# 데이터 시각화 과제

**20171621 민대인**

## 시각화 패키지 Matplotlib

Matplotlib는 파이썬에서 데이터를 chart나 plot으로 visualization하는 패키지이다. 또한, 차트나 플롯 이외에도 저수준 api를 사용한 다양한 시각화 기능을 제공한다.

| Matplotlib |
| --- |
| 라인 플롯 (line plot) |
| 스캐터 플롯 (scatter plot) |
| 컨투어 플롯 (contour plot) |
| 서피스 플롯 (surface plot) |
| 바 차트 (bar chart) |
| 히스토그램 (histogram) |
| 박스 플롯 (box plot) |

## pylab 서브패키지

pylab은 Matplotlib의 서브패키지로서, 수치해석 소프트웨어의 시각화 명령을 그대로 사용할 수 있도록 Matplotlib의 하위 API를 포장한 명령어 집합을 제공한다.

**In [1]:**

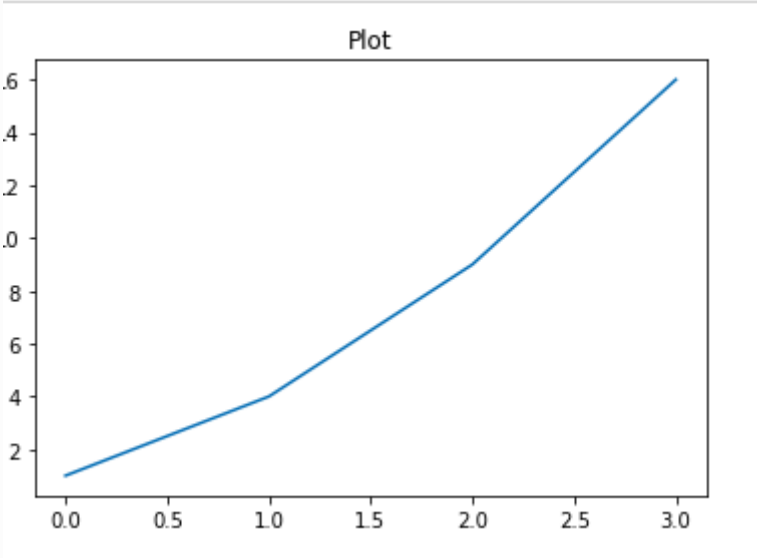
```
import matplotlib as mpl
import matplotlib.pylab as plt
```

## 라인 플롯

데이터가 시간, 순서 등에 따라 변화하는 선을 그림.

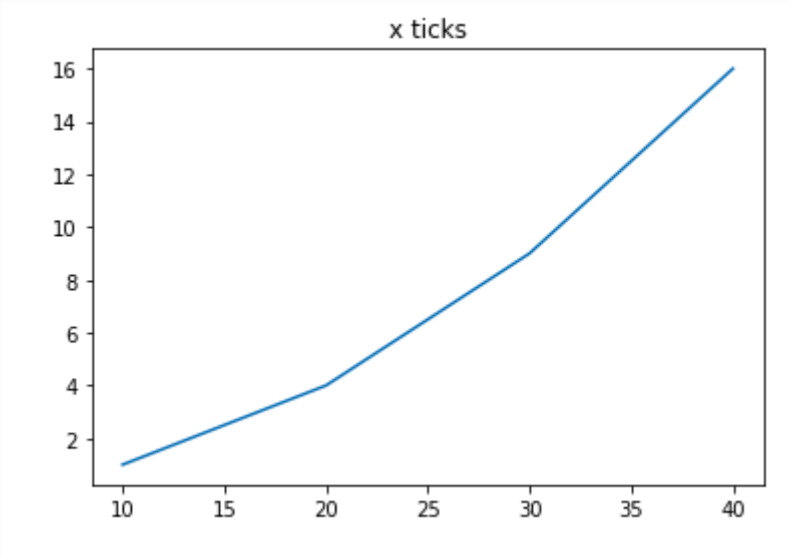
**In [2]:**

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




**In [3]:**

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




## 한글폰트 사용

Matplotlib에서 한글을 사용하면 한글이 깨져서 나온다. 그래서 다음과 같이 설정을 해주어야 한다.

