Mejorando los gráficos

Matplotlib ofrece multitud de posibilidades para mejorar la calidad de nuestros gráficos. Se recomienda leer la referencia oficial para una información completa.

En este cuaderno, vamos a centrarnos en los siguientes aspectos: las leyendas y etiquetas, configuración de los ejes, añadir anotaciones al grafo y seleccionar un estilo general.

Configuración del entorno.

Lo primero es configurar el entorno de ejecución. Véase el cuaderno "Primeros pasos con Matplotlib" para más detalles.

In [1]:
```
%matplotlib notebook
import matplotlib as mpl
import matplotlib.pyplot as plt
print('Matplotlib version: {}'.format(mpl.__version__))
import numpy as np
np.set_printoptions(floatmode='fixed', precision=3)
import pandas as pd
```

```
Matplotlib version: 3.5.2
```

Mejorando las leyendas y etiquetas.

Ya hemos visto que para añadir una leyenda a un objeto Axes, necesitamos en primer lugar haber asignado una etiqueta con el argumento `label` a cada gráfico dibujado en el objeto Axes.

Después usando el método `Axes.legend()` activamos que se visualice una caja con las leyendas de cada gráfico dibujado.

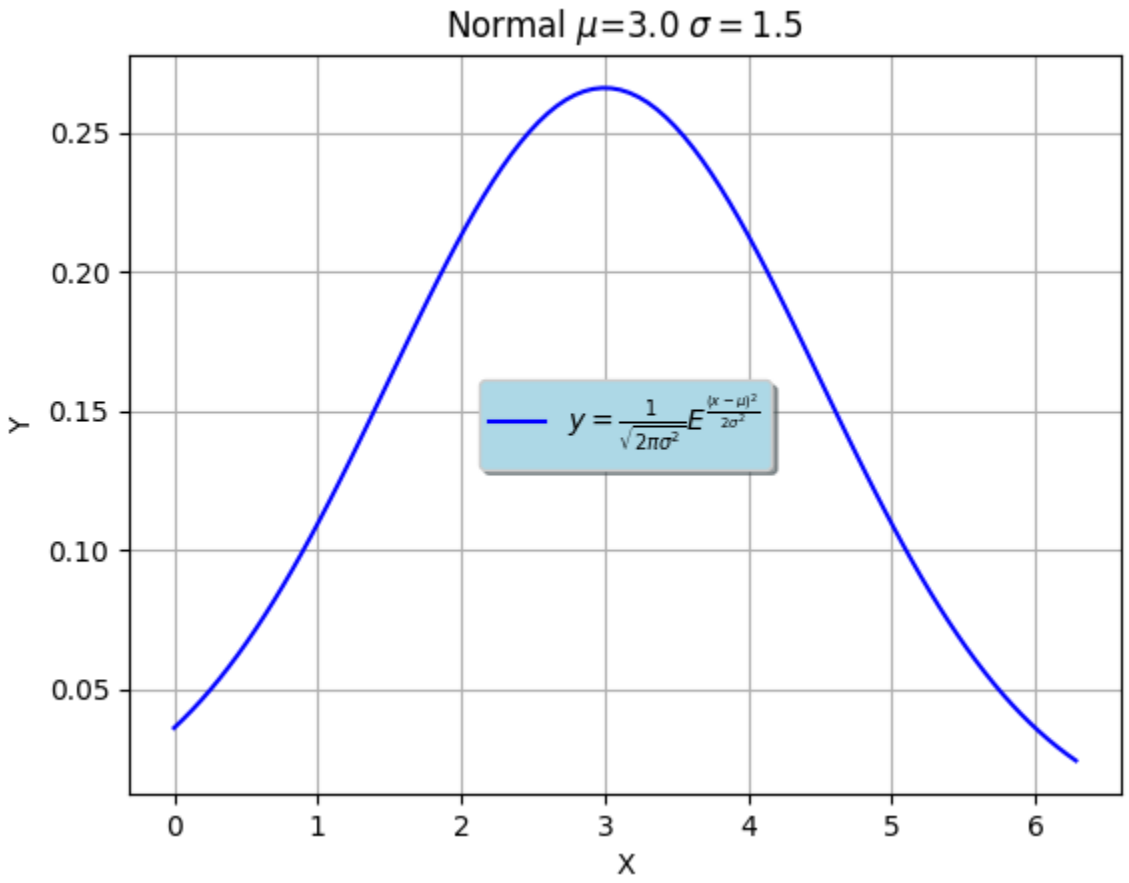
Si el texto de la etiqueta de una de los gráficos usa expresiones matemáticas podemos utilizar el un subconjunto del lenguaje de marcas Tex para escribir estas expresiones usando la notación `label=r'$texto Tex$'`. En esta referencia puedes consultar la notación empleada.

