.

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

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

```
plt.plot([1,4,9,16])
plt.show()
```

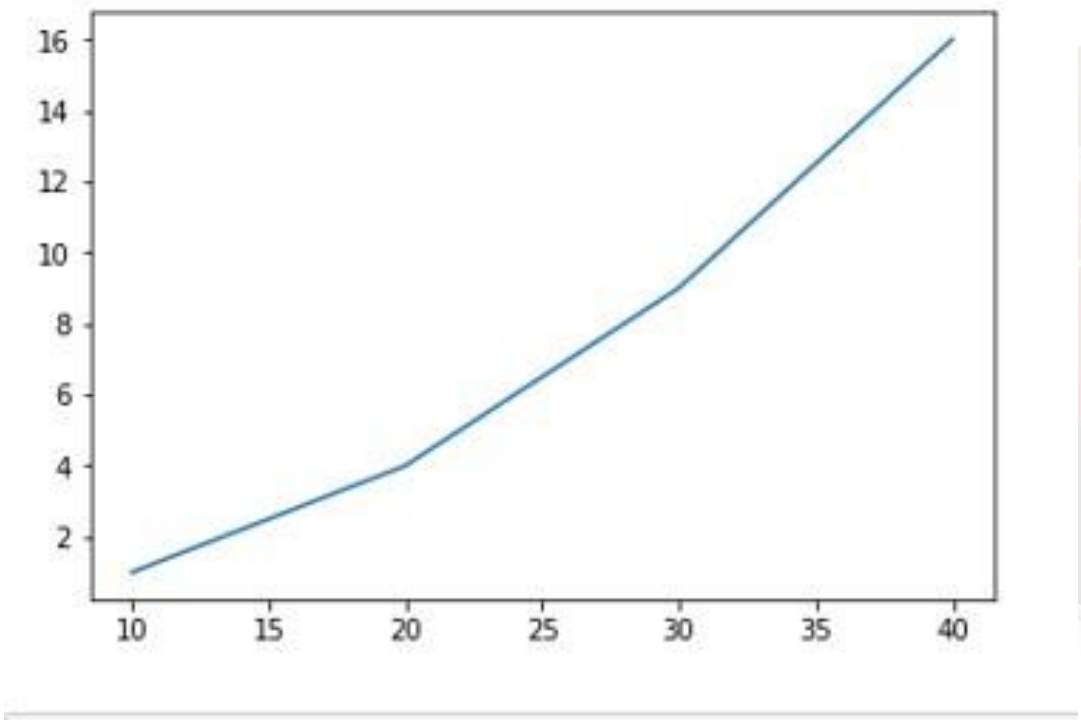



이 때 **x 축의 자료 위치** 즉, **틱(tick)**은 자동으로 0, 1, 2, 3 이 된다.

즉, 하나의 리스트만 넣으면 **리스트의 index**가 x축, 값을 y축으로 인식해서 그래프 표시

- x tick 위치를 별도로 명시하고 싶다면 다음과 같이 두 개의 같은 길이의 리스트 혹은 배열 자료를 넣는다

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



# show( )명령어

- show 명령은 시각화 명령을 실제 차트로 rendering하고 마우스 움직임 등의 이벤트를 기다리라는 지시이다.

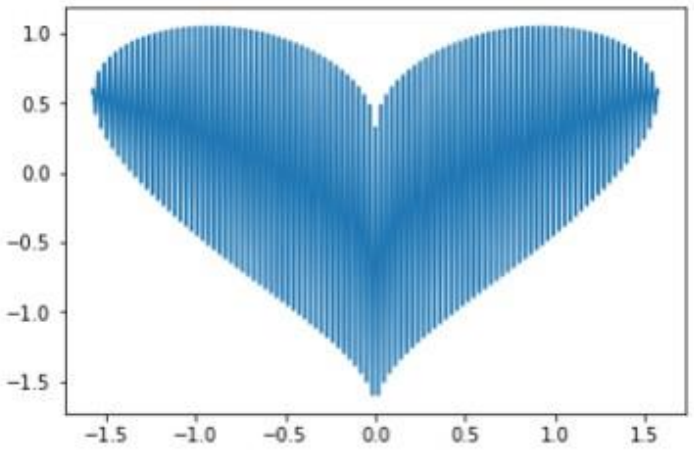
- 만약 외부 렌더링을 하지 않고 IPython이나 Jupyter 노트북에서 내부 플롯(inline plot)을 사용하도록 다음과 같이(%matplotlib inline) 미리 설정하였다면 별도의 이벤트 처리를 할 수 없기 때문에 show 명령을 추가적으로 지시하지 않아도 자동으로 그림이 그려진다.

- Jupyter 노트북은 서버측에서 가동되므로 반드시 내부 플롯을 사용해야 한다.

  **%matplotlib inline**

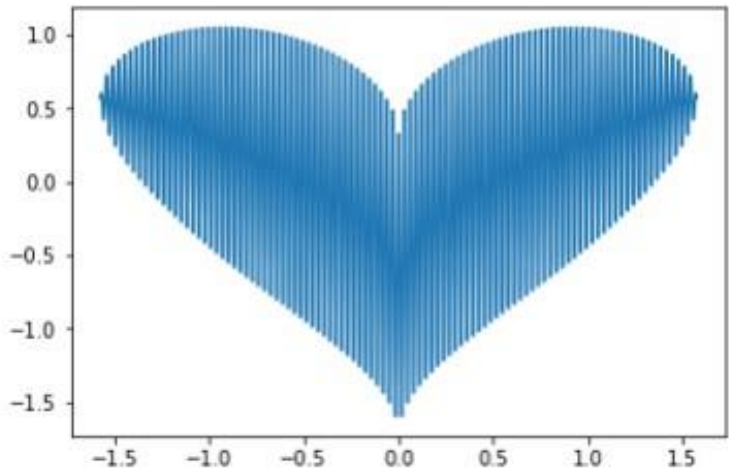
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1.6, 1.6, 10000)
f = lambda x: (np.sqrt(np.cos(x)) * np.cos(200*x) + np.sqrt(abs(x))-0.7)* pow((4-x*x),0.01)
plt.plot(x, list(map(f, x)))
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipyker
```




```
from pylab import *

x = linspace(-1.6, 1.6, 10000)
f = lambda x: (sqrt(cos(x)) * cos(200*x) + sqrt(abs(x))-0.7)* \
    pow((4-x*x),0.01)
plot(x, list(map(f, x)))
show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\__main__.p
```

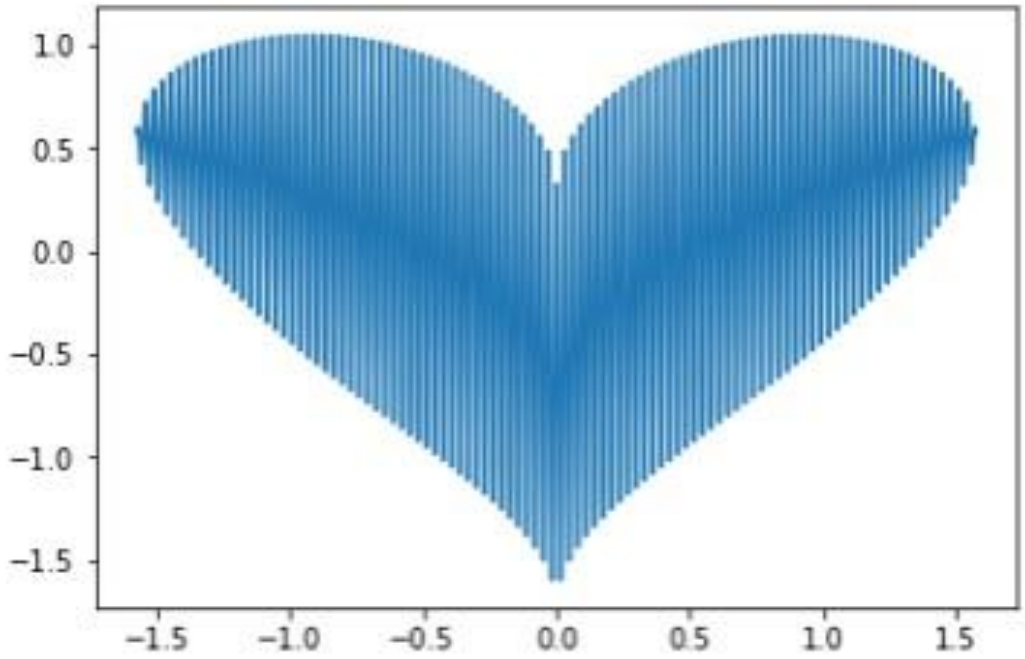



간단한 시각화 프로그램을 만드는 경우에는 pylab 서브패키지의 명령만으로도 충분하다.

```
%matplotlib inline
from pylab import *

x = linspace(-1.6, 1.6, 10000)
f = lambda x: (sqrt(cos(x)) * cos(200*x) + sqrt(abs(x))-0.7)* \
    pow((4-x*x),0.01)
plot(x, list(map(f, x)))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\__main__.py
```

```
[<matplotlib.lines.Line2D at 0x1bedd4fbcc0>]
```




**%matplotlib inline**
실행하면 show() 생략해도 자동으로 그림이 그려진다.

단, inline은 특정cell에서 plot생성하면 추가조작이 불가능하다.

```
%matplotlib nbagg
from pylab import *

x = linspace(-1.6, 1.6, 10000)
f = lambda x: (sqrt(cos(x)) * cos(200*x) + sqrt(abs(x))-0.7)* \
    pow((4-x*x),0.01)
plot(x, list(map(f, x)))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\__main__.py:5: RuntimeWarn
```

Figure 1


```
[<matplotlib.lines.Line2D at 0x218d771b7b8>]
```

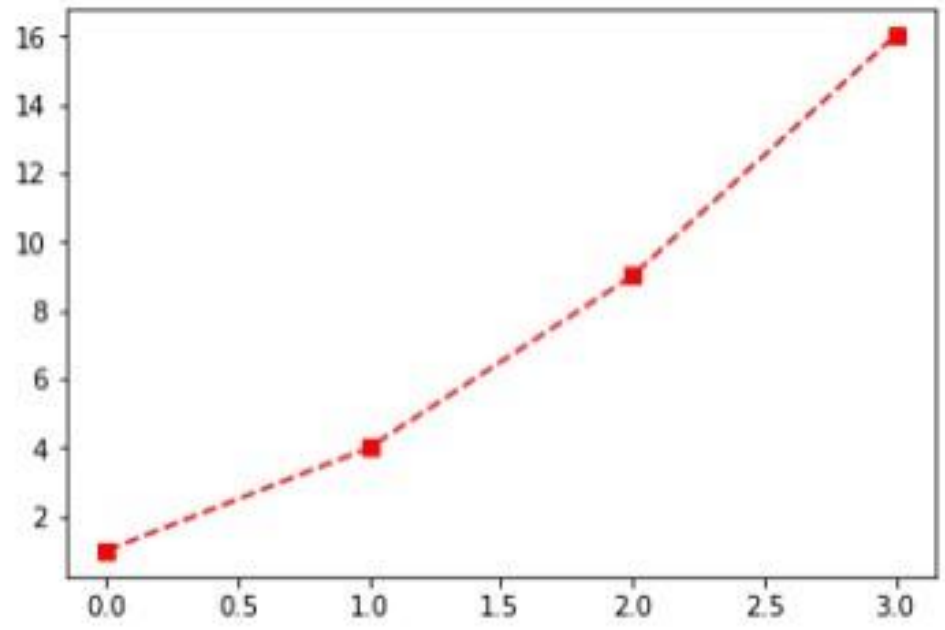
%matplotlib nbagg실행하면 show() 생략해도 자동으로 그림이 그려진다.

nbagg는 interactive하게 조작가능하다.

# 스타일 지정

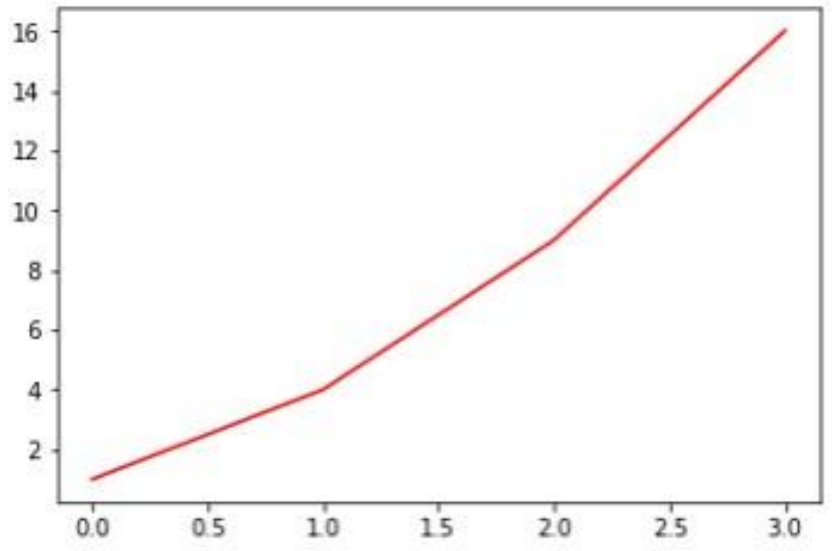
- 플롯 명령어는 다양한 style을 지원한다.
- plot 명령어에서는 추가 문자열 인수를 사용하여 스타일을 지원한다.
- **스타일 문자열은 색깔(color), 마커(marker), 선 종류(line style)**의 순서로 지정한다. 만약 이 중 일부가 생략되면 디폴트값이 적용된다.