**리눅스**

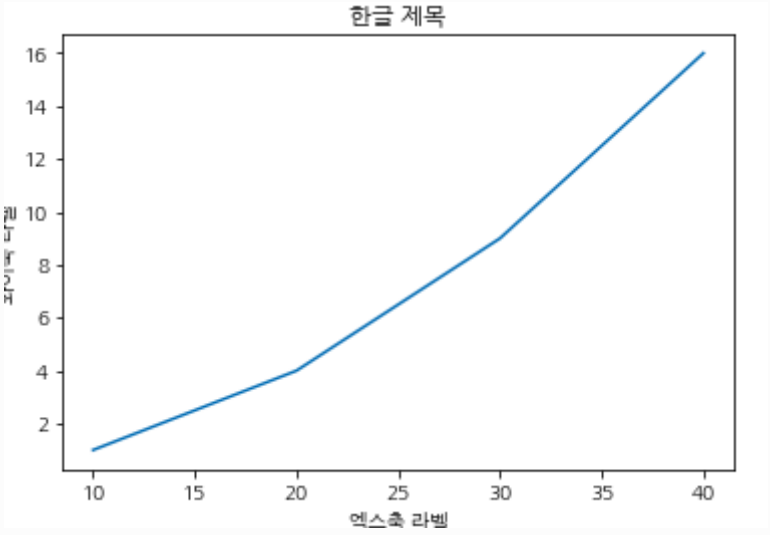
```
$ sudo apt-get install -y fonts-nanum*
$ sudo fc-cache -fv
$ rm ~/.cache/matplotlib -rf
```

한글이 설정 되었으면 예제를 실행해보자.

**In [4], In [5]:**

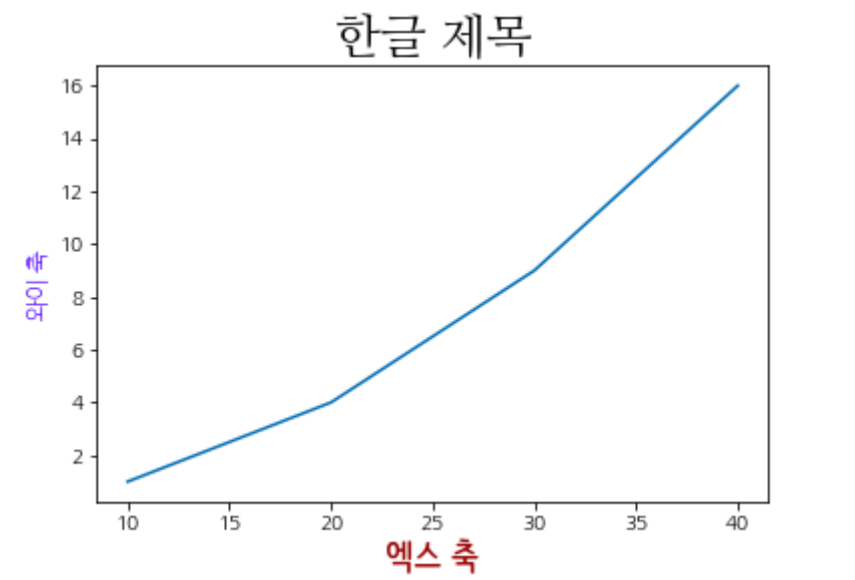
```
# 폰트 설정
mpl.rc('font', family='NanumGothic')
# 유니코드에서  음수 부호설정
mpl.rc('axes', unicode_minus=False)
```

```
plt.title('한글 제목')
plt.plot([10, 20, 30, 40], [1, 4, 9, 16])
plt.xlabel("엑스축 라벨")
plt.ylabel("와이축 라벨")
plt.show()
```




**In [6]:** 별도의 폰트 설정

```
font1 = {'family': 'NanumMyeongjo', 'size': 24,
         'color':  'black'}
font2 = {'family': 'NanumBarunpen', 'size': 18, 'weight': 'bold',
         'color':  'darkred'}
font3 = {'family': 'NanumBarunGothic', 'size': 12, 'weight': 'light',
         'color':  'blue'}

plt.plot([10, 20, 30, 40], [1, 4, 9, 16])
plt.title('한글 제목', fontdict=font1)
plt.xlabel('엑스 축', fontdict=font2)
plt.ylabel('와이 축', fontdict=font3)
plt.show()
```

