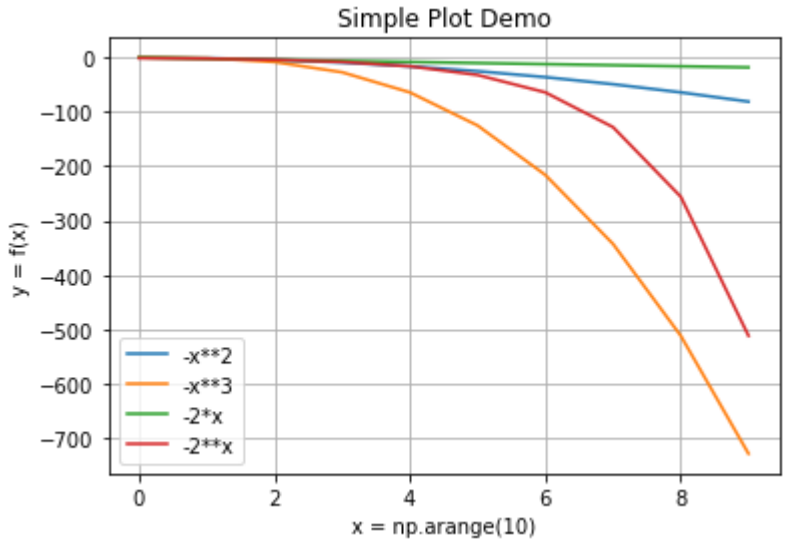
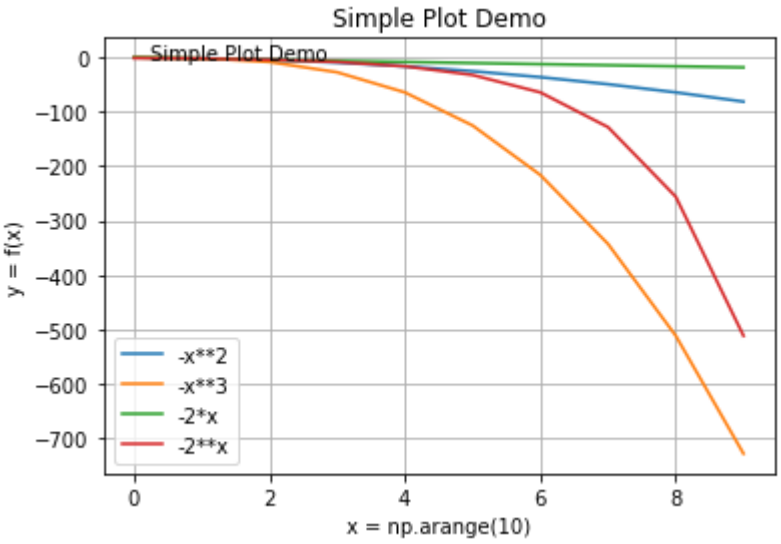
In [3]:

```
# Object-Oriented Plotting
# You can create plots in an object-oriented way.
# Note that we are using the axis object to plot
# and set the labels and a title.

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(10)
fig, ax = plt.subplots()
ax.plot(x, -x**2, label='-x**2')
ax.plot(x, -x**3, label='-x**3')
ax.plot(x, -2*x, label='-2*x')
ax.plot(x, -2**x, label='-2**x')
ax.set_xlabel('x = np.arange(10)')
ax.set_ylabel('y = f(x)')
ax.set_title('Simple Plot Demo')
ax.legend()
ax.grid(True)
plt.show()
```



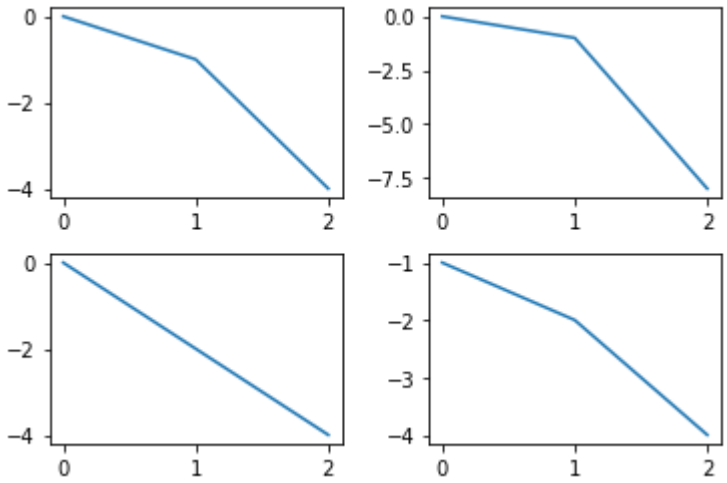
In [4]:

```
# You can also add the text with the functions ax.text()
# or the function plt.text(). The functions accept the coordinates
# and the text to be displayed. The following is an example:

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(10)
fig, ax = plt.subplots()
ax.plot(x, -x**2, label='-x**2')
ax.plot(x, -x**3, label='-x**3')
ax.plot(x, -2*x, label='-2*x')
ax.plot(x, -2**x, label='-2**x')
ax.set_xlabel('x = np.arange(10)')
ax.set_ylabel('y = f(x)')
ax.set_title('Simple Plot Demo')
ax.legend()
ax.grid(True)
ax.text(0.25, -5, "Simple Plot Demo")
plt.show()
```