```
plt.plot([1,4,9,16], 'rs--')
plt.show()
```

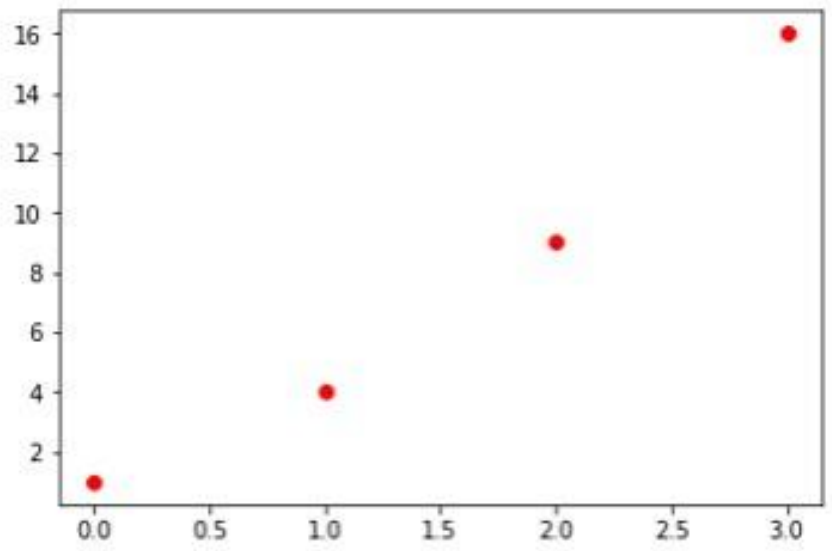



r: red
s: square marker
-- : dashed line style

```
plt.plot([1,4,9,16], "r")
plt.show()
```




```
plt.plot([1,4,9,16], 'ro')
plt.show()
```



# 색

- 색을 지정하는 방법은 색 이름 혹은 약자를 사용하거나 # 문자로 시작되는 RGB코드를 사용한다.
- 전체 색 목록은 다음 웹사이트를 참조한다.

  http://matplotlib.org/examples/color/named_colors.html
- 자주 사용되는 색깔은 한글자 약자를 사용할 수 있다.

| 문자열 | 약자 |
|---|---|
| blue | b |
| green | g |
| red | r |
| cyan | c |
| magenta | m |
| yellow | y |
| black | k |
| white | w |

# named_colors.py

# 마커

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

| 마커 문자열 | 의미 |
|---|---|
| . | point marker |
| , | pixel marker |
| o | circle marker |
| v | triangle_down marker |
| ^ | triangle_up marker |
| < | triangle_left marker |
| > | triangle right marker |
| 1 | tri_down marker |
| 2 | tri_up marker |
| 3 | tri_left marker |
| 4 | tri_right marker |
| s | square marker |
| p | pentagon marker |
| * | star marker |
| h | hexagon1 marker |
| H | hexagon2 marker |
| + | plus marker |
| x | x marker |
| D | diamond marker |
| d | thin_diamond marke |

# 선 스타일

- 선 스타일에는 **실선(solid), 대시선(dashed), 점선(dotted), 대시-점선(dash-dot)** 이 있다.
- 지정 문자열은 다음과 같다.

| 선 스타일 문자열 | 의미 |
|---|---|
| - | solid line style |
| -- | dashed line style |
| -. | dash-dot line style |
| : | dotted line style |

# 기타 스타일

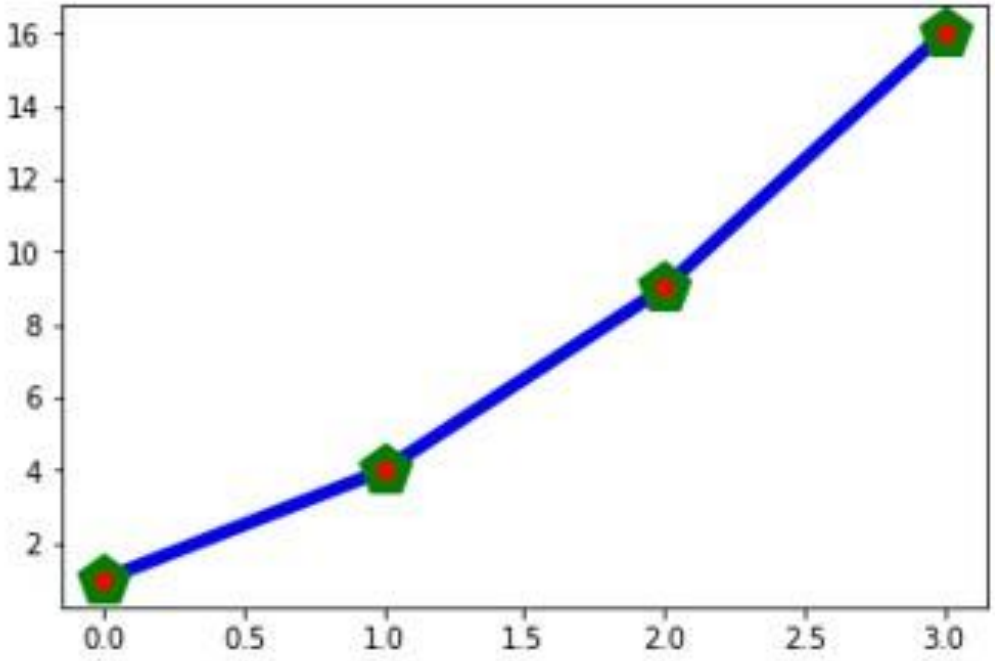
- 라인 플롯에서는 인수 이름을 사용해서 여러 가지 스타일을 지정할 수 있다.
- 사용할 수 있는 스타일 인수의 목록은 matplotlib.lines.Line2D 클래스에 대한 다음 웹사이트를 참조한다.

http://matplotlib.org/api/lines_api.html#matplotlib.lines.Line2D

| 스타일 문자열 | 약자 | 의미 |
|---|---|---|
| color | **c** | 선 색깔 |
| linewidth | **lw** | 선 굵기 |
| linestyle | **ls** | 선 스타일 |
| **marker** | | 마커 종류 |
| markersize | **ms** | 마커 크기 |
| markeredgecolor | **mec** | 마커 선 색깔 |
| markeredgewidth | **mew** | 마커 선 굵기 |
| markerfacecolor | **mfc** | 마커 내부 색깔 |

# 실습 1

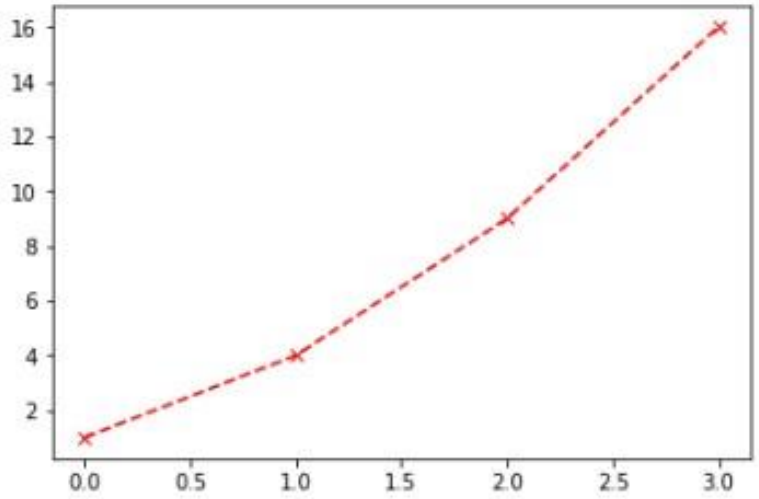
- 스타일 인수를 사용해서 아래와 같이 그리시오.



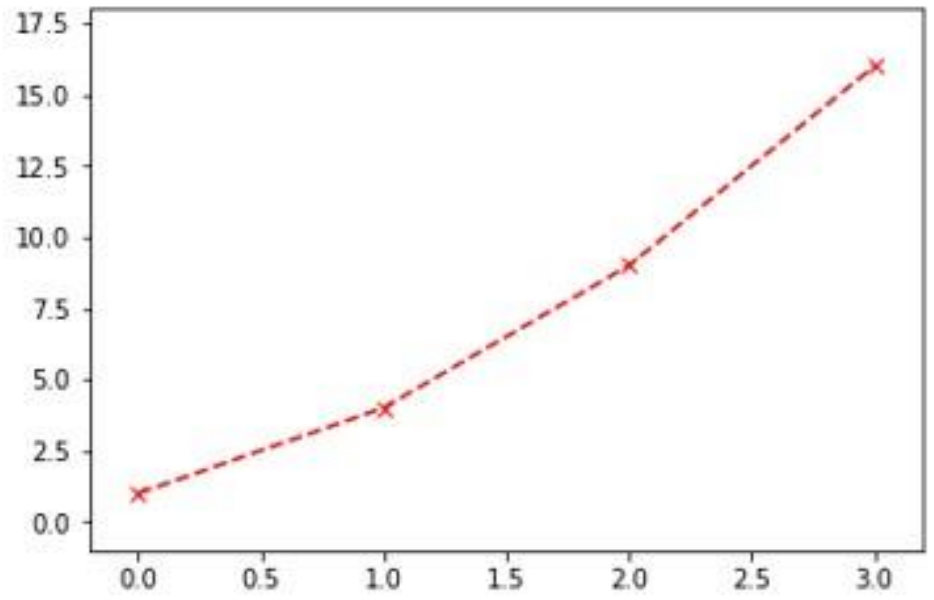
# 그림 범위 지정

- 플롯 그림을 보면 몇몇 점들은 그림의 범위 경계선에 있어서 잘 보이지 않는 경우가 있을 수 있다.
- 그림의 범위를 수동으로 지정하려면 **xlim** 명령과 **ylim** 명령을 사용한다.
- 이 명령들은 그림의 범위가 되는 x축, y축의 최소값과 최대값을 지정한다.

```
plt.plot([1,4,9,16], 'rx--')
plt.show()
```




```
plt.plot([1,4,9,16], 'rx--')
plt.xlim(-0.2, 3.2)
plt.ylim(-1, 18)
plt.show()
```

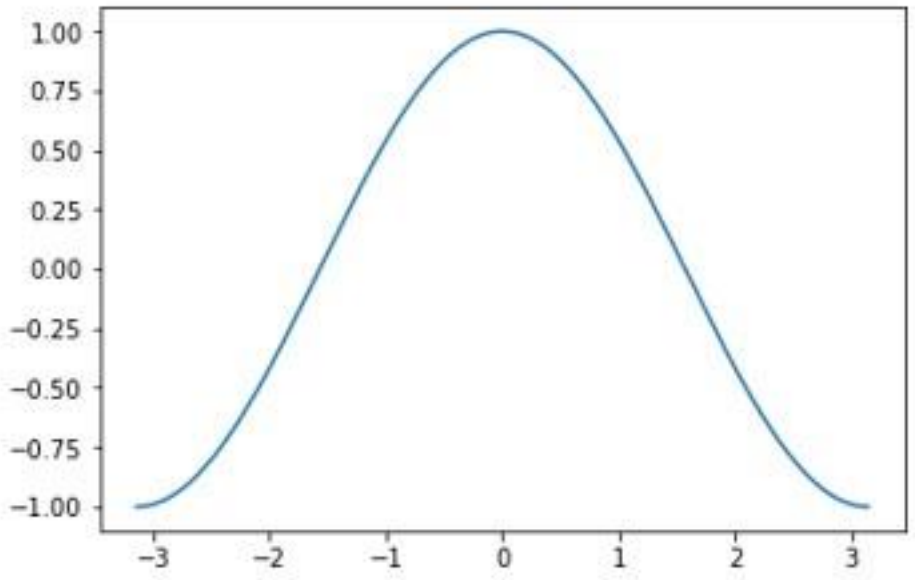


# 틱 설정

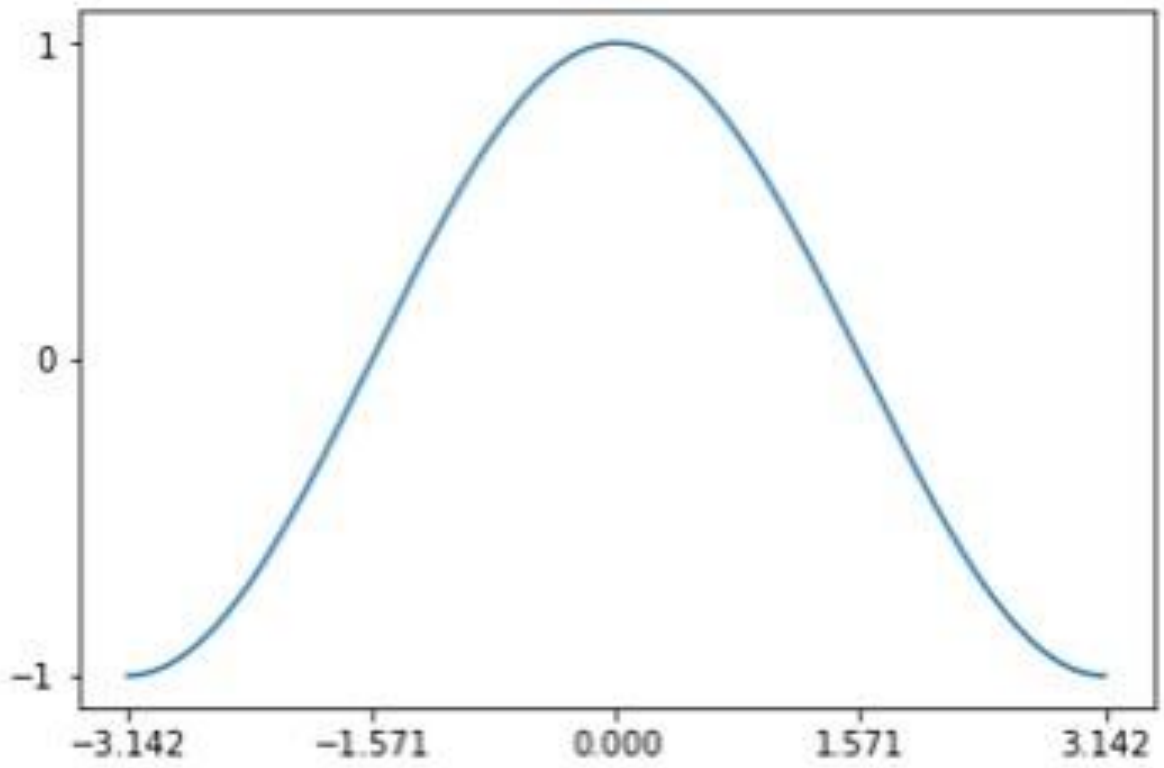
- 플롯이나 차트에서 축상의 위치 표시 지점을 **tick**이라고 하고 이 틱에 써진 숫자 혹은 글자를 틱 라벨(**tick label**)이라고 한다.
- **xticks 명령, yticks 명령을 사용하여 수동설정 가능하다.**

- 아래 plot에서 x축은 -**pi, -pi/2, 0, pi/2, pi**, y축은 **-1, 0, 1**로 표시하려면…