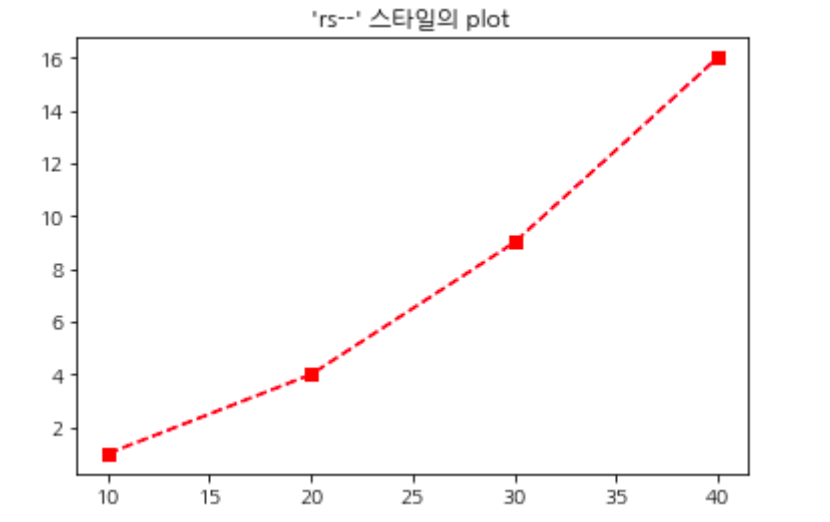



## 스타일 지정

보는 사람이 그림을 더 잘 알아보기 위해서, 색깔, 마커, 선 스타일을 지정하는 스타일 지정을 할 수 있다. 스타일 문자열은 색깔, 마커, 선 종류의 순서로 지정된다.

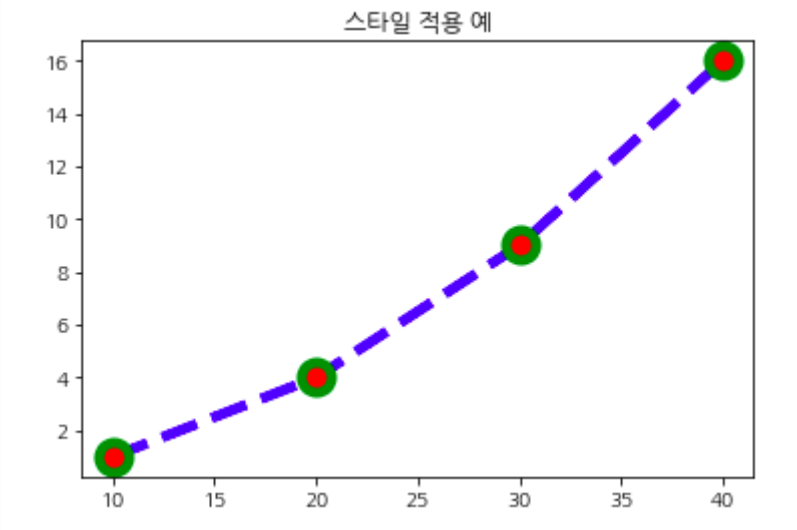
**In [7]:** 색: 빨강, 마커 : 네모, 선 종류 : 점선

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




**In [8]:**

```
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.title("스타일 적용 예")
plt.show()
```

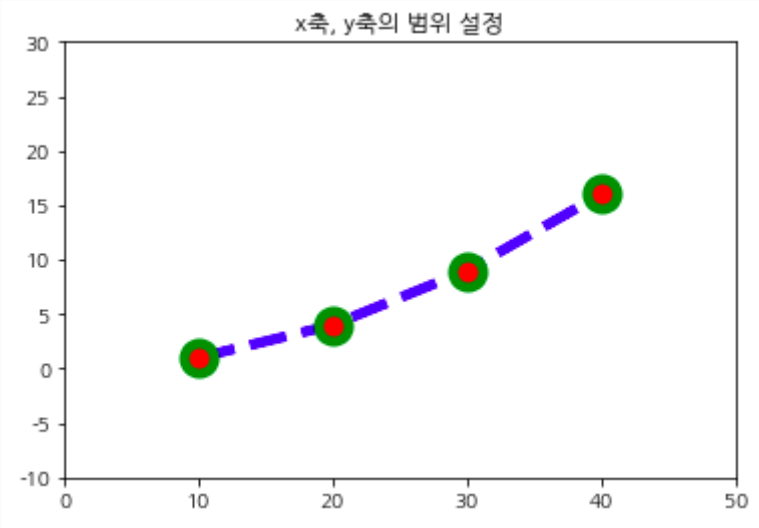



## 그림 범위 지정

몇몇 점들은 그림의 범위 경계선에 있어서 잘 보이지 않는 경우가 있을 수 있어, 수동으로 범위를 지정해 사용자들이 좀 더 편하게 그림을 볼 수 있게 한다.

**In [9]:**

```
plt.title("x축, y축의 범위 설정")
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.xlim(0, 50)
plt.ylim(-10, 30)
plt.show()
```