Ejercicio 01: Dibujar un gráfico de línea con la función

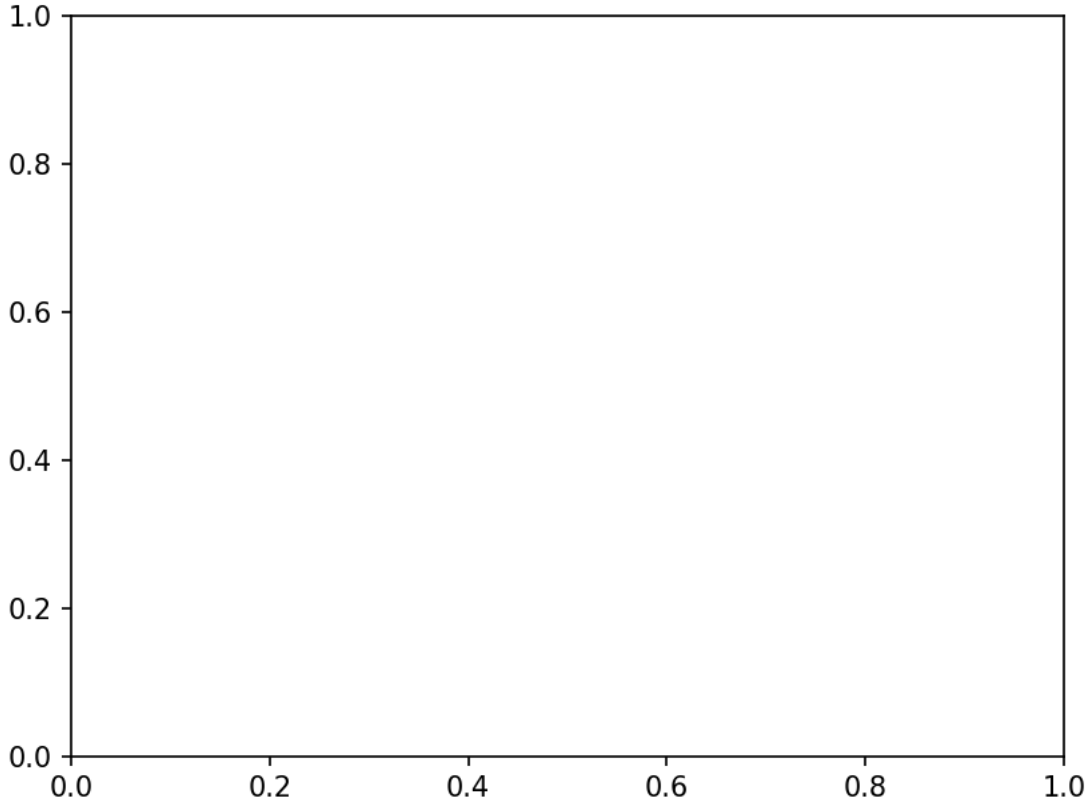
$$y = \frac{1}{\sqrt{2\pi\sigma^2}} E^{\frac{(x-\mu)^2}{2\sigma^2}}$$

con $\mu = 3.0$, $\sigma = 1.5$ y $x \in [0, 6]$ con 100 muestras.

Intenta ajustar los parámetros para que el resultado sea los más parecido posible a la siguiente figura.