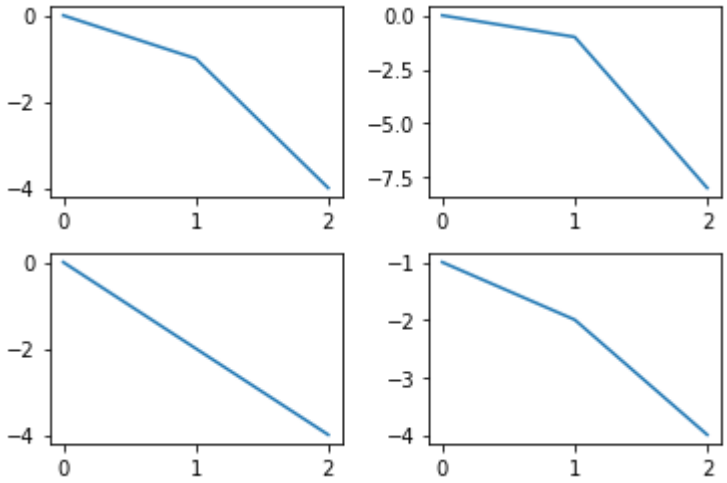
In [6]:

```
# Subplots
# You can show multiple separate graphs in the same output.
# The technique is known as subplotting.
# Subplots can have their own titles, own labels,
# and other specifications. Subplots are created in a grid.
# The first subplot position is at the top left. The other
# subplot positions are relative to the first position.
# The following is an example:

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(3)
plt.subplots_adjust(wspace=0.3,hspace=0.3)
plt.subplot(2, 2, 1)
plt.plot(x, -x**2)
plt.subplot(2, 2, 2)
plt.plot(x, -x**3)
plt.subplot(2, 2, 3)
plt.plot(x, -2*x)
plt.subplot(2, 2, 4)
plt.plot(x, -2**x)
plt.show()

# The first two arguments passed to plt.subplot() represent
# the grid size, and the third argument indicates the position
# of that particular subplot.
```



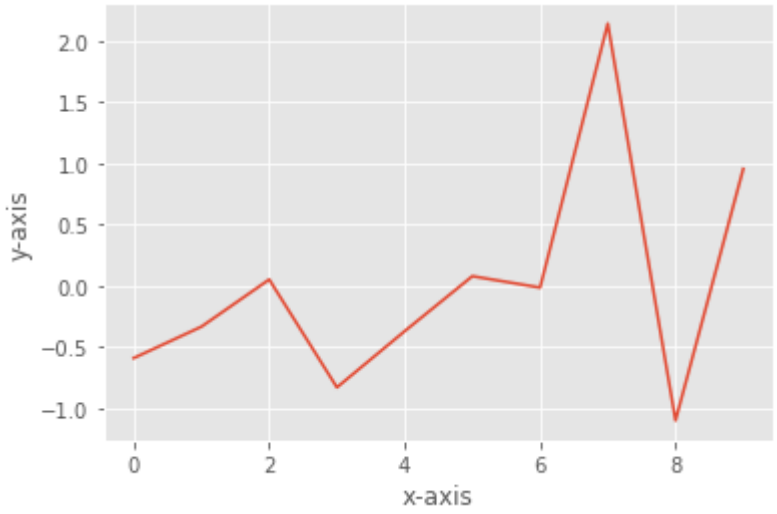
In [7]:

```
# You can write the same code in object-oriented fashion as follows:

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(3)
fig, axs = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.3,hspace=0.3)
axs[0, 0].plot(x, -x**2)
axs[0, 1].plot(x, -x**3)
axs[1, 0].plot(x, -2*x)
axs[1, 1].plot(x, -2**x)
plt.show()
```