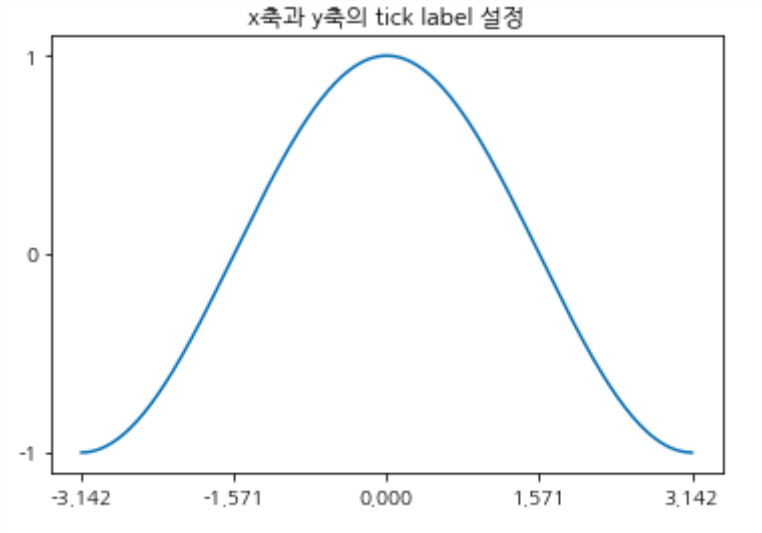



## 틱 설정

플롯이나 차트에서 축상의 위치 표시지점을 tick이라 하고, 이 틱에 써진 숫자 혹은 글자를 tick label이라고 한다.

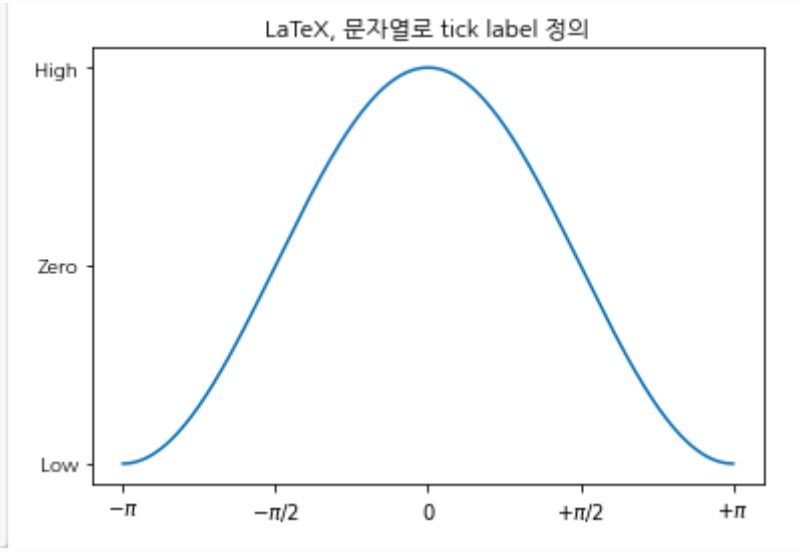
**In [10]:**

```
import numpy as np
X = np.linspace(-np.pi, np.pi, 256)
C = np.cos(X)
plt.title("x축과 y축의 tick label 설정")
plt.plot(X, C)
plt.xticks([-np.pi, -np.pi / 2, 0, np.pi / 2, np.pi])
plt.yticks([-1, 0, +1])
plt.show()
```




**In [11]:**

```
X = np.linspace(-np.pi, np.pi, 256)
C = np.cos(X)
plt.title("LaTeX, 문자열로 tick label 정의")
plt.plot(X, C)
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], ["Low", "Zero", "High"])
plt.show()
```

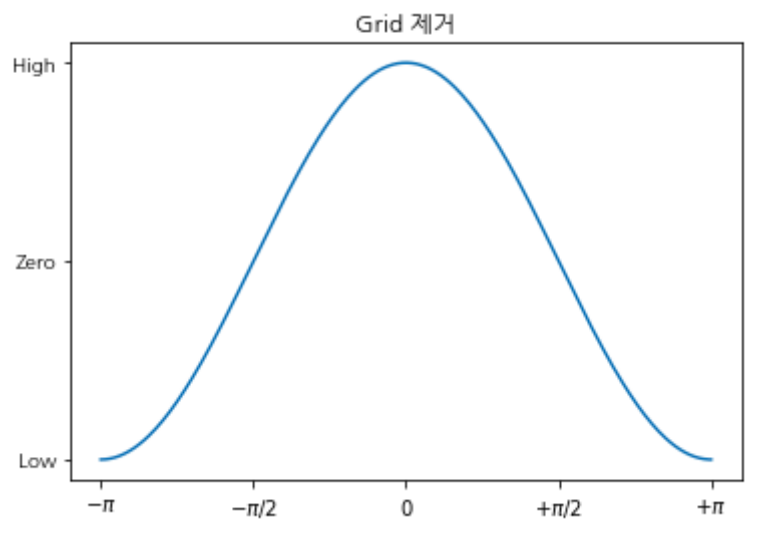



## 그리드 설정

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

**In [12]:**

```
X = np.linspace(-np.pi, np.pi, 256)
C = np.cos(X)
plt.title("Grid 제거")
plt.plot(X, C)
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], ["Low", "Zero", "High"])
plt.grid(False)
plt.show()
```