```
X = np.linspace(-np.pi, np.pi, 256)
C = np.cos(X)
plt.plot(X, C)
plt.show()
```

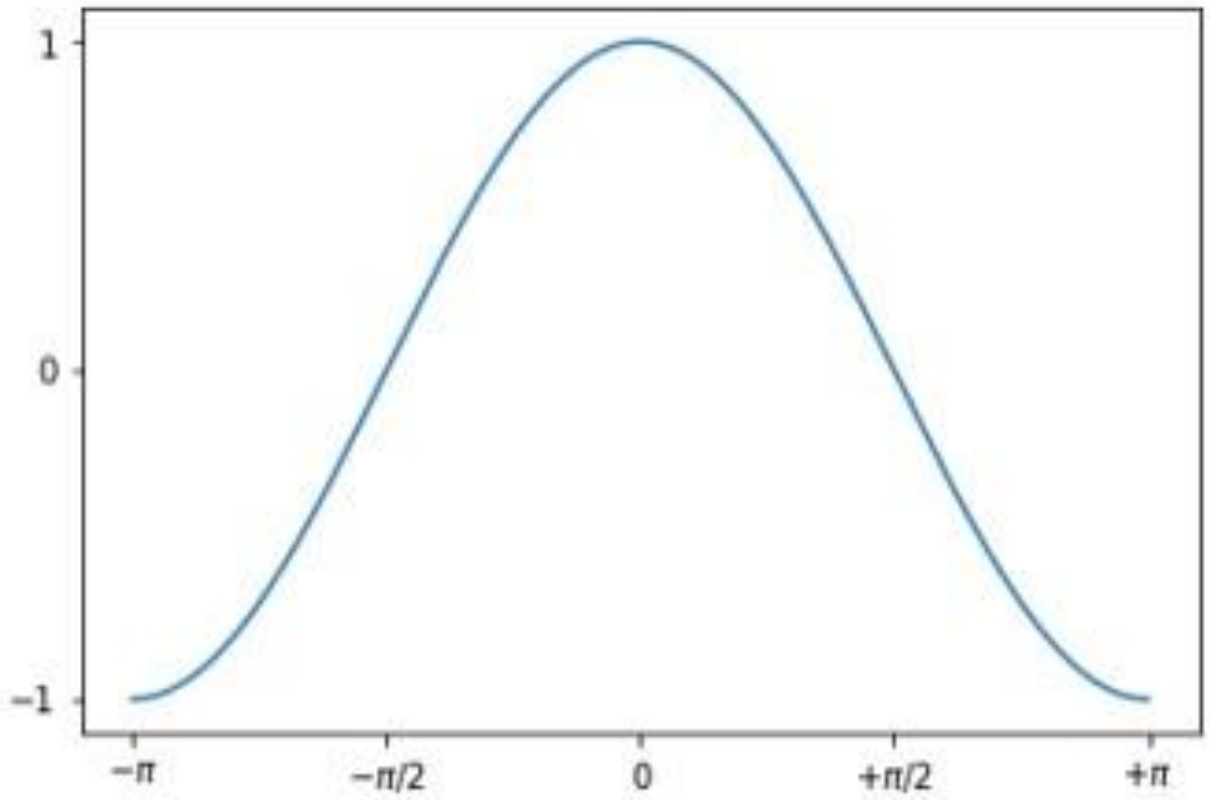


```
X = np.linspace(-np.pi, np.pi, 256)
C = np.cos(X)
plt.plot(X, C)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.yticks([-1, 0, +1])
plt.show()
```



# 실습 2

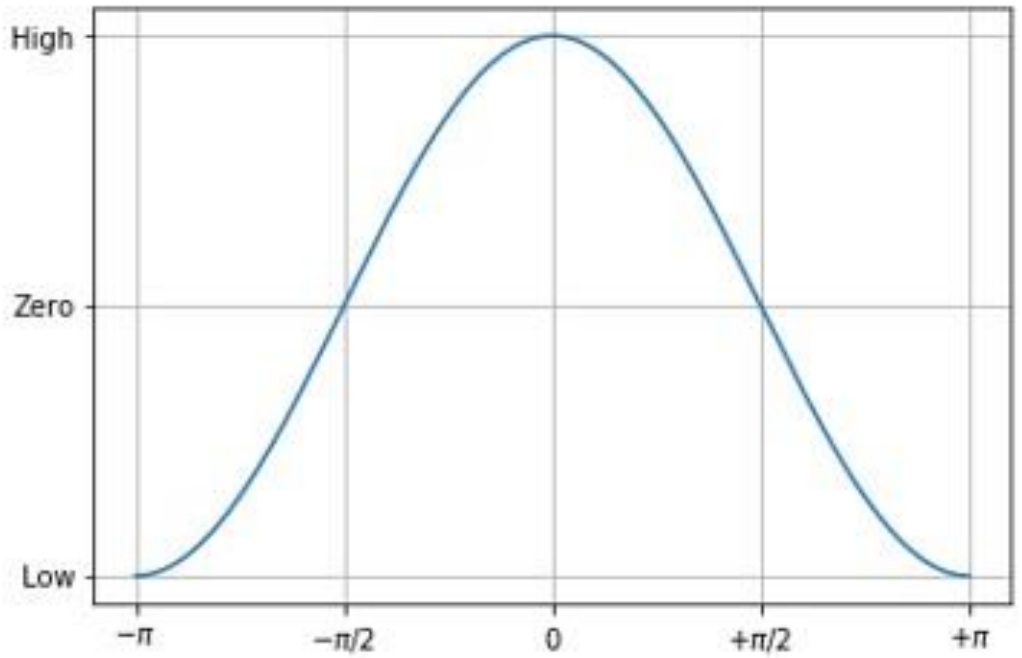
- 아래와 같은 cosine graph를 그리시오.
- 틱 라벨 문자열에서 **$$** 사이에 LaTeX 수학 문자식을 넣어 해결하시오.



# 그리드 설정

- 그리드를 사용하지 않으려면 **grid(False)** 명령을 사용한다.
- 다시 그리드를 사용하려면 **grid(True)**를 사용한다.

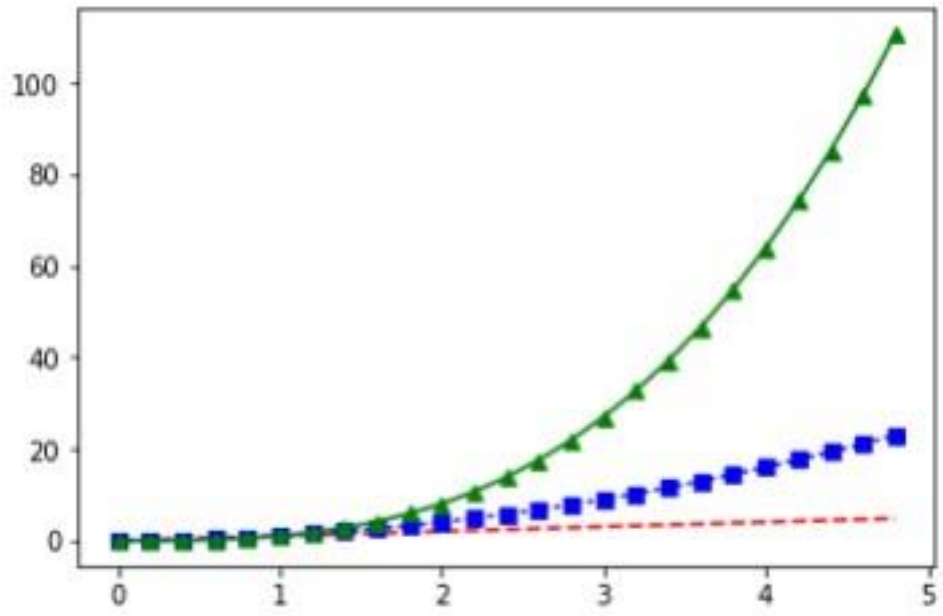
```
plt.grid(True)
plt.show()
```



# 여러 개의 선을 그리기

- 라인 플롯에서 여러 개의 선을 그리기 위해서는 x 데이터, y 데이터, 스타일 문자열을 반복하여 인수로 넘긴다.
- 이 경우에는 하나의 선을 그릴 때 처럼 x 데이터나 **스타일 문자열을 생략할 수 없다.**

```
t = np.arange(0., 5., 0.2)
plt.plot(t, t, 'r--', t, t**2, 'bs:', t, t**3, 'g^-')
plt.show()
```

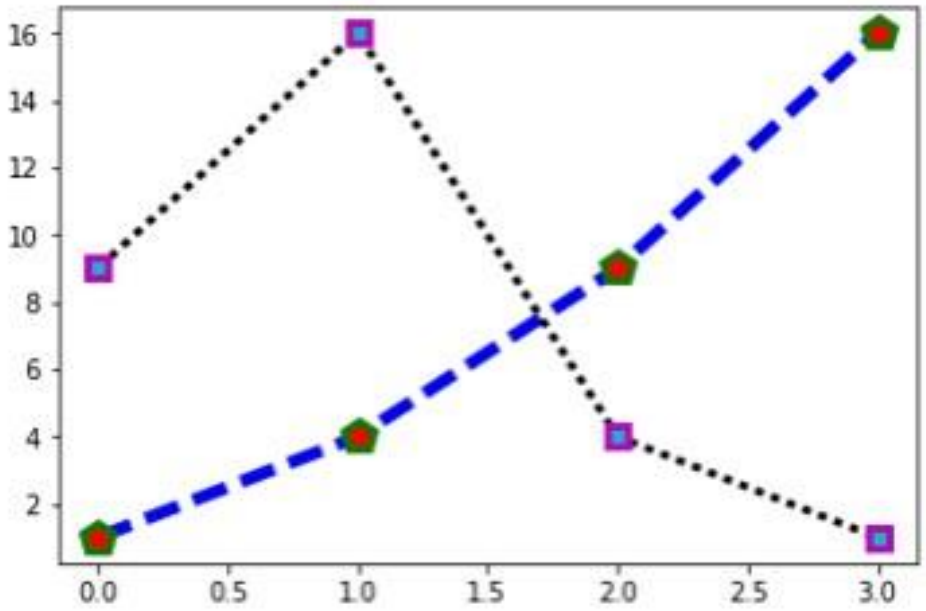


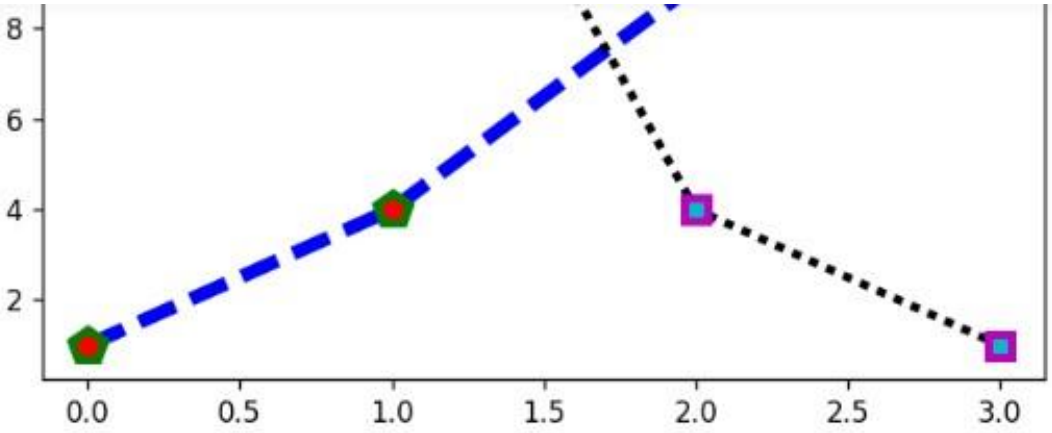

y = x
y = x**2
y = x**3
에 대한 함수의 그래프를 표현

# Plot 겹쳐 그리기 : hold(True) 명령

- 하나의 plot 명령이 아니라 복수의 plot 명령을 하나의 그림에 겹쳐서 그릴 수도 있다. 기존의 그림 위에 겹쳐 그리도록 하는 명령은 **hold(True)** 이다.
- 겹치기를 종료하는 것은 hold(False) 이다. (1.5.1 버전 이전에서 필요)