In [2]:
```
mu = 3.0
sigma = 1.5
x  = []
y = []
#Pon tu código aquí.
#Sugerencia: genera un intervalo lineal $x\in[0, 2\pi]$
#con 100 muestras y úsalo para calcular y
# y2 = [sin(x)]^2;

#

fig = plt.figure()
ax = plt.axes()

#Pon tu código aquí
#Sugerencia: añade el título las leyendas de los ejes.
#Usa Tex para los símbolos griegos.

#

#Pon tu código aquí.
#Sugerencia: Utiliza lenguaje Tex para indicar las etiquetas.
# y los parámetros apropiados de Axes.legend() para generar la
#leyenda.

#
```



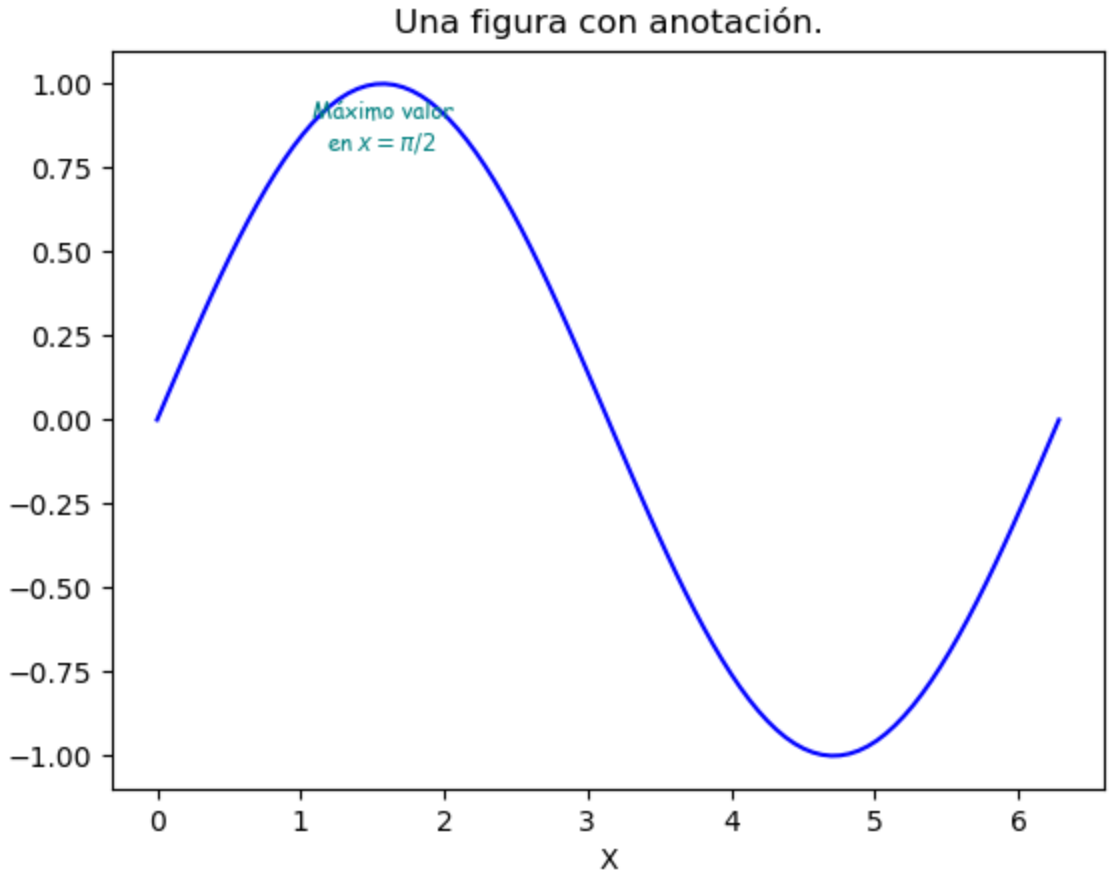
Añadiendo anotaciones a una figura.

Es posible añadir anotaciones a un gráfico. Para añadir un texto usaremos el método `Axes.text()`.

Hay que indicar las coordenadas x,y donde añadir el texto. Por defecto estas coordenadas se referencian respecto a los datos visualizados en el objeto Axes.