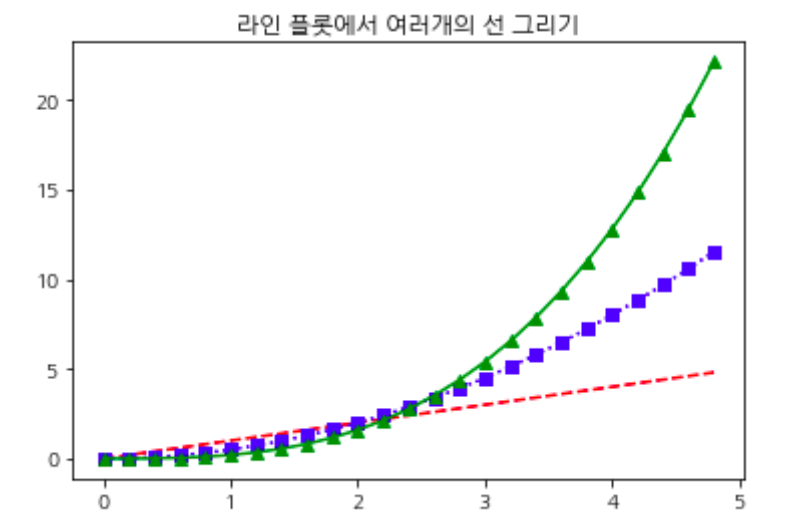



## 여러 개의 선을 그리기

라인 플롯에서는 여러 개의 선을 그릴 때, x 데이터, y 데이터, 스타일의 문자열을 반복하여 인수로 넘긴다.

**In [13]:**

```
t = np.arange(0., 5., 0.2)
plt.title("라인 플롯에서 여러개의 선 그리기")
plt.plot(t, t, 'r--', t, 0.5 * t**2, 'bs:', t, 0.2 * t**3, 'g^-')
plt.show()
```

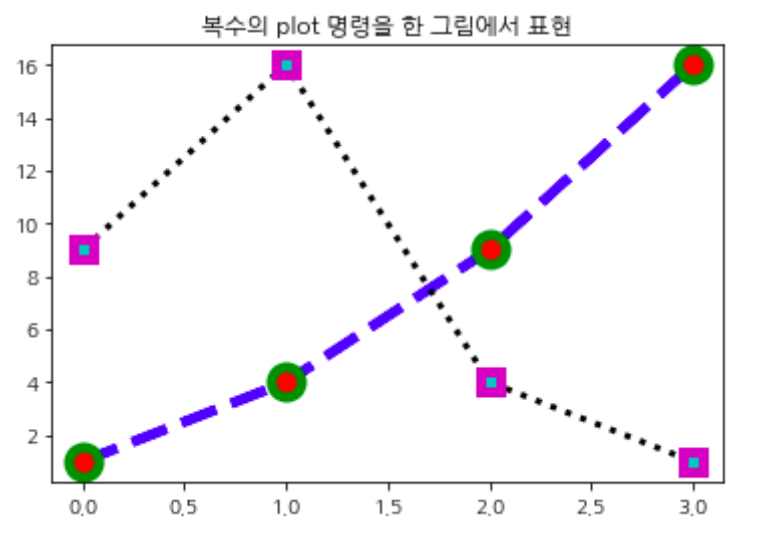



## 겹쳐 그리기

하나의 plot 명령이 아니라 복수의 plot 명령을 하나의 그림에 겹쳐서 그릴 수도 있다.

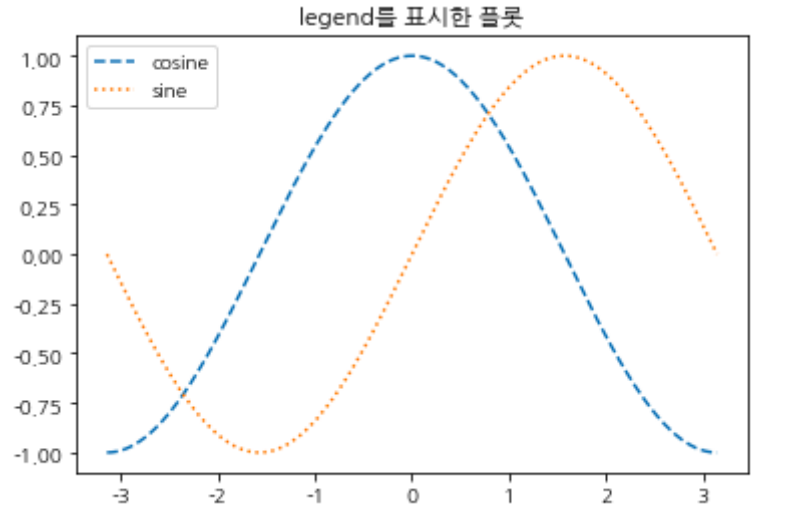
**In [14]:**

```
plt.title("복수의 plot 명령을 한 그림에서 표현")
plt.plot([1, 4, 9, 16],
         c="b", lw=5, ls="--", marker="o", ms=15, mec="g", mew=5,
mfc="r")
# plt.hold(True)   # <- 1,5 버전에서는 이 코드가 필요하다.
plt.plot([9, 16, 4, 1],
         c="k", lw=3, ls=":", marker="s", ms=10, mec="m", mew=5,
mfc="c")
# plt.hold(False)  # <- 1,5 버전에서는 이 코드가 필요하다.
plt.show()
```