```
plt.plot([1,4,9,16], c="b", lw=5, ls="--", marker="p", ms=12, mec="g", mew=3, mfc="r")
plt.hold(True)
plt.plot([9,16, 4, 1], c="k", lw=3, ls=":", marker="s", ms=8, mec="m", mew=3, mfc="c")
plt.hold(False)
plt.show()
```





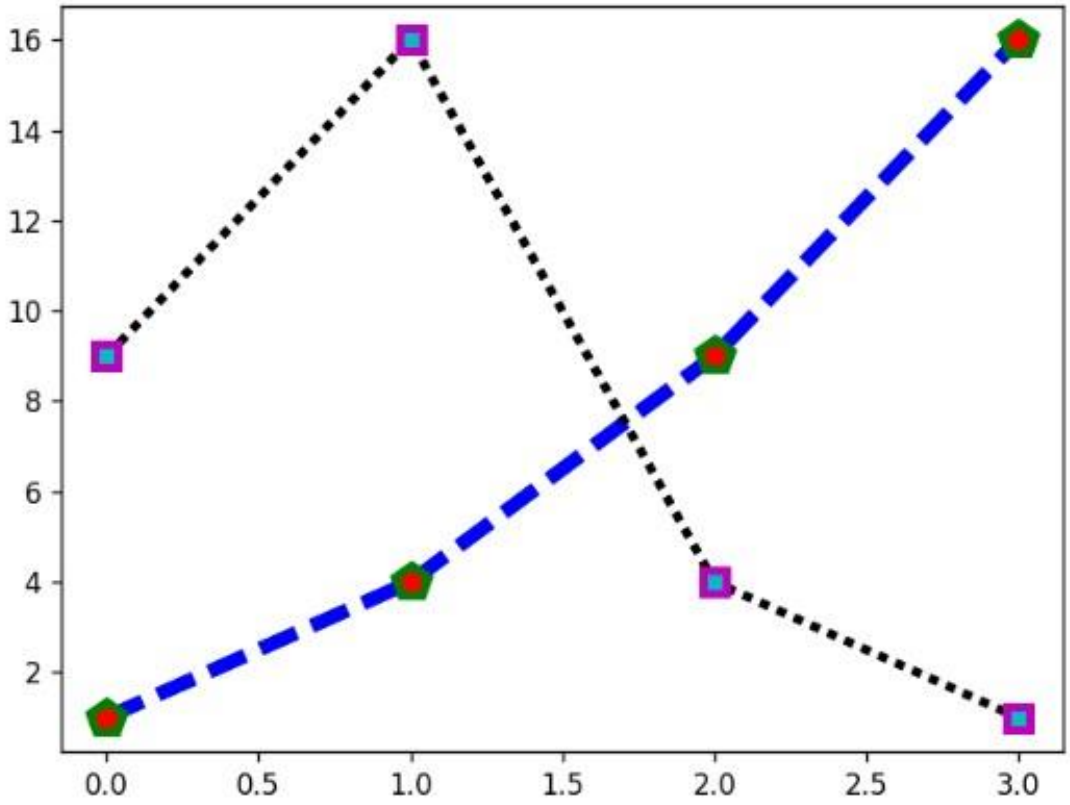

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\__main__.py:2: MatplotlibDeprecationWarning: pyp
lot.hold is deprecated.
    Future behavior will be consistent with the long-time default:
    plot commands add elements without first clearing the
    Axes and/or Figure.
  from ipykernel import kernelapp as app
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\__init__.py:917: UserWarning: axes.hold is depr
ecated. Please remove it from your matplotlibrc and/or style files.
  warnings.warn(self.msg_depr_set % key)
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\rcsetup.py:152: UserWarning: axes.hold is depre
cated, will be removed in 3.0
  warnings.warn("axes.hold is deprecated, will be removed in 3.0")
```

```
[<matplotlib.lines.Line2D at 0x287ed98c6a0>]
```

```
mpl.__version__
```

```
'2.0.0'
```

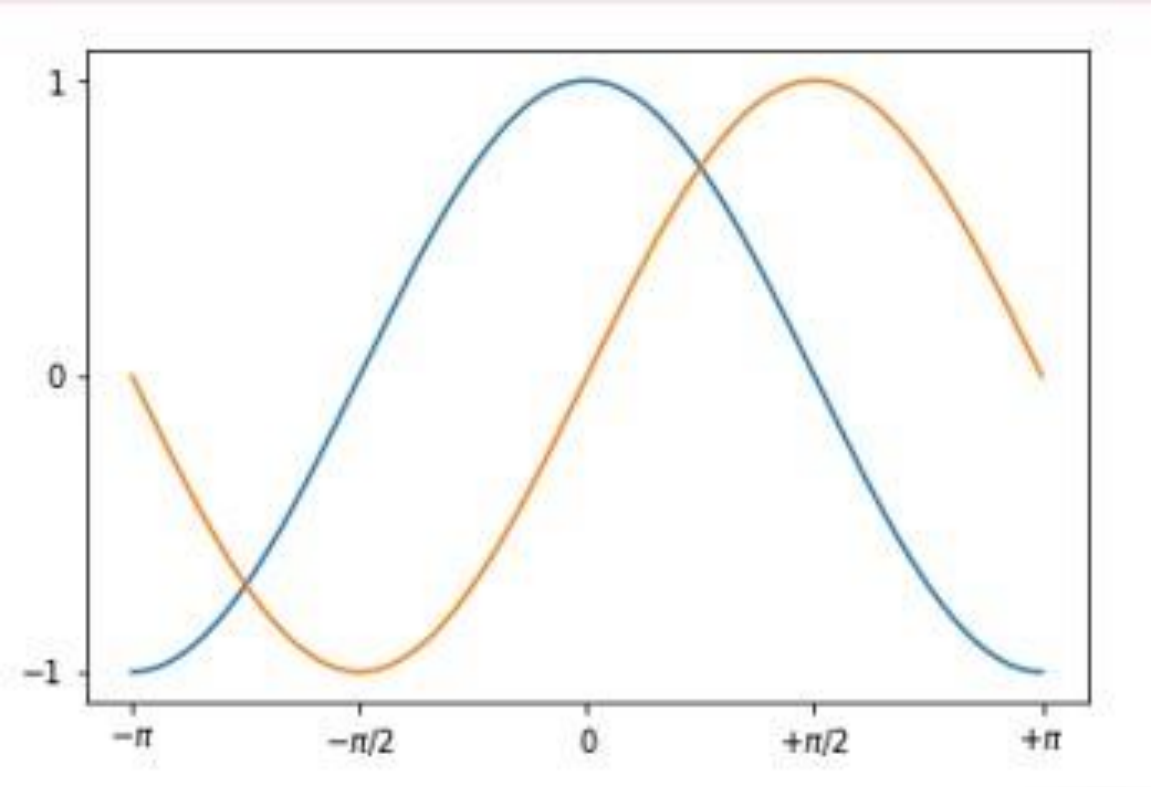
```
plt.plot([1,4,9,16], c='b', lw=5, ls='--', marker='p', ms=12, mec='g', mew=3, mfc='r')
plt.plot([9,16,4,1], c='k', lw=3, ls=':', marker='s', ms=8, mec='m', mew=3, mfc='c')
```




```
[<matplotlib.lines.Line2D at 0x287ee76f240>]
```

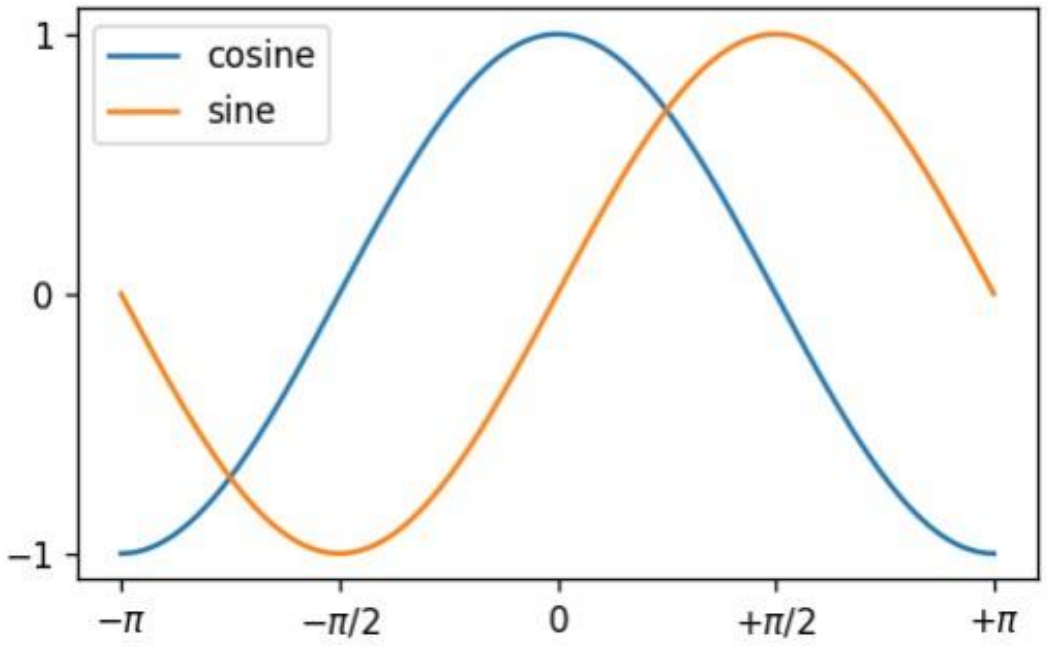
# 실습 3

- Sine함수와 cosine함수를 아래와 같이 겹쳐 그리시오.



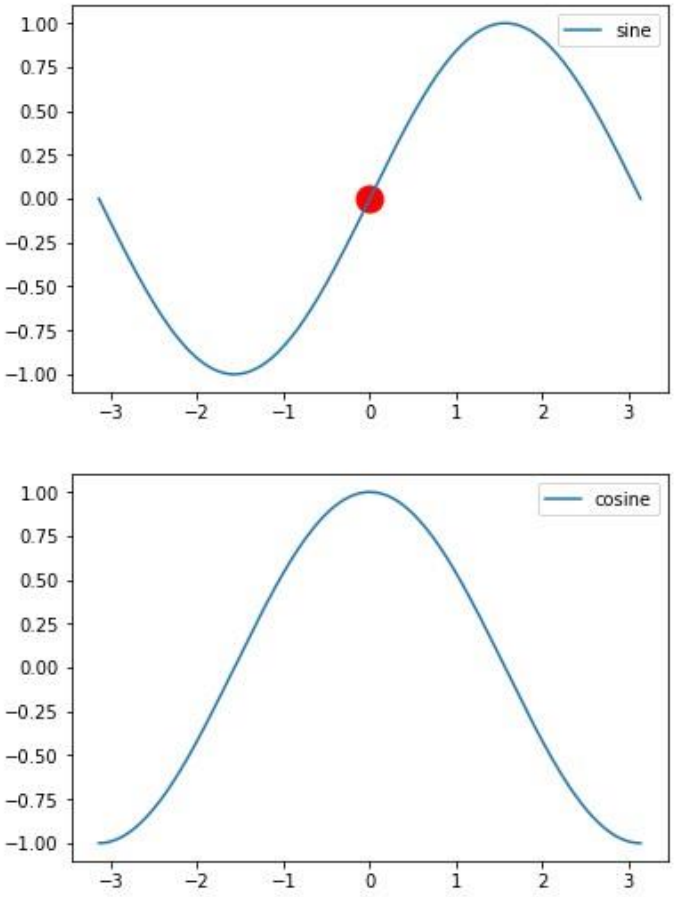
```
X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, label="cosine")
plt.hold(True)
plt.plot(X, S, label="sine")
#plt.legend(loc=0)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],\
[r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.yticks([-1, 0, 1])
plt.show()
```

```
f1=plt.figure(figsize=(5,3))
X=np.linspace(-np.pi, np.pi, 256)
C,S=np.cos(X), np.sin(X)
plt.plot(X,C, label="cosine")
plt.plot(X,S, label="sine")
plt.legend()
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],\
           [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$',r'$+\pi$'] )
plt.yticks([-1, 0, 1])
```



- Default가 hold(True)이고, 더 이상 지원되지 않으므로, hold(False)를 구현하려면…

```
:
import matplotlib.pylab as plt
import numpy as np
X=np.linspace(-np.pi,np.pi,256)
S=np.sin(X)
C=np.cos(X)
plt.plot(0,0,c='red',lw=5,marker='o',ms=15)
plt.plot(X,S,label='sine')
plt.legend(loc=0)
fig=plt.figure()
ax1=fig.add_subplot(1,1,1)
ax1.plot(X,C,label='cosine')
ax1.legend(loc=0)
plt.show()
```



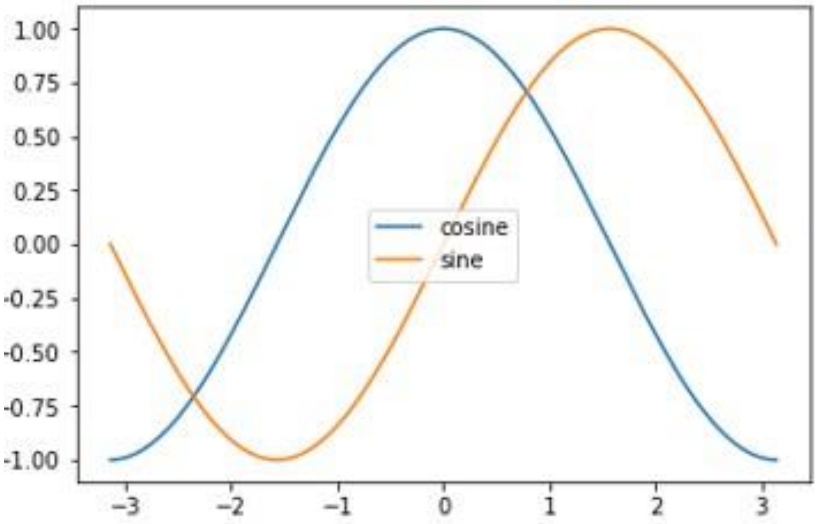
# 범례

- 범례의 위치는 자동으로 정해지지만 수동으로 설정하고 싶으면 loc 인수를 사용한다.
- 인수에는 문자열 혹은 숫자가 들어가며 가능한 코드는 다음과 같다.