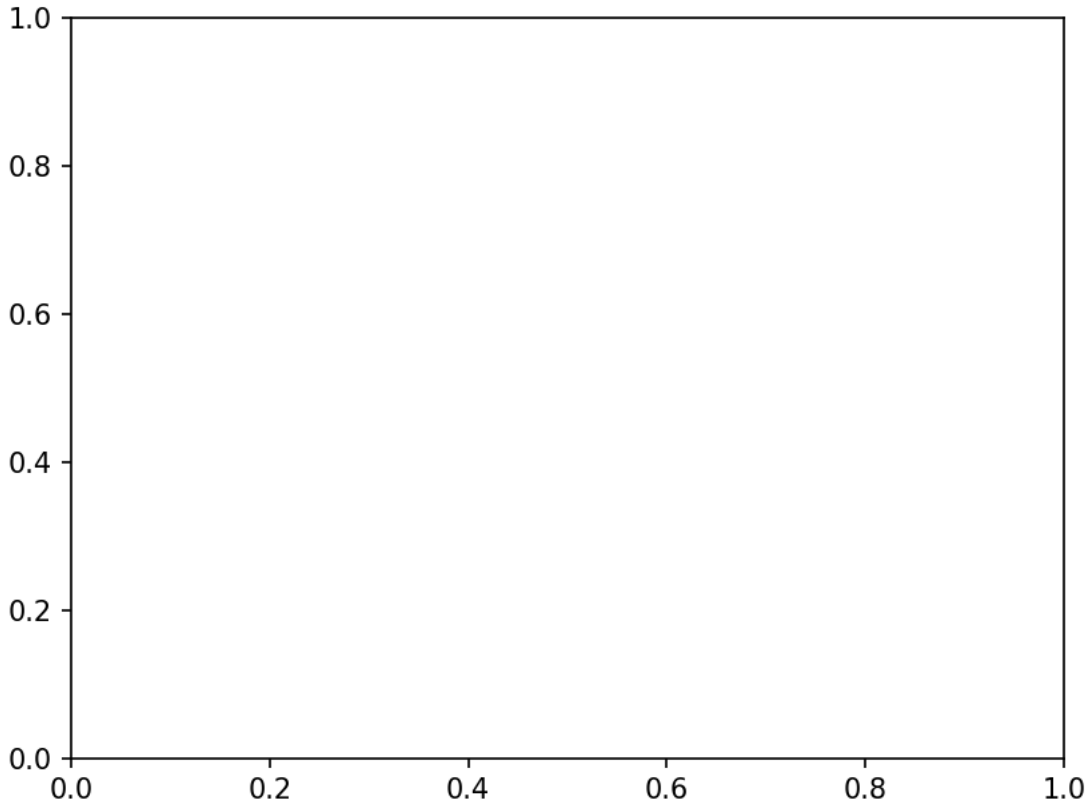
Ejercicio 02: Visualizar la función $\sin(x)$ y añadir una etiqueta cercana al valor máximo localizado en $x = \pi/2$.

Intenta ajustar los parámetros para que el resultado sea lo más parecido posible a la siguiente figura.



In [3]:

```
x = []
y = []
#Pon tu código aquí.
#Sugerencia: generar un intervalo lineal x=[0,2pi] con 100
#muestras y evalua la función y=sin(x)

#

fig = plt.figure()
ax = plt.axes()

#Pon tu código aquí.
#Sugerencia: añade el título y la etiquetas a los ejes.

#

#Pon tu código aquí.
#Sugerencia: dibuja un gráfico de línea.

#

#Pon tu código aquí.
#Sugerencia: usa el método ax.text() con los parámetros convenientes.
#Usa la familia de fuentes 'fantasy' y usa código latex en el
#texto para escribir el texto. La alineación horizontal será centrada.

#
```



Como se ha dicho, las coordenadas se indican por defecto con respecto a los datos (`transform=ax.transData`), pero podríamos dar las coordenadas respecto al objeto Axes con `transform=ax.transAxes` . Alternativamente podemos anotar la figura como un todo usando el método Figure.text().