**In [15]:**

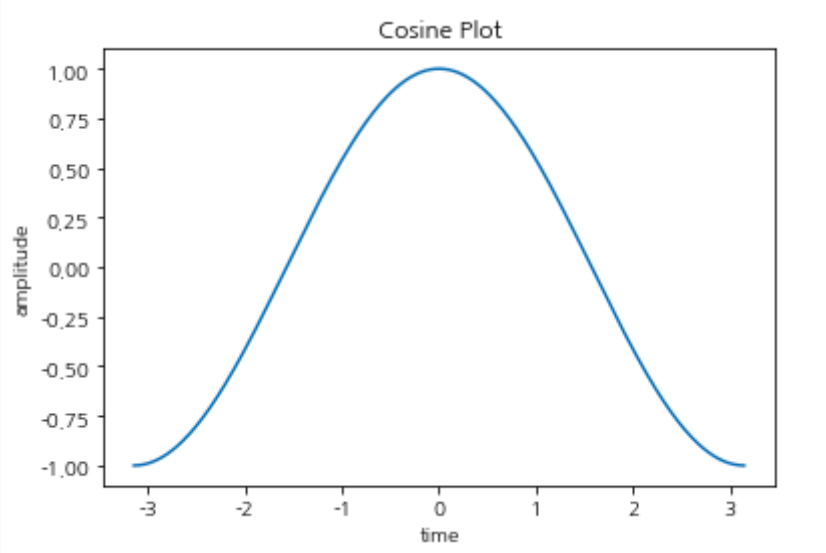
```
X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)
plt.title("legend를 표시한 플롯")
plt.plot(X, C, ls="--", label="cosine")
plt.plot(X, S, ls=":", label="sine")
plt.legend(loc=2)
plt.show()
```



## x축, y축 라벨, 타이틀

xlabel. ylabel 명령을 사용해 라벨을 붙일 수 있다.

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




## 그림의 구조

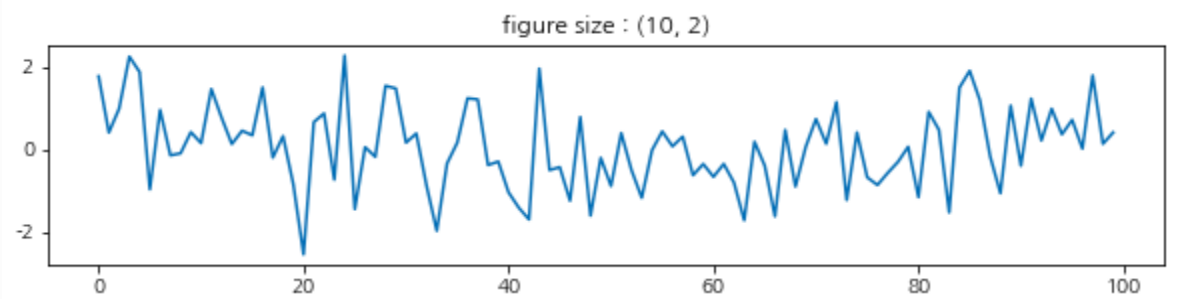
$$Axis(Axis\text{는 한 개 이상}) \subset Axes(Axis\text{는 두 개 이상}) \subset Figure$$

**Figure 객체**

모든 그림은 Figure 객체이다. (Matplotlib.figure.figure)