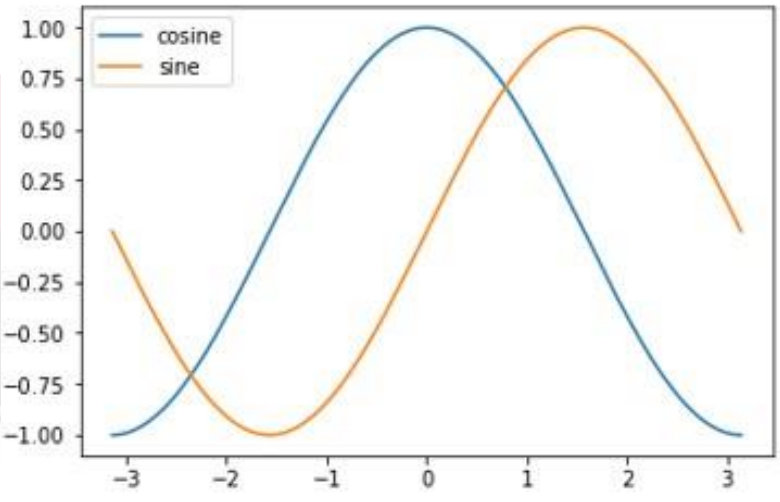
| loc 문자열 | 숫자 |
|---|---|
| best | 0 |
| upper right | 1 |
| upper left | 2 |
| lower left | 3 |
| lower right | 4 |
| right | 5 |
| center left | 6 |
| center right | 7 |
| lower center | 8 |
| upper center | 9 |
| center | 10 |

# Legend( ) 명령

```
X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, label="cosine")
plt.hold(True)
plt.plot(X, S, label="sine")
plt.legend(loc=10)
plt.show()
```



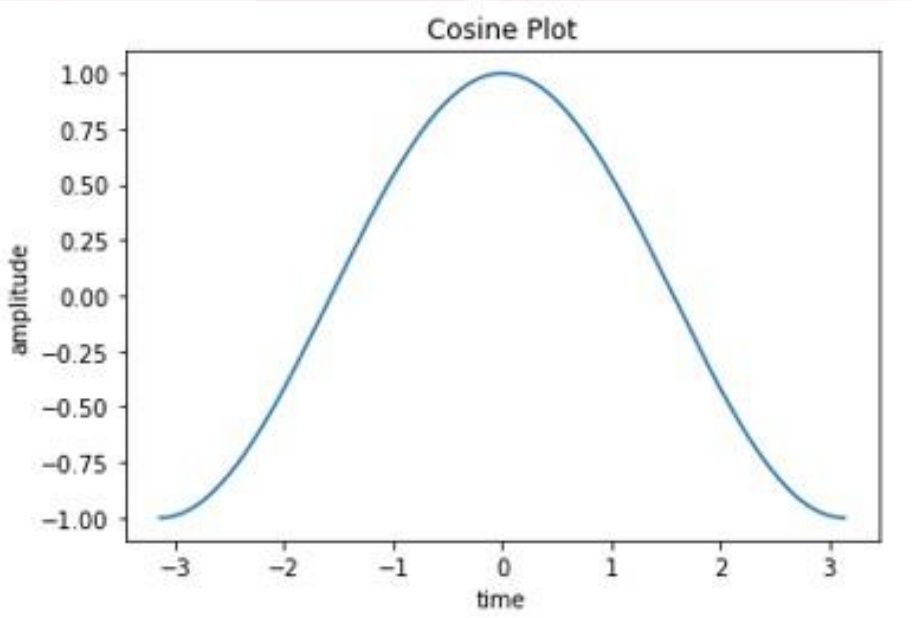

```
X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, label="cosine")
plt.hold(True)
plt.plot(X, S, label="sine")
plt.legend(loc=0)
plt.show()
```



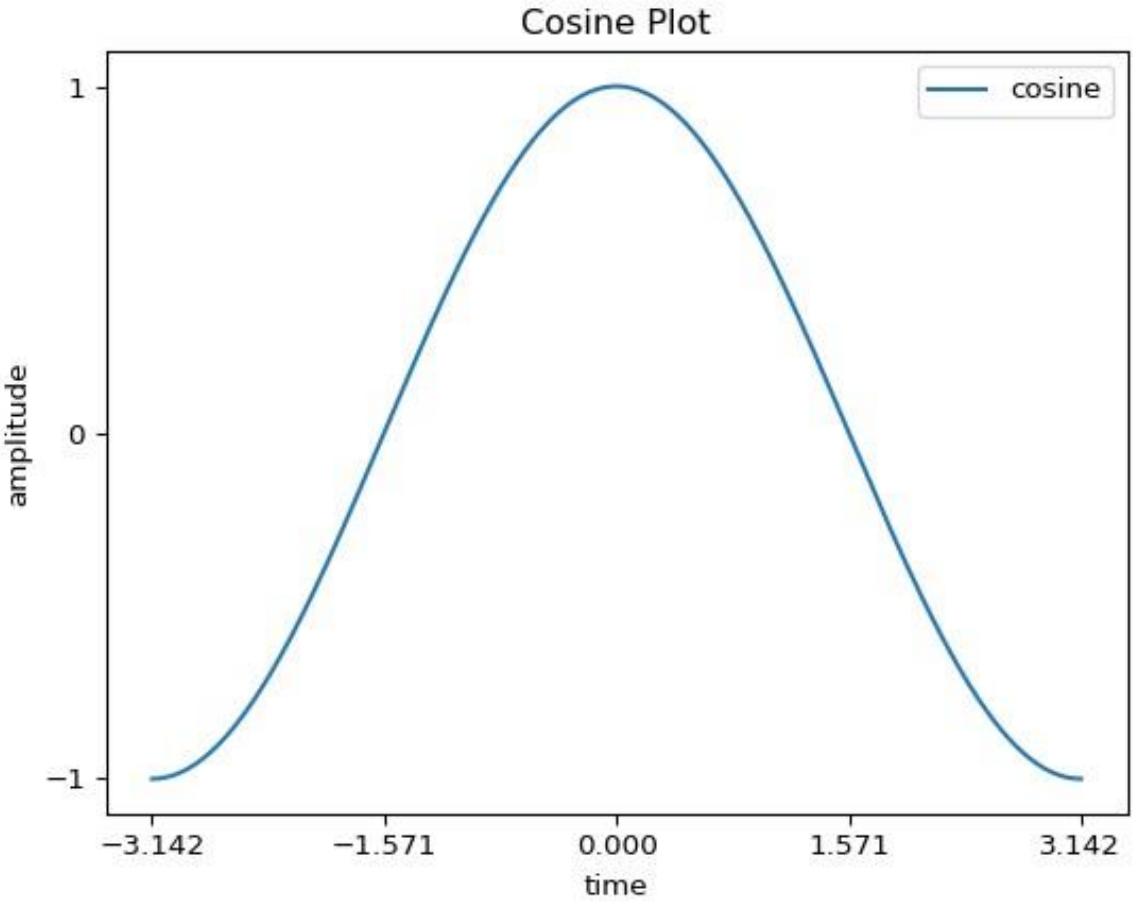
# x축, y축 라벨, 타이틀

- 라벨을 붙이려면 **xlabel. ylabel** 명령을 사용한다.
- 플롯의 위에는 **title** 명령으로 제목을 붙일 수 있다.

```
X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, label="cosine")
plt.xlabel("time")
plt.ylabel("amplitude")
plt.title("Cosine Plot")
plt.show()
```



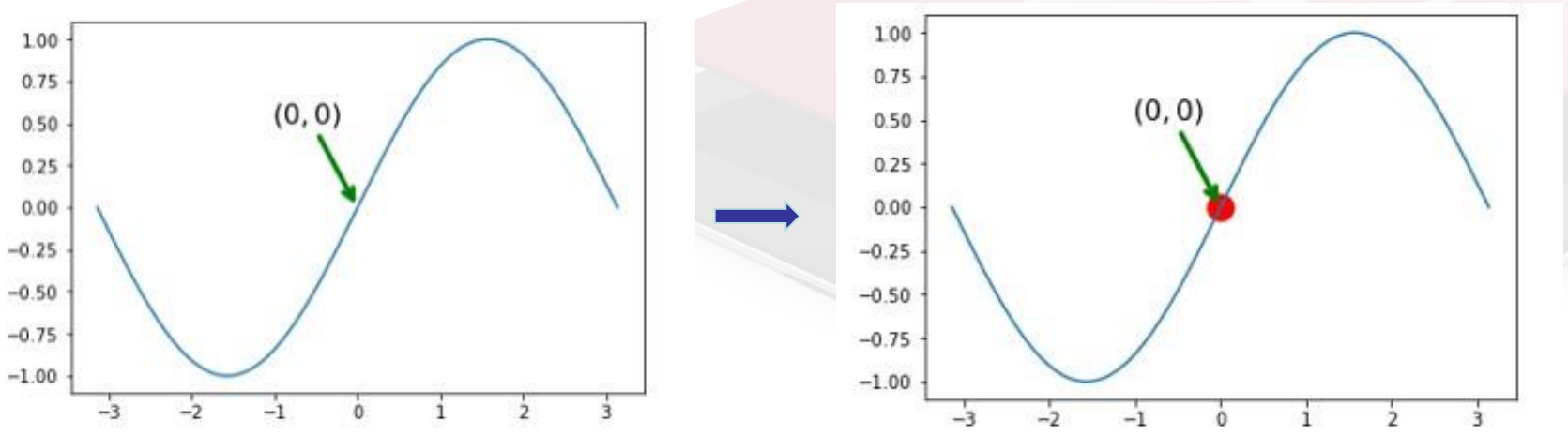
```
X = np.linspace(-np.pi, np.pi, 256)
C = np.cos(X)
plt.plot(X, C, label='cosine')
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.yticks([-1, 0, 1])
plt.legend(loc=0)
plt.xlabel('time')
plt.ylabel('amplitude')
plt.title('Cosine Plot')
```



# 부가설명

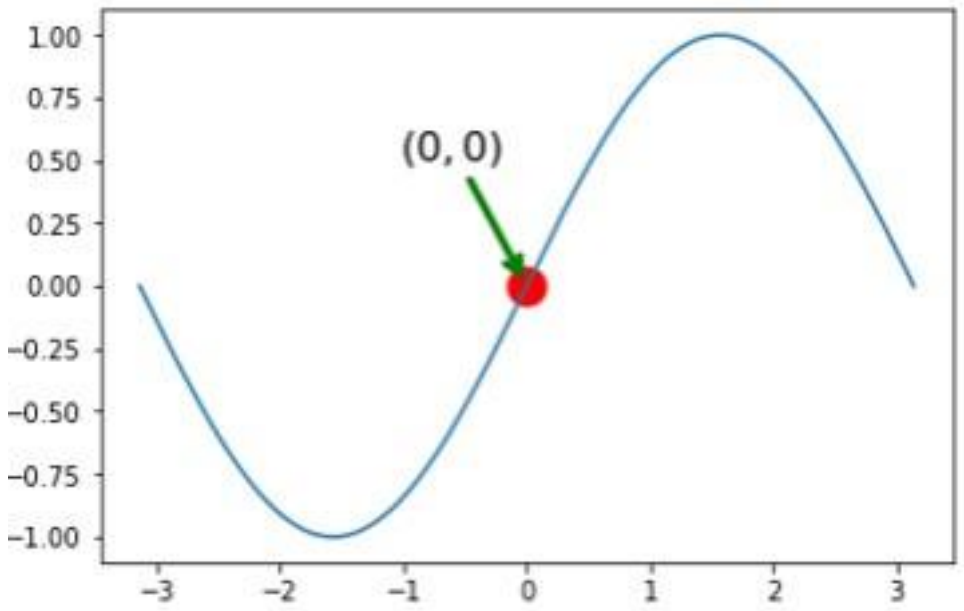
- annotate 명령을 사용하면 그림 내에 화살표를 포함한 부가 설명 문자열을 넣을 수 있다.

```
plt.plot(X, S, label="sine")
plt.annotate(r'$(0,0)$', xy=(0, 0), xytext=(-50, 50),xycoords='data',
             textcoords='offset points', fontsize=16,
            arrowprops=dict(arrowstyle="->", linewidth=3, color="g"))
plt.show()
```

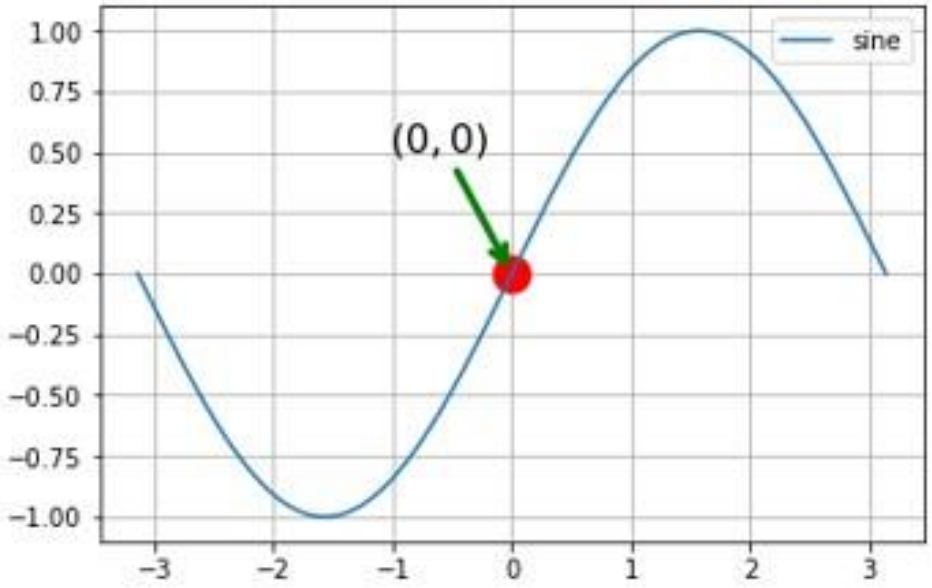


# Annotation

```
plt.plot(X, S, label="sine")
plt.scatter([0], [0], color="r", linewidth=10)
plt.annotate(r'$(0,0)$', xy=(0, 0), xycoords='data', xytext=(-50, 50),
             textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", linewidth=3, color="g"))
plt.show()
```