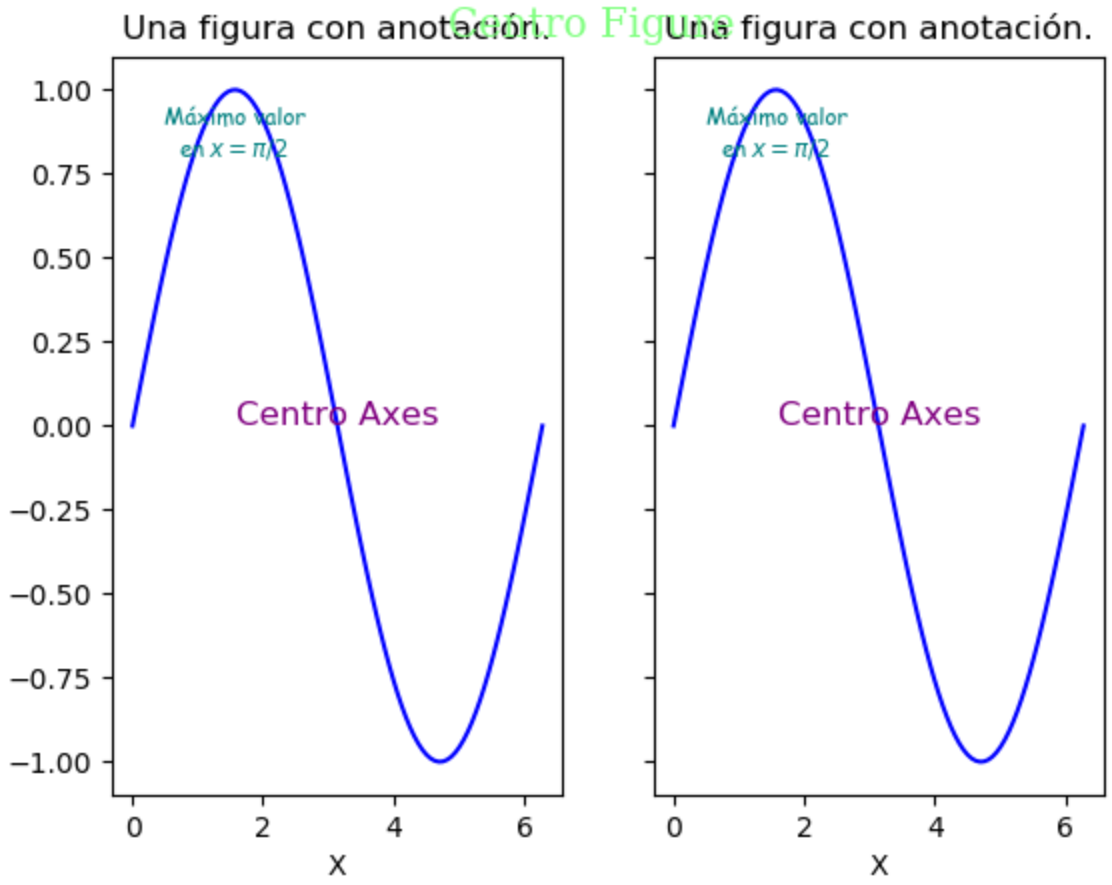
En estos dos últimos casos, se asume que los valores x,y de las coordenadas se dan en el intervalo $[0, 1]$.

Ejercicio 03: Repetir el ejercicio anterior repitiendolo en dos subplots y añadiendo dos anotaciones más:

- `Centro Axes` en coordenadas (0.5, 0.5) relativas al objeto Axes. - `Centro Figure` en coordenadas (0.5, 0.8) relativas al objeto Figure.

En todos los casos usar alineación horizontal centrada.