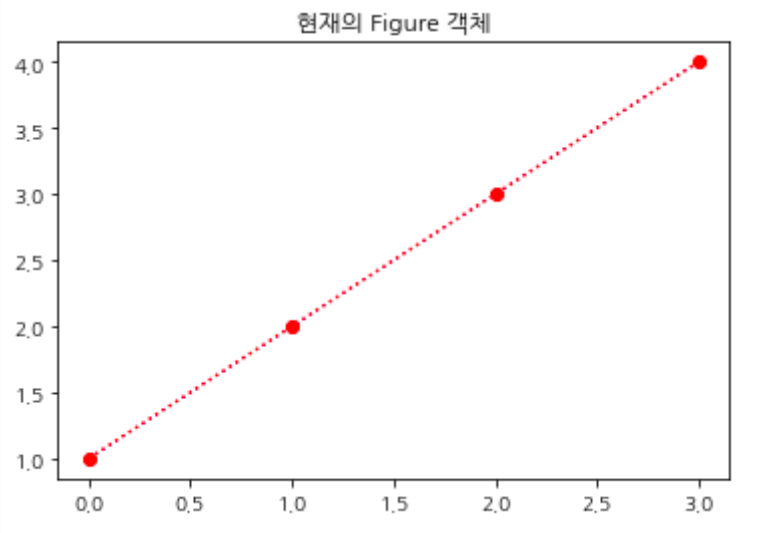
**In [17]:**

```
np.random.seed(0)
f1 = plt.figure(figsize=(10, 2))
plt.title("figure size : (10, 2)")
plt.plot(np.random.randn(100))
plt.show()
```




**In [18]:** gcf 명령을 통한 Figure 객체 할당.

```
f1 = plt.figure(1)
plt.title("현재의 Figure 객체")
plt.plot([1, 2, 3, 4], 'ro:')

f2 = plt.gcf()
print(f1, id(f1))
print(f2, id(f2))
plt.show()
```



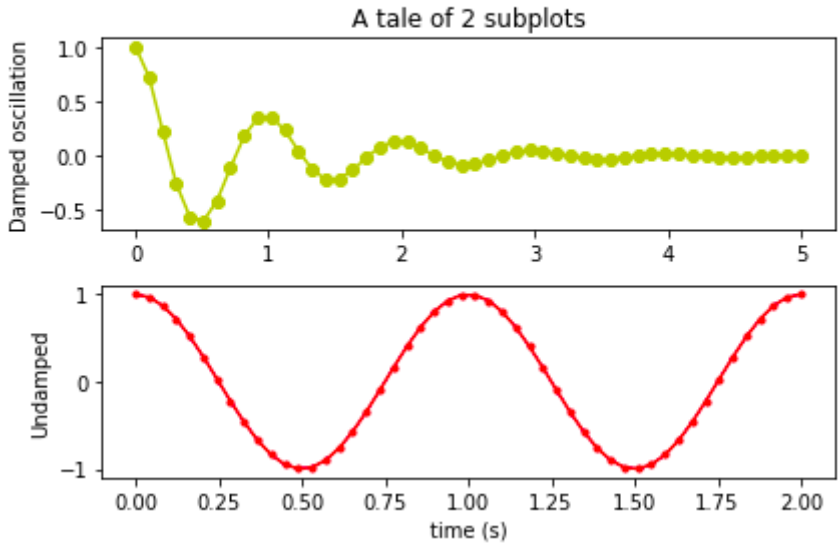

**Axes 객체와 subplot**

하나의 Figure 안에 여러 개의 플롯을 배열 형태로 보여야하는 경우도 있다. Figure 안에 있는 각각의 플롯은 Axes라고 불리는 객체에 속한다.

**In [19]:**

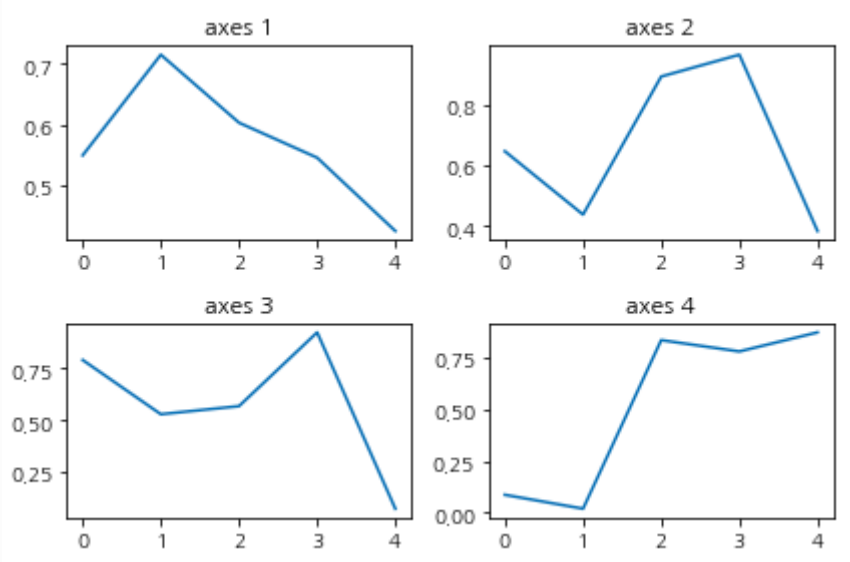
```
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)

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.tight_layout()
plt.show()
```