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

X=np.linspace(-np.pi,np.pi,256)
S=np.sin(X)

plt.plot(0,0,c='red',lw=5,marker='o',ms=15)
plt.plot(X,S,label='sine')
plt.legend(loc=0)
plt.grid(True)
plt.annotate(r'$(0,0)$', xy=(0,0), xytext=(-50,50),xycoords='data',
             textcoords='offset points',fontsize=16,\
           arrowprops=dict(arrowstyle='->',linewidth=3,color='g'))
plt.show()
```

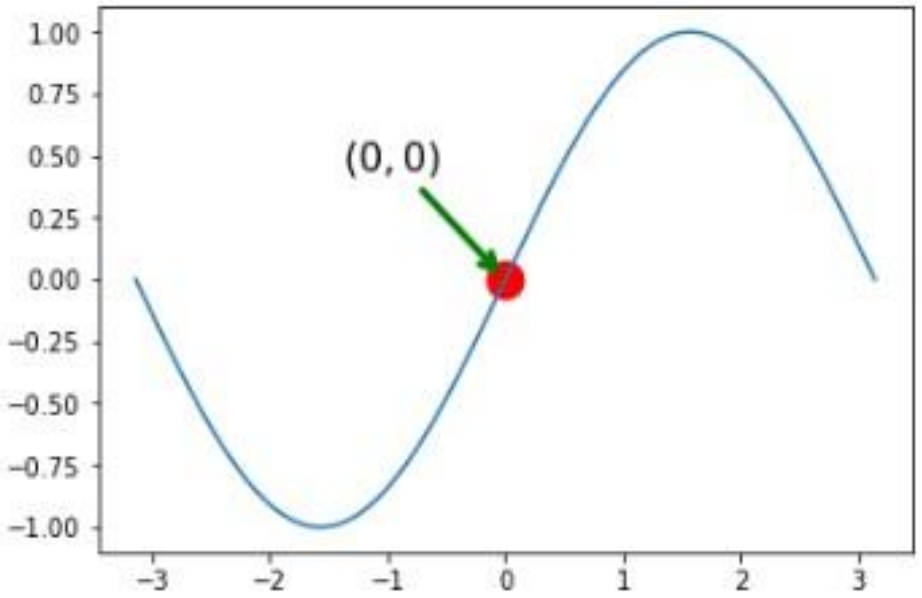


**a variety of other coordinate systems**

- you can specify the coordinate system of xy and xytext with one of the following strings for xycoords and textcoords
- default : 'data'

| argument | coordinate system |
|---|---|
| **'figure points'** | points from the lower left corner of the figure |
| **'figure pixels'** | pixels from the lower left corner of the figure |
| **'figure fraction'** | 0,0 is lower left of figure and 1,1 is upper right |
| **'axes points'** | points from lower left corner of axes |
| **'axes pixels'** | pixels from lower left corner of axes |
| **'axes fraction'** | 0,0 is lower left of axes and 1,1 is upper right |
| **'data'** | use the axes data coordinate system |

```
X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)
plt.plot(X, S, label="sine")
plt.scatter([0], [0], color="r", linewidth=10)
plt.annotate(r'$(0,0)$', xy=(0, 0), xycoords='data', xytext=(0.3, 0.7),
             textcoords='axes fraction', fontsize=16,
             arrowprops=dict(arrowstyle="->", linewidth=3, color="g"))
plt.show()
```



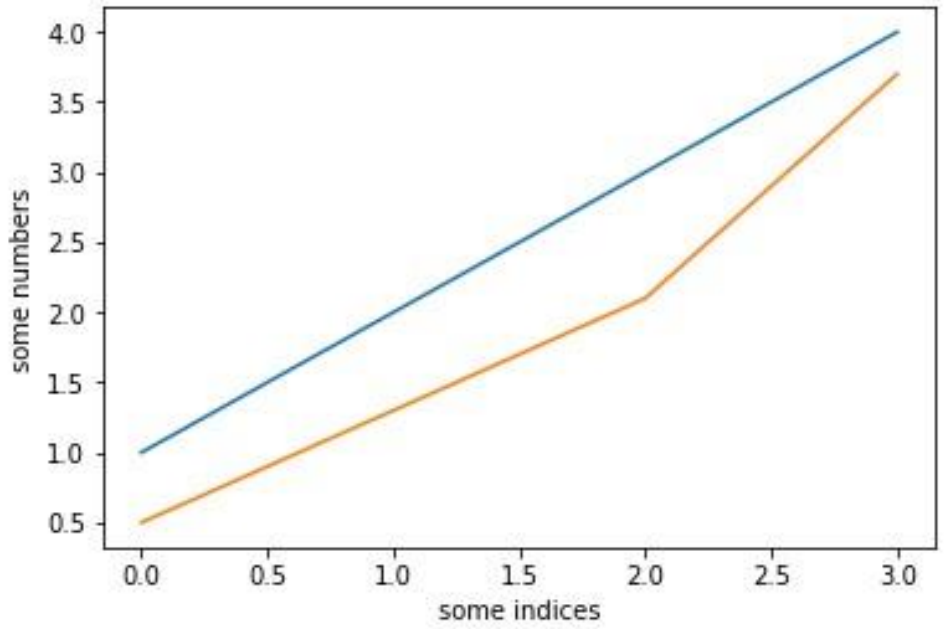

xy는 'data' 좌표계를 사용했고, xytext는 'axes fraction'좌표계를 사용

# plot 이해하기

plot 함수를 실행하면 하나의 list가 생기고 **Line2D object**이 생기며, **plot함수를 하나 더 실행해서 기존 list에 원소로 추가 가능**

```
import matplotlib.pyplot as plt
a = plt.plot([1,2,3,4])
b = plt.ylabel('some numbers')
c = plt.xlabel('some indices')
a +=(plt.plot([0.5, 1.3, 2.1, 3.7]))
print(a)
print(type(a[0]))
print(b)
print(c)
plt.show()
```

```
[<matplotlib.lines.Line2D object at 0x000001E69BD1C240>,
<matplotlib.lines.Line2D object at 0x000001E69BBB7E80>]
<class 'matplotlib.lines.Line2D'>
Text(0,0.5,'some numbers')
Text(0.5,0,'some indices')
```

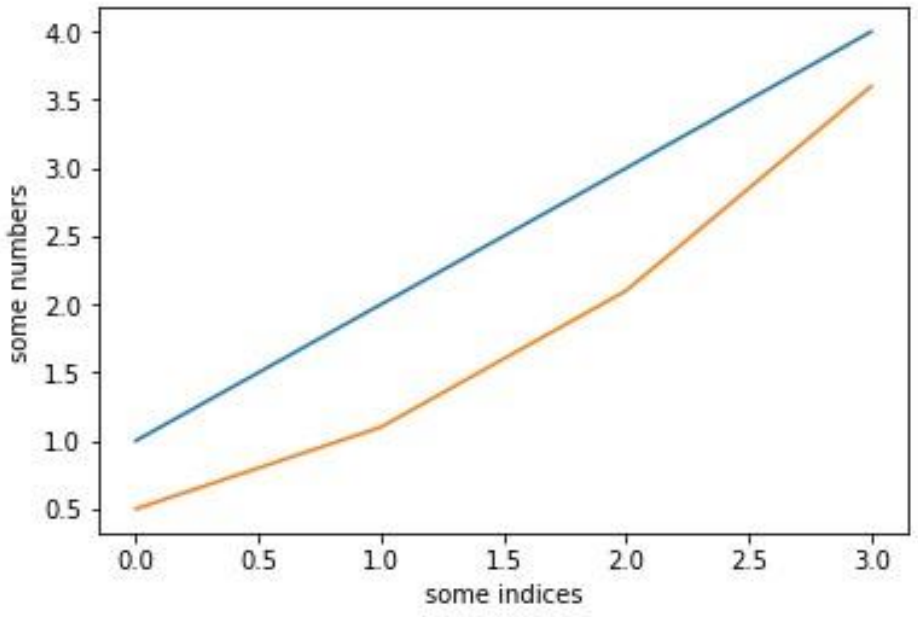


# plot 이해하기

2개의 plot 함수를 실행해서 출력해도 앞장의 경우와 동일하게 출력됨

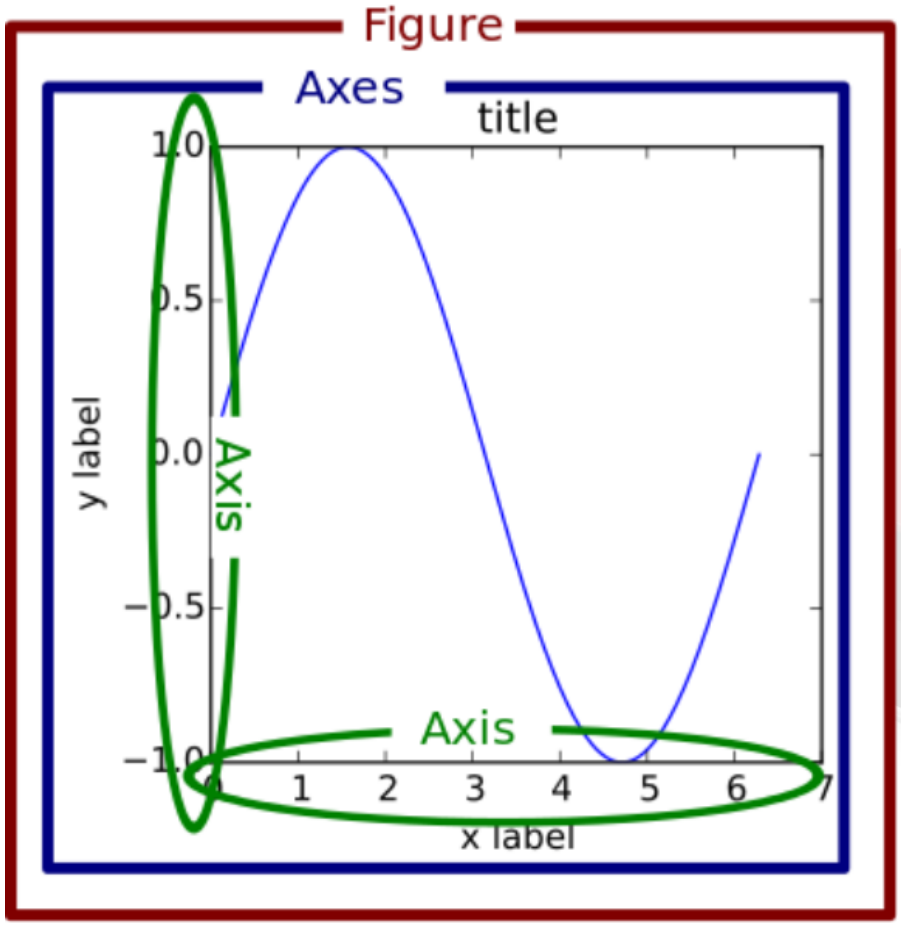
```
import matplotlib.pyplot as plt
a = plt.plot([1,2,3,4])
b = plt.ylabel('some numbers')
c = plt.xlabel('some indices')
d = plt.plot([0.5, 1.1, 2.1, 3.6])
print(a)
print(type(a[0]))
print(b)
print(c)
print(d)
plt.show()
```

```
[<matplotlib.lines.Line2D object at 0x000001E69C335C50>]
<class 'matplotlib.lines.Line2D'>
Text(0,0.5,'some numbers')
Text(0.5,0,'some indices')
[<matplotlib.lines.Line2D object at 0x000001E69BFC40F0>]
```



# 그림의 구조

- matplotlib가 그리는 그림은 Figure, Axes, Axis 등으로 이어지는 구조를 가진다.

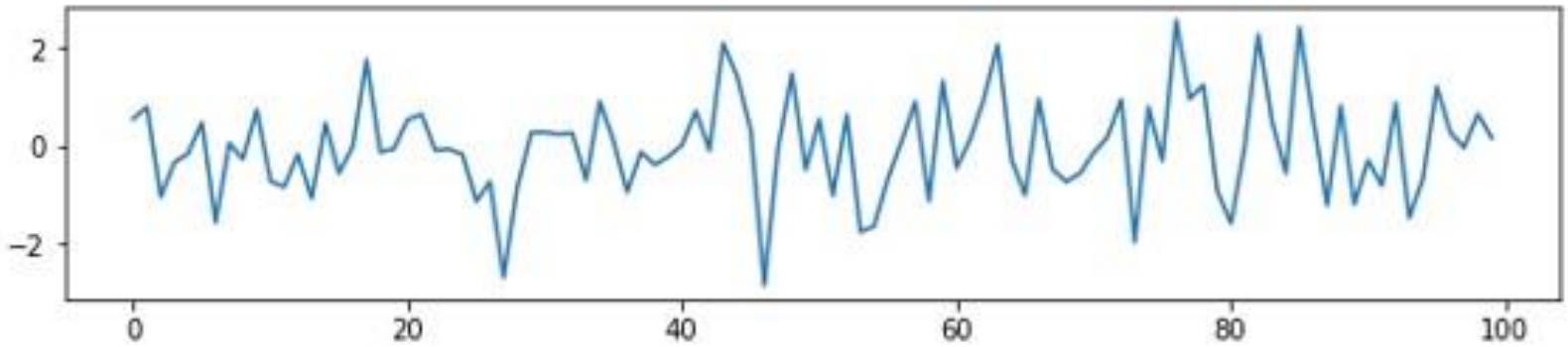


# Figure

- 모든 그림은 matplotlib.figure.**Figure** 클래스 객체에 포함되어 있다.
- 내부 플롯(inline plot)이 아닌 경우에는 하나의 Figure는 하나의 id 숫자와 윈도우(Window)를 가진다.
- https://matplotlib.org/stable/api/figure_api.html
- figure 명령을 사용하여 그 반환값으로 Figure 객체를 얻을 수 있으며, 일반적인 **plot 명령을 실행하면 자동으로 Figure를 생성해**준다.
- figure 명령을 명시적으로 사용하는 경우
  - 여러 개의 윈도우를 동시에 띄워야 하거나(line plot이 아닌 경우)
  - Jupyter 노트북 등에서(line plot의 경우) 그림의 크기를 설정하고 싶을 때이다. 그림의 크기는 figsize 인수로 설정한다.