In [9]:

```
# Styles
# A style dictates things such as marker size, colors, and fonts.
# There are many built-in styles in Matplotlib. The following is a short
# example of applying a built-in style:

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('ggplot')
data = np.random.randn(10)
plt.plot(data)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()

# Here we are using the style of ggplot2, which is a
# visualization package for the R programming language.
```



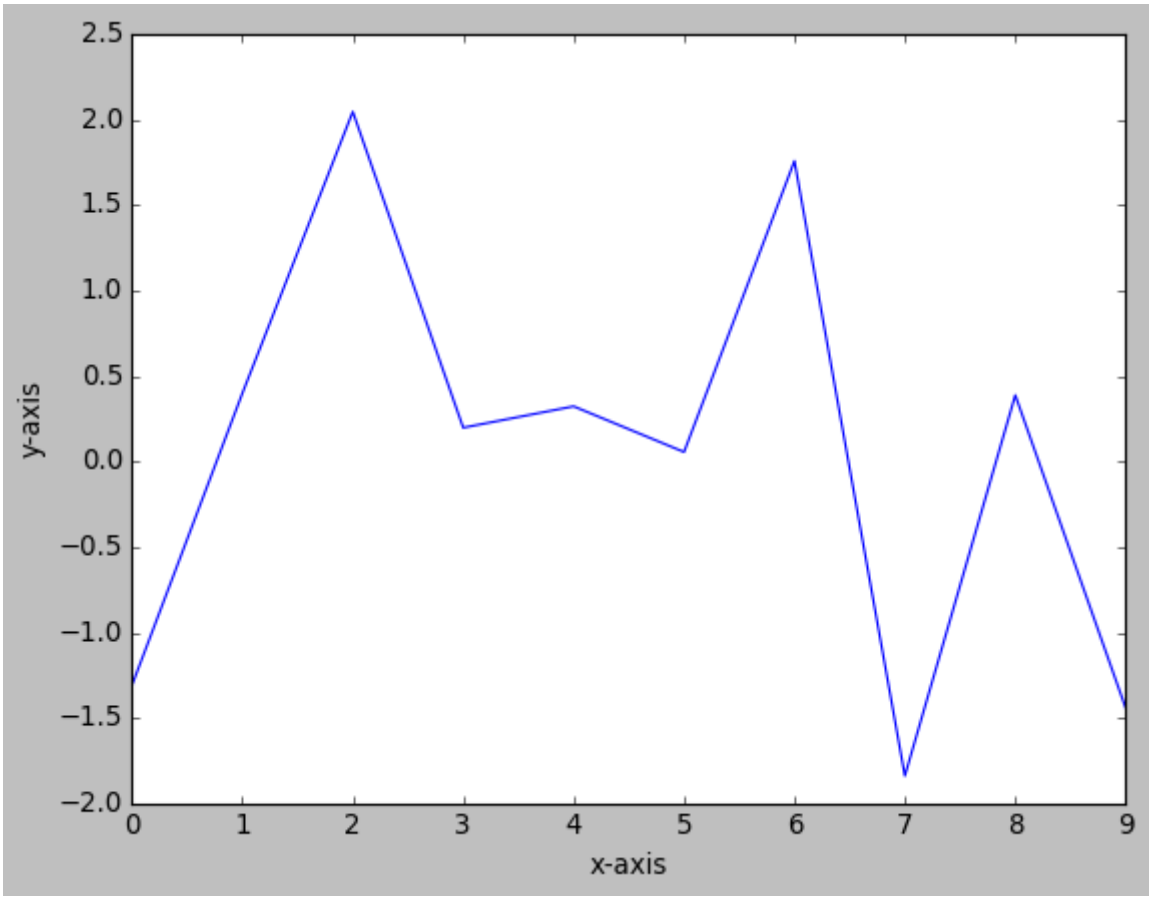
In [10]:

```
# You must be curious to know the names of all the available styles.
# You can print the names using this:

print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_backgro
und', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seabor
n-bright', 'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette',
'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook',
'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seab
orn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10']
```