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



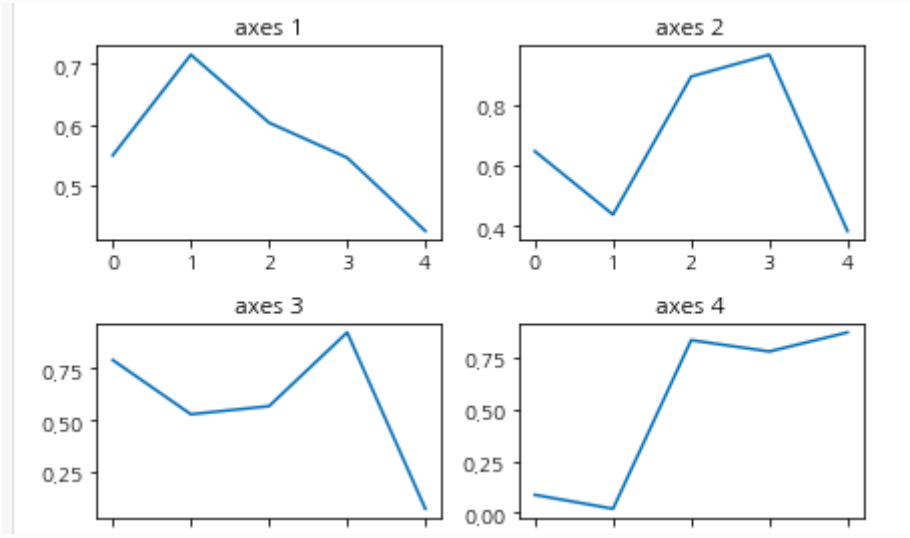

**In [20]:**

```
np.random.seed(0)

plt.subplot(221)
plt.plot(np.random.rand(5))
plt.title("axes 1")

plt.subplot(222)
plt.plot(np.random.rand(5))
plt.title("axes 2")

plt.subplot(223)
plt.plot(np.random.rand(5))
plt.title("axes 3")

plt.subplot(224)
plt.plot(np.random.rand(5))
plt.title("axes 4")

plt.tight_layout()
plt.show()
```




**In [21]:**

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

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()
plt.show()
```

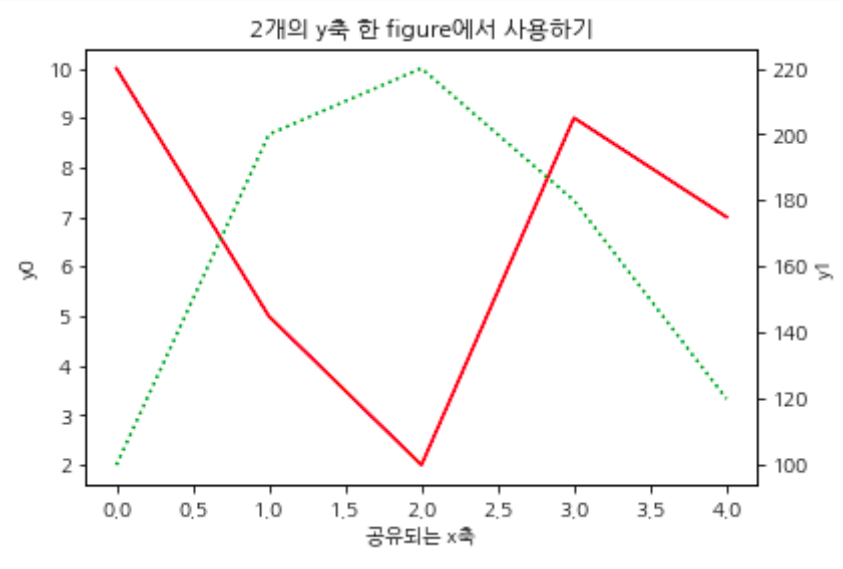



**Axis 객체와 축**

하나의 Axes 객체는 두 개 이상의 Axis 객체를 가진다. Axis 객체는 플롯의 가로축이나 세로축을 나타내는 객체이다.

**In [22]:**

```
fig, ax0 = plt.subplots()
ax1 = ax0.twinx()
ax0.set_title("2개의 y축 한 figure에서 사용하기")
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)
ax0.set_xlabel("공유되는 x축")
plt.show()
```




## 간단한 소감

Matplotlib을 통해, 실제 데이터를 시각화 해보면서 수식을 쉽게 그릴 수 있다는 점을 알 수 있었다. 이를 활용해, 요즘 주식에 관심이 있는데 효과적인 매도, 매수점을 찾기 위한 프로그램도 만들 수 있을 것 같다는 생각을 했다. 수치해석을 열심히 공부해서 꼭 이 프로그램을 개발해 보고 싶다.