```
f1 = plt.figure(figsize=(10,2))
plt.plot(np.random.randn(100))
plt.show()
```

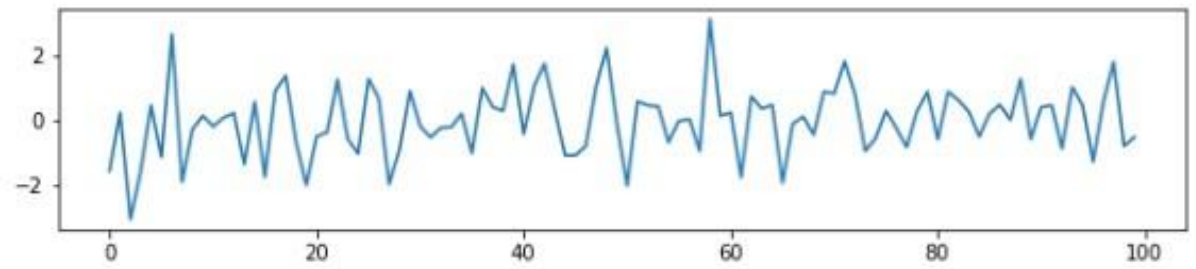
```
f1 = plt.figure(figsize=(10,2))
plt.plot(np.random.randn(100))
plt.show()
```



# 그림저장 savefig("파일이름")

```
f1 = plt.figure(figsize=(10,2))
plt.plot(np.random.randn(100))
print(f1)
plt.show()
```

```
Figure(720x144)
```

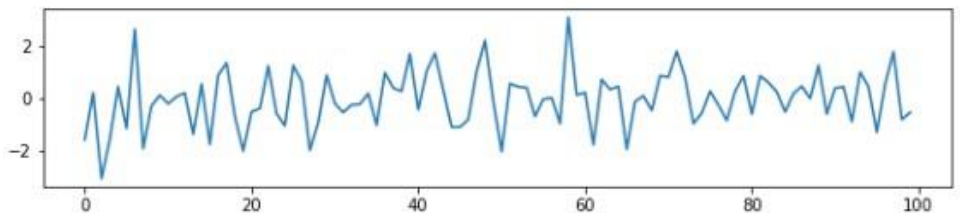



```
f1.savefig('my_figure.png') #savefig()이용해서 다양한 형식의 그림저장가능
```

```
%ls -lh my_figure.png
```

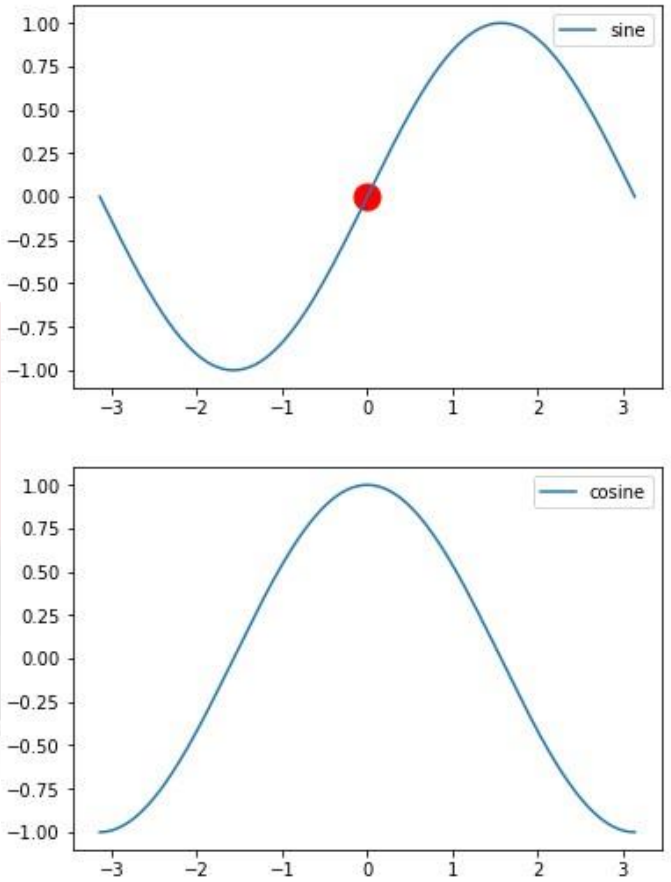
```
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: F807-6E28

 C:\Users\KyungHee 디렉터리


 C:\Users\KyungHee 디렉터리

2019-11-24  오후 05:23            21,223 my_figure.png
               1개 파일              21,223 바이트
               0개 디렉터리  96,850,538,496 바이트 남음
```

```
from IPython.display import Image
Image("my_figure.png")
```



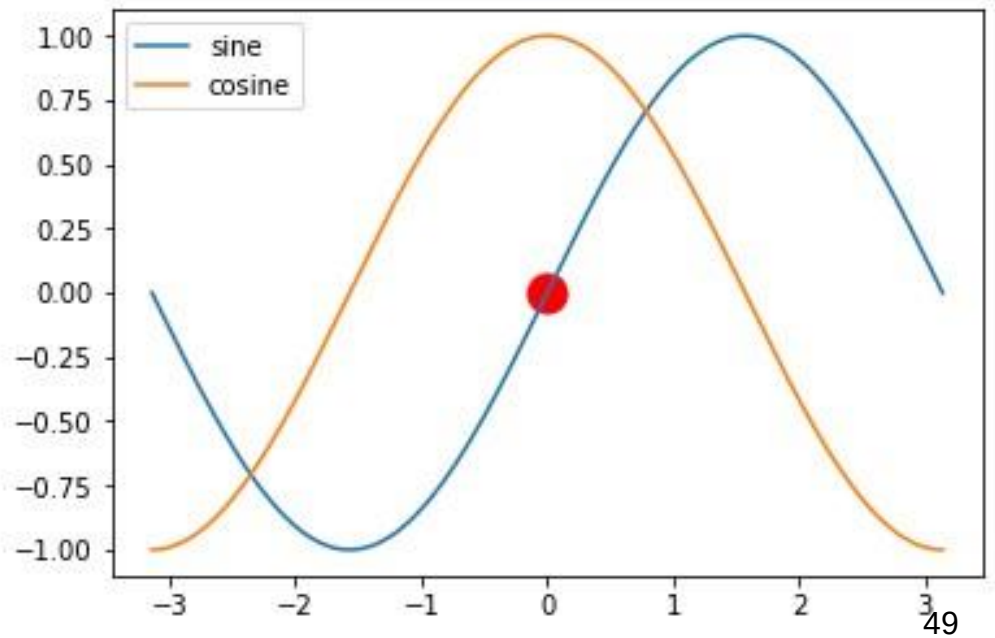
## Figure 객체를 두 개 만들어, Figure 클래스 별로 별도 캔버스구성

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

X=np.linspace(-np.pi,np.pi,256)
S=np.sin(X)
C=np.cos(X)

fig1=plt.figure()
plt.plot(0,0,c='red',lw=5,marker='o',ms=15)
plt.plot(X,S,label='sine')
plt.legend(loc=0)

fig2=plt.figure()
plt.plot(X,C,label='cosine')
plt.legend(loc=0)
plt.show()
```



# 비교해보세요...

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

X=np.linspace(-np.pi,np.pi,256)
S=np.sin(X)
C=np.cos(X)
plt.plot(0,0,c='red',lw=5,marker='o',ms=15)
plt.plot(X,S,label='sine')
plt.legend(loc=0)

plt.plot(X,C,label='cosine')
plt.legend(loc=0)
plt.show()
```

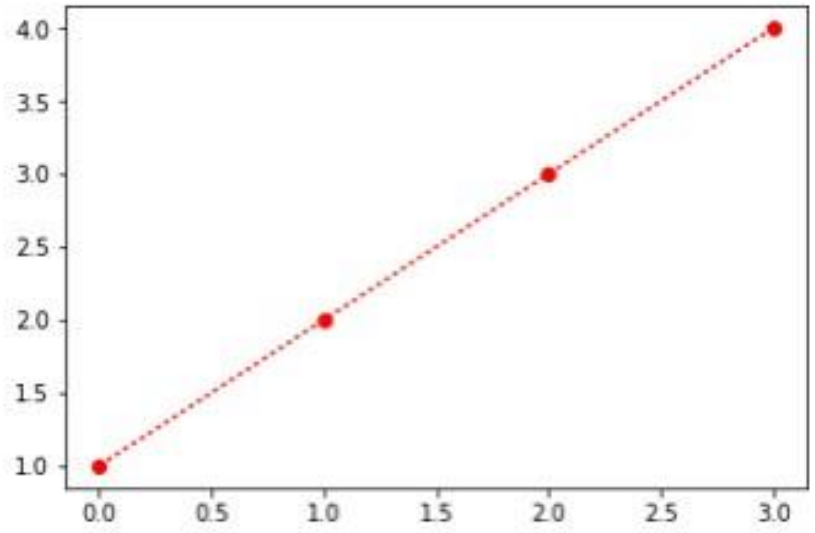


# gcf() returns the Current figure handle

- 명시적으로 figure 명령을 사용하지 않은 경우에 Figure 객체를 얻으려면 gcf 명령을 사용한다.

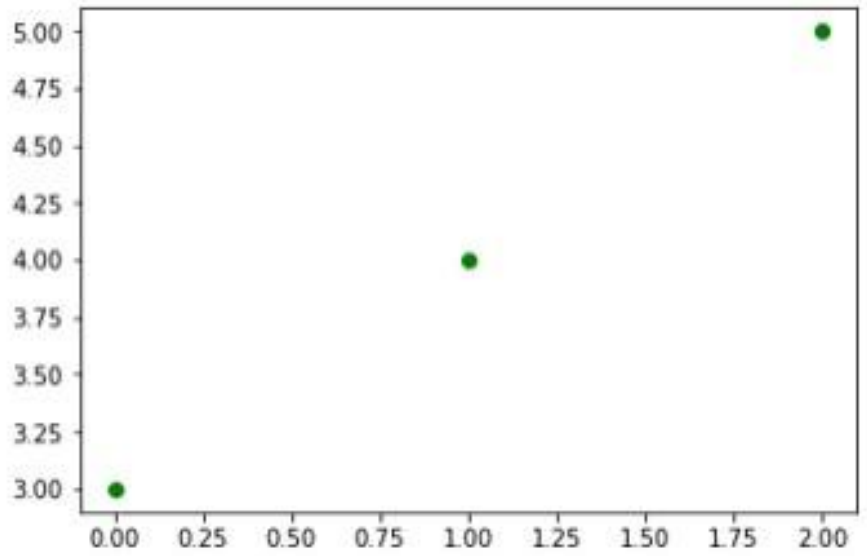
```
f1 = plt.figure()
plt.plot([1,2,3,4], 'ro:')
f2 = plt.gcf()
print(f1, id(f1))
print(f2, id(f2))
plt.show()
```

```
Figure(432x288) 2472451216888
Figure(432x288) 2472451216888

<matplotlib.figure.Figure at 0x23fa9edd828>
```




```
plt.plot([3,4,5], 'go')
f3 = plt.gcf()
print(f3, id(f3))
plt.show()
```

```
Figure(432x288) 2472463834360
```



# Axes와 subplot

- 하나의 윈도우(Figure)안에 여러 개의 플롯을 배열 형태로 보여야 하는 경우, Figure 안에 있는 각각의 플롯은 Axes 라고 불리는 객체에 속한다.

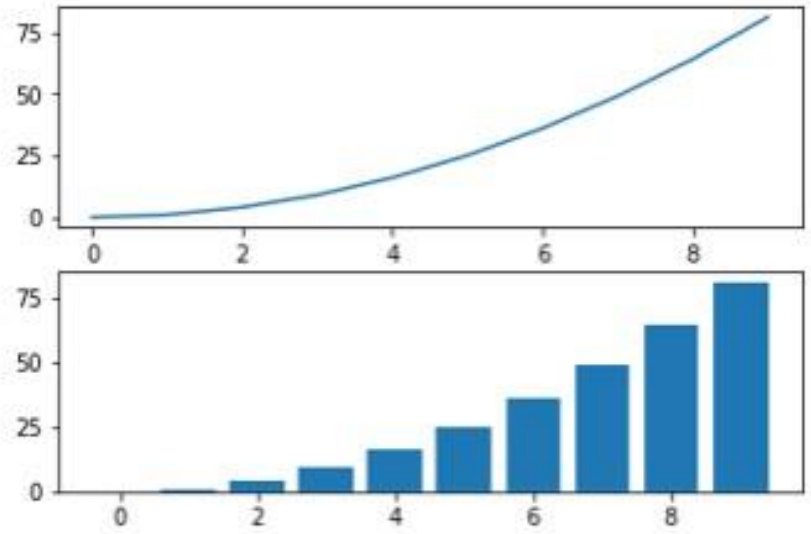
- Axes 객체에 대한 자세한 설명은 다음 웹사이트를 참조한다.
- https://matplotlib.org/stable/api/axes_api.html

- Figure 객체에 add_**subplot** 명령을 사용해서 Axes 객체를 얻거나, subplot 명령을 바로 사용해도 자동으로 Axes를 생성해 준다.

- subplot 명령은 그리드(grid) 형태의 Axes 객체들을 생성하는데 **Figure가 행렬(matrix)이고 Axes가 행렬의 원소**라고 생각하면 된다. 예를 들어 위와 아래 두 개의 플롯이 있는 경우 행이 2 이고 열이 1인 2x1 행렬이다.

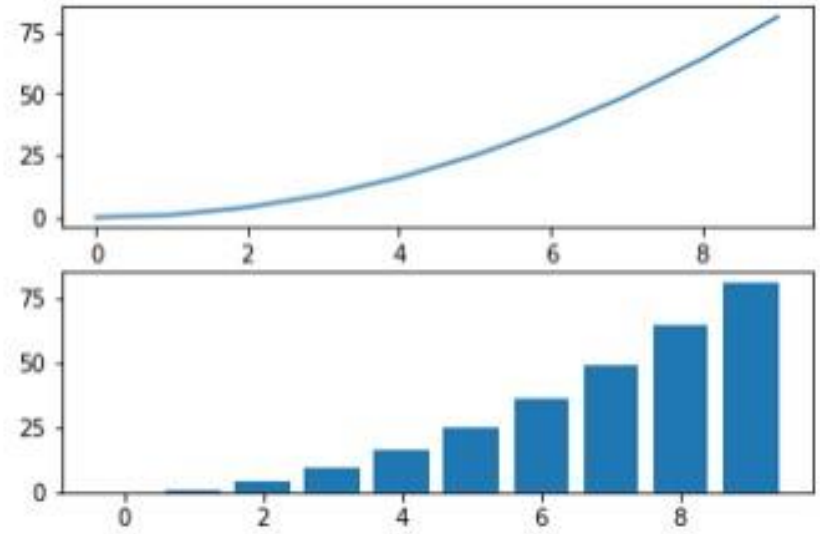
- subplot 명령은 세 개의 인수를 가지는데 **처음 두 개의 원소가 전체 그리드 행렬의 모양을 지시하는** 두 숫자이고 **세 번째 인수가 그 중 어느 것인지를** 의미하는 숫자이다.