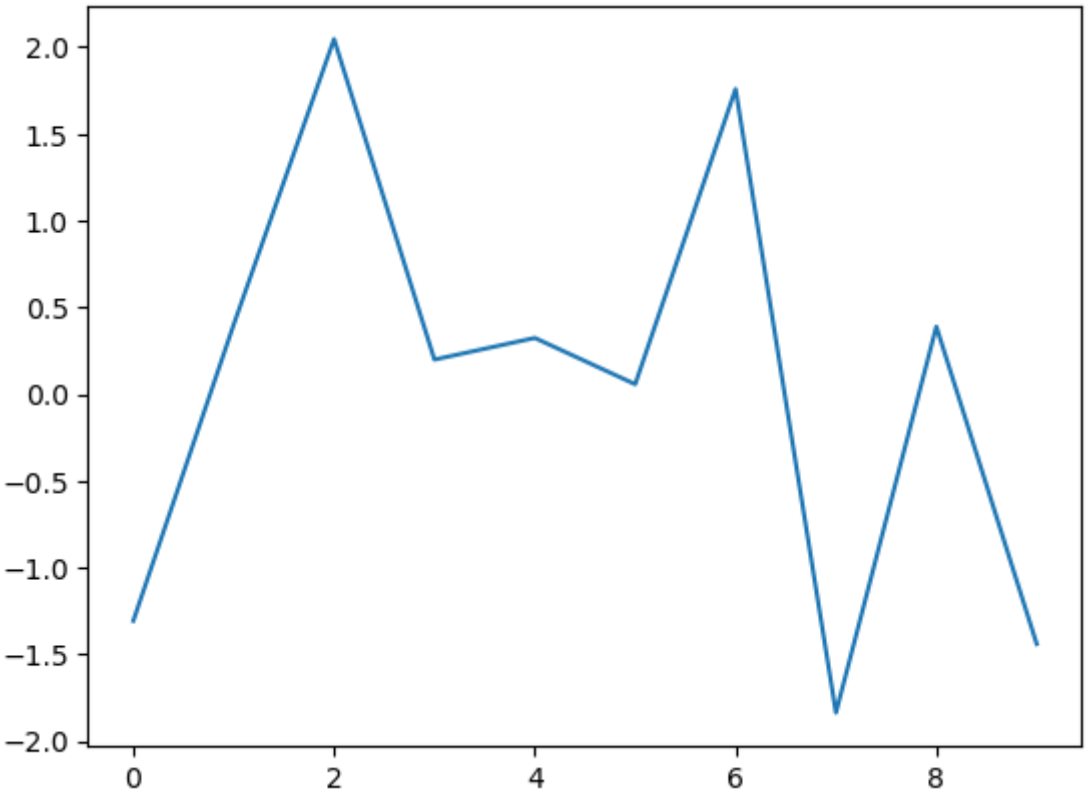
In [11]:

```
# Let's apply the classic matplotlib style as follows:

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
data = np.random.randn(10)
plt.style.use('classic')
plt.plot(data)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```



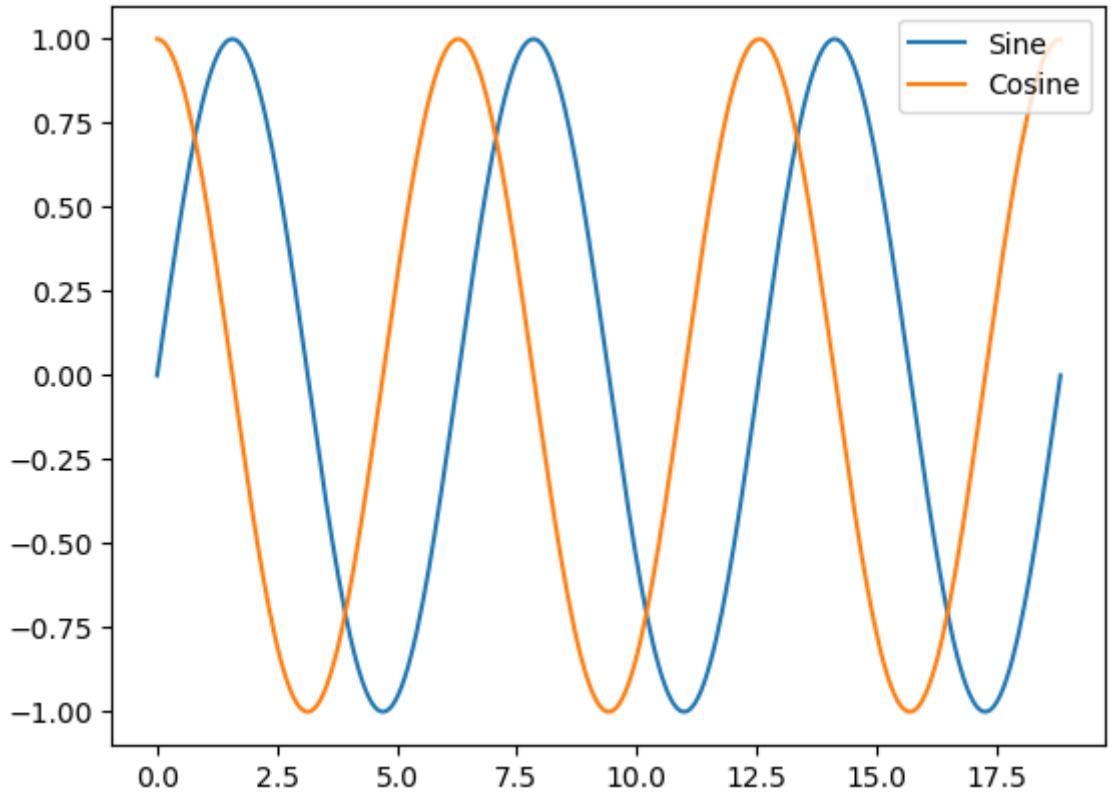
In [12]:

```
# Note that once you apply a style, that style applies to the
# entire notebook. So, if you want to switch back to the default style,
# you can use the following code:

plt.style.use('default')
plt.plot(data)
plt.show()
```



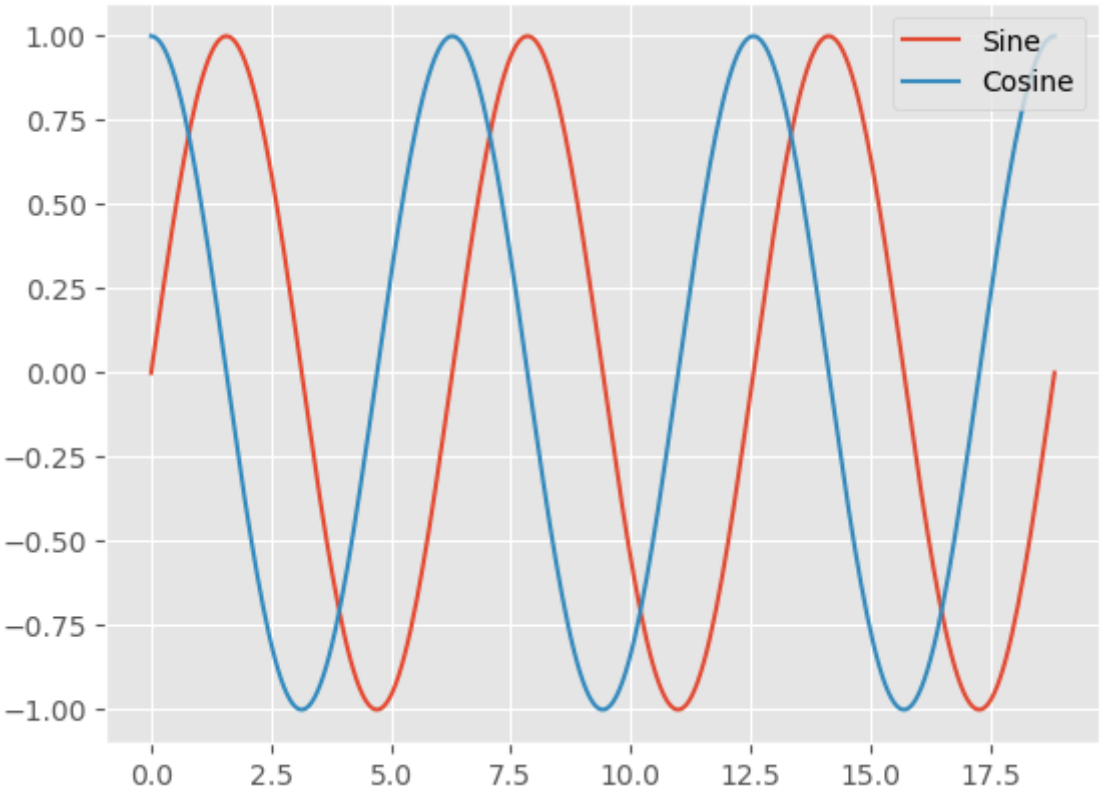
In [13]:

```
# Now let's demonstrate how the colors are affected
# when we change the styles. Let's define the data as shown here:

import matplotlib.pyplot as plt
import numpy as np
n = 3
data = np.linspace(0, 2*n*np.pi, 300)
# In addition, let's define a custom function as follows:
def sinusoidal(sty):
    plt.style.use(sty)
    fig, ax = plt.subplots()
    ax.plot(data, np.sin(data), label='Sine')
    ax.plot(data, np.cos(data), label='Cosine')
    ax.legend()

# A function is a routine that can be called to perform some operation.
# Until now, we have been using library functions that come with Python
# itself and libraries like NumPy and Matplotlib.
# Here, in the code snippet, we have defined our own custom function.
# This custom function accepts an argument.
# We are using the passed argument as a style for our visualization.
# Let's call this function with the default styling, as shown here:

sinusoidal('default')
plt.show()
```



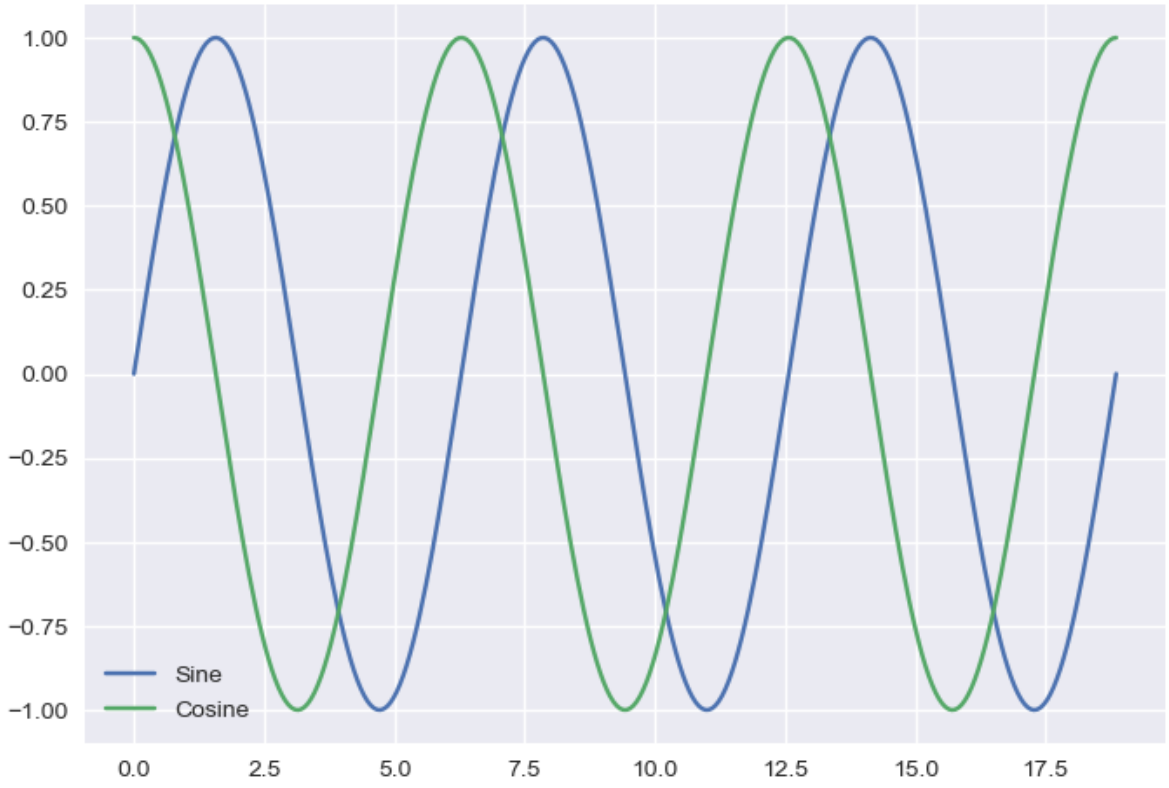
In [14]:

```
# Let's use the ggplot style as follows:

sinusoidal('ggplot')
plt.show()
```



In [15]:

```
# Let's see the Seaborn style, as shown here:

sinusoidal('seaborn')
plt.show()
```



In [16]:

```
# You have seen that the styling is applied globally to the
# entire notebook, and you have learned to switch to the default
# styling. You can locally change the styling for a
# block of code as follows:

with plt.style.context('Solarize_Light2'):
    data = np.linspace(0, 6 * np.pi)
    plt.plot(np.sin(data), 'g.--')
    plt.show()
```