```
fig = plt.figure()
ax1 = fig.add_subplot(2,1,1)
ax2 = fig.add_subplot(2,1,2)
x = range(0,10)
y = [v**2 for v in x]
ax1.plot(x,y)
ax2.bar(x,y)
plt.show()
```



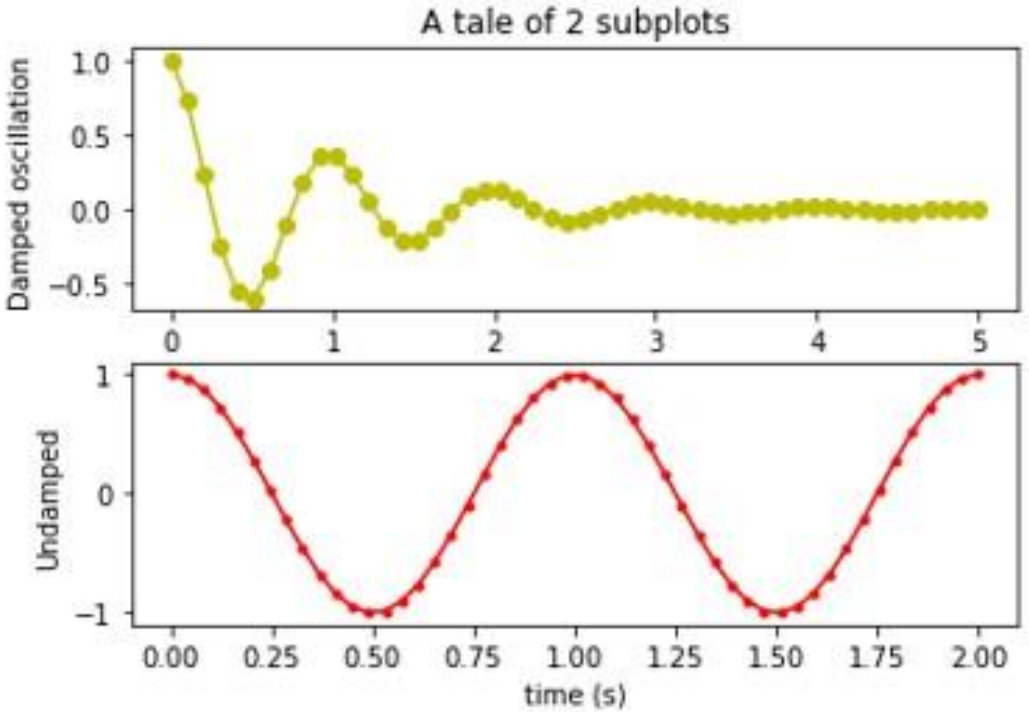

```
x = range(0,10)
y = [v**2 for v in x]
plt.subplot(2,1,1)
plt.plot(x,y)
ax1 = plt.gca()
plt.subplot(2,1,2)
plt.bar(x,y)
ax2 = plt.gca()
print(ax1, id(ax1))
print(ax2, id(ax2))
plt.show()
```

```
Axes(0.125,0.536818;0.775x0.343182) 2472461187000
Axes(0.125,0.125;0.775x0.343182) 2472463252952
```



```
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)

ax1 = plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'yo-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')
print(ax1)

ax2 = plt.subplot(2, 1, 2)
plt.plot(x2, y2, 'r.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
print(ax2)

plt.show()
```

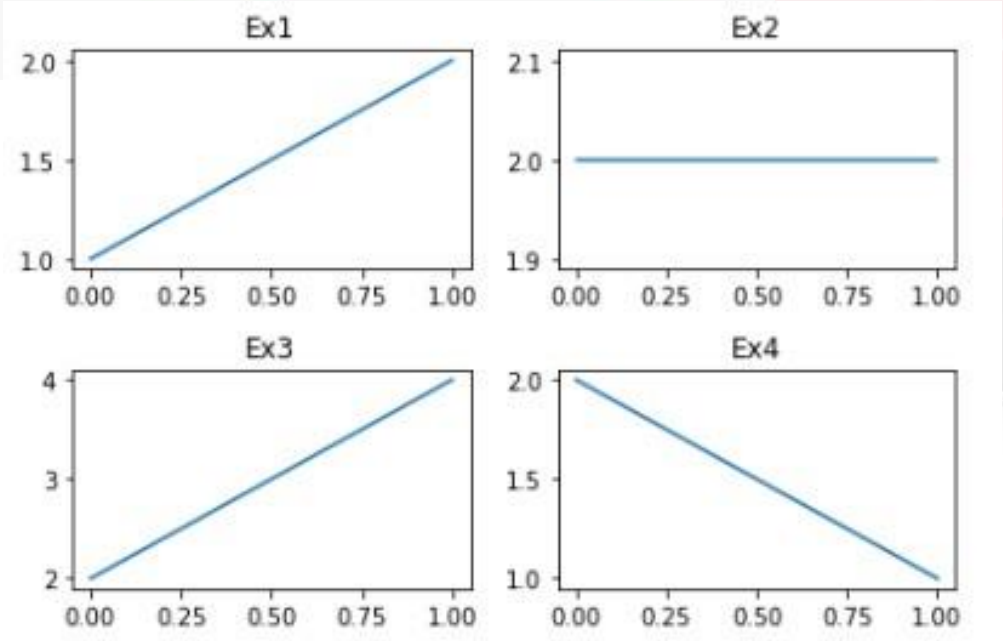
```
Axes(0.125,0.536818;0.775x0.343182)
Axes(0.125,0.125;0.775x0.343182)
```



# 2x2 형태의 네 개의 플롯이라면

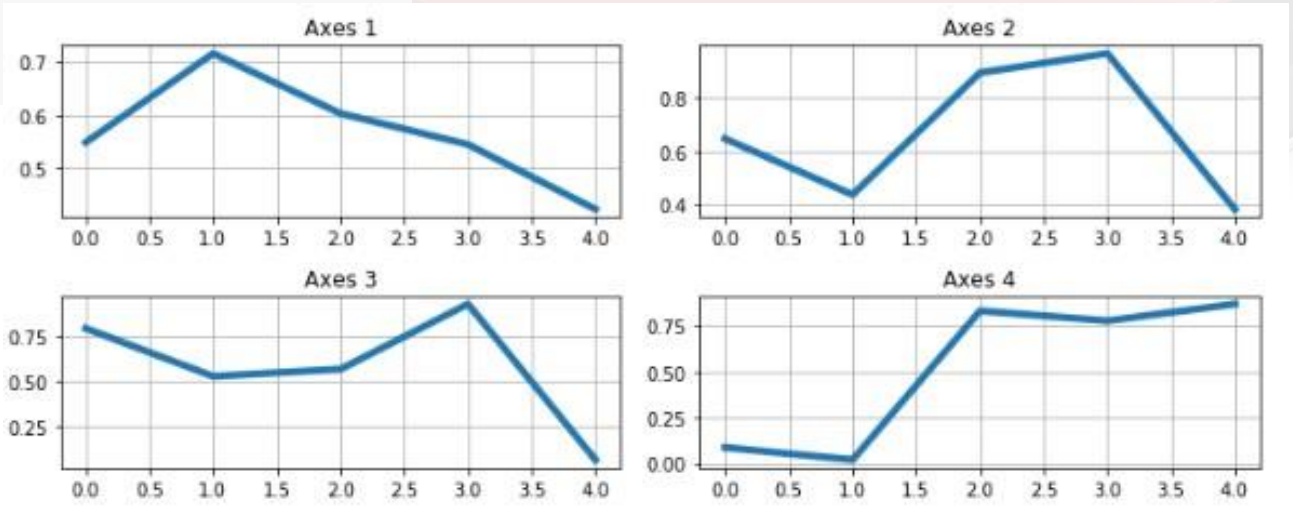
- subplot 의 인수는 (2,2,1)를 줄여서 221 라는 하나의 숫자로 표시할 수도 있다.
- Axes의 위치는 위에서 부터 아래로, 왼쪽에서 오른쪽으로 카운트한다.

```
plt.subplot(221); plt.plot([1, 2]); plt.title("Ex1")
plt.subplot(222); plt.plot([2, 2]); plt.title("Ex2")
plt.subplot(223); plt.plot([2, 4]); plt.title("Ex3")
plt.subplot(224); plt.plot([2, 1]); plt.title("Ex4")
plt.tight_layout()
plt.show()
```



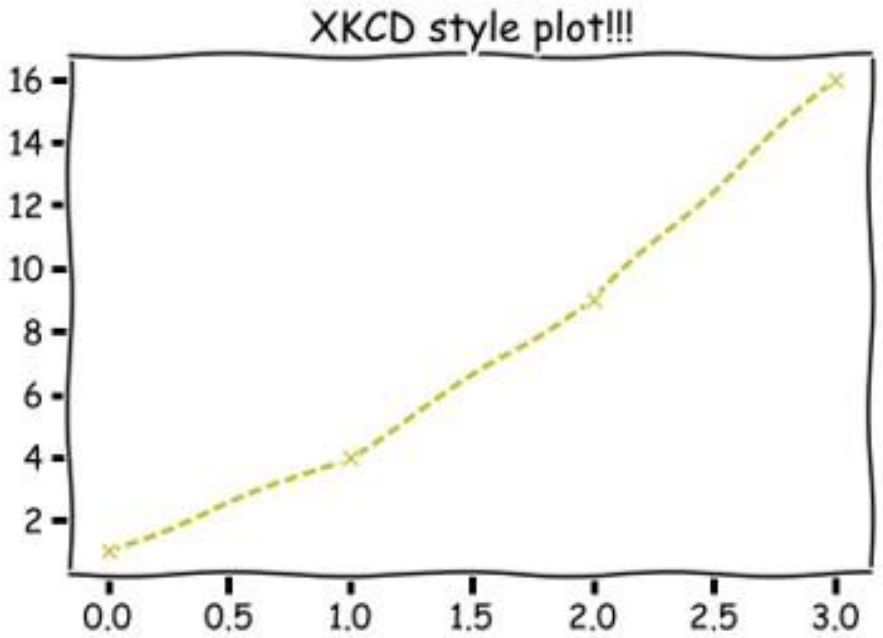
# subplots: 복수개의 Axes를 생성

```
fig, axes = plt.subplots(2,2) # 복수개

np.random.seed(0)
axes[0,0].plot(np.random.rand(5))
axes[0,0].set_title("Axes 1")
axes[0,1].plot(np.random.rand(5))
axes[0,1].set_title("Axes 2")
axes[1,0].plot(np.random.rand(5))
axes[1,0].set_title("Axes 3")
axes[1,1].plot(np.random.rand(5))
axes[1,1].set_title("Axes 4")

plt.tight_layout()
```



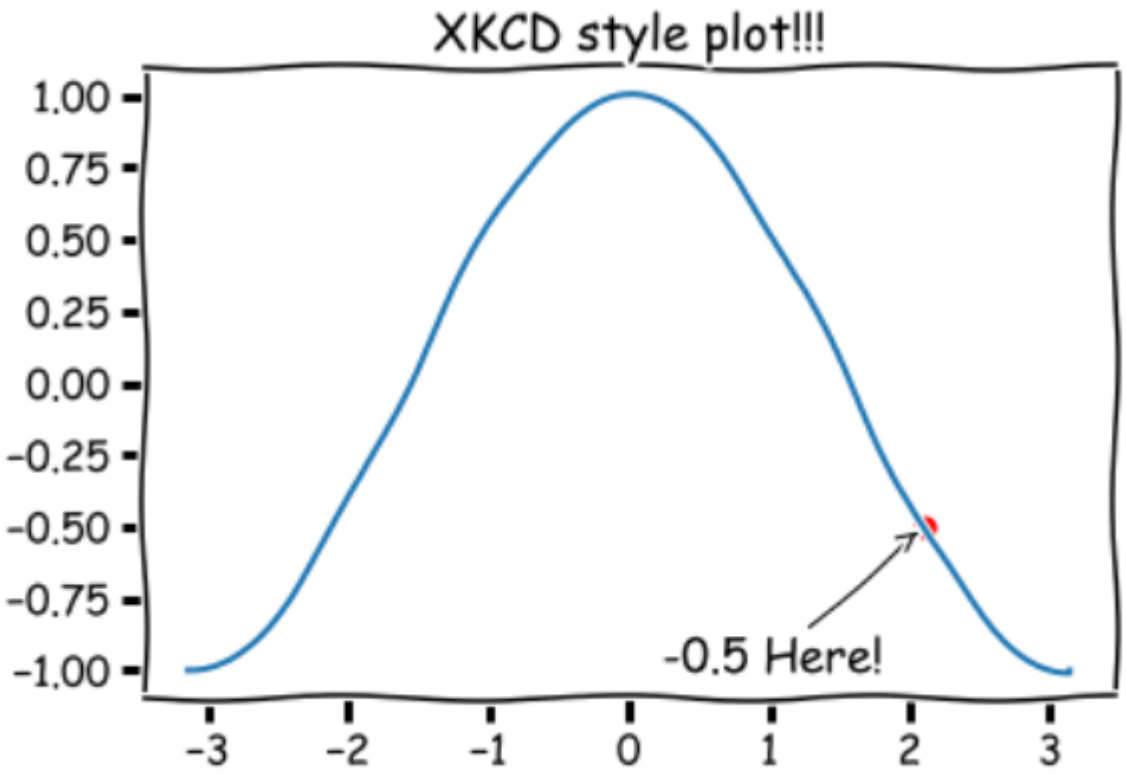
# xkcd 스타일

```
with plt.xkcd():
    plt.title('XKCD style plot!!!')
    plt.plot([1,4,9,16], 'yx--')
plt.show()
```



# 실습 4

아래와 같이 $x = 2\pi/3$ 위치에 $annotation$을 넣은 $cosine$ 함수를 그리시오



# 과제

아래와 같은 그래프를 그리시오.


"Stove Ownership" from xkcd by Randall Monroe