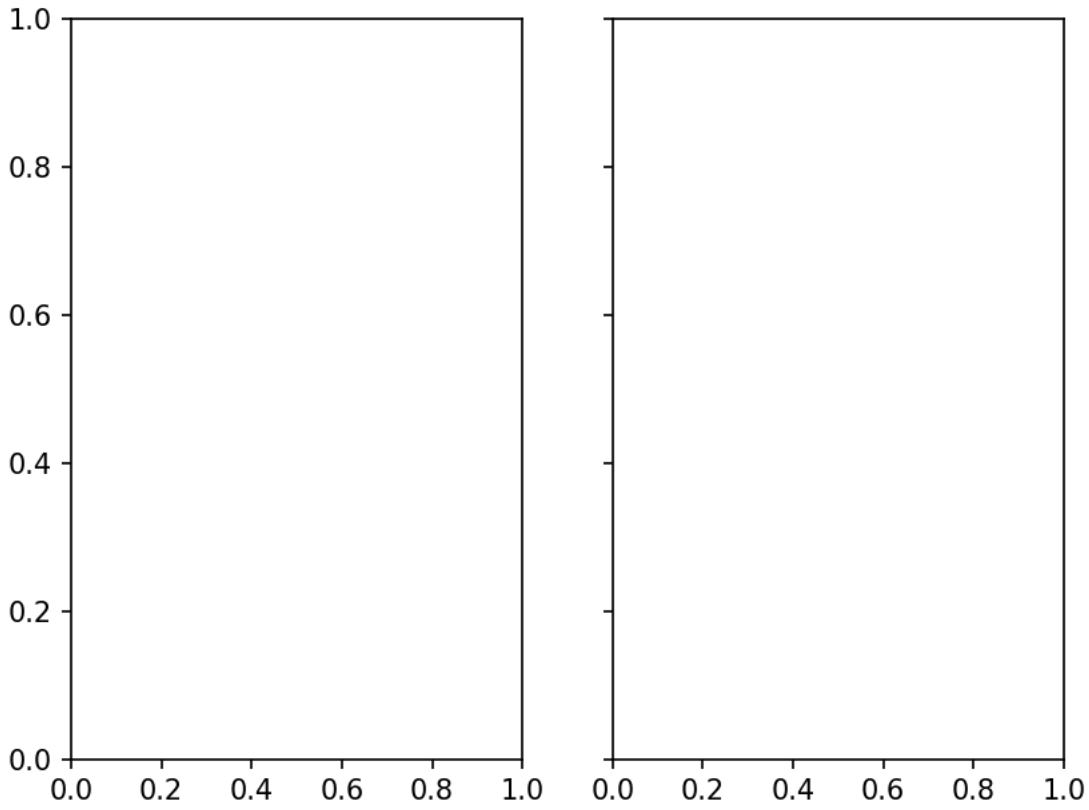
Intenta ajustar los parámetros para que el resultado sea lo más parecido posible a la siguiente figura.



In [4]:

```
x = []
y = []
#Pon tu código aquí.
#Sugerencia: generar un intervalo lineal x=[0,2pi] con 100
#muestras y evalua la función y=sin(x)

#

fig, axs = plt.subplots(1, 2, sharey=True)
#Usa el ejemplo anterior para generar dos subplots.

#Pon tu código aquí.
#Sugerencia: añade el título y la etiquetas a los ejes.

#

#Pon tu código aquí.
#Sugerencia: dibuja un gráfico de línea.

#

#Pon tu código aquí.
#Sugerencia: usa el método ax.text() con los parámetros convenientes.
#Usa la familia de fuentes 'fantasy' y usa código latex en el
#texto para escribir el texto. La alineación horizontal será centrada.

#

#Pon tu código aquí.
#Sugerencia: añade el título y la etiquetas a los ejes.

#
```

```
#Pon tu código aquí.
#Sugerencia: dibuja un gráfico de línea.

#

#Pon tu código aquí.
#Sugerencia: usa el método ax.text() con los parámetros convenientes.
#Usa la familia de fuentes 'fantasy' y usa código latex en el
#texto para escribir el texto. La alineación horizontal será centrada.

#####

#Pon tu código aquí.
#Sugerencia: usa el método ax.text() con los parámetros convenientes
#para poner el texto 'Centro Axes' en las coordenadas (0.5,0.5)
#relativa al objeto Axes usando el atributo transform para
#indicar que se transforme según ax.transAxes.

#####

#Pon tu código aquí.
#Sugerencia: usa el método ax.text() con los parámetros convenientes.
#para poner el texto 'Centro Figure' relativo al objeto fig.

#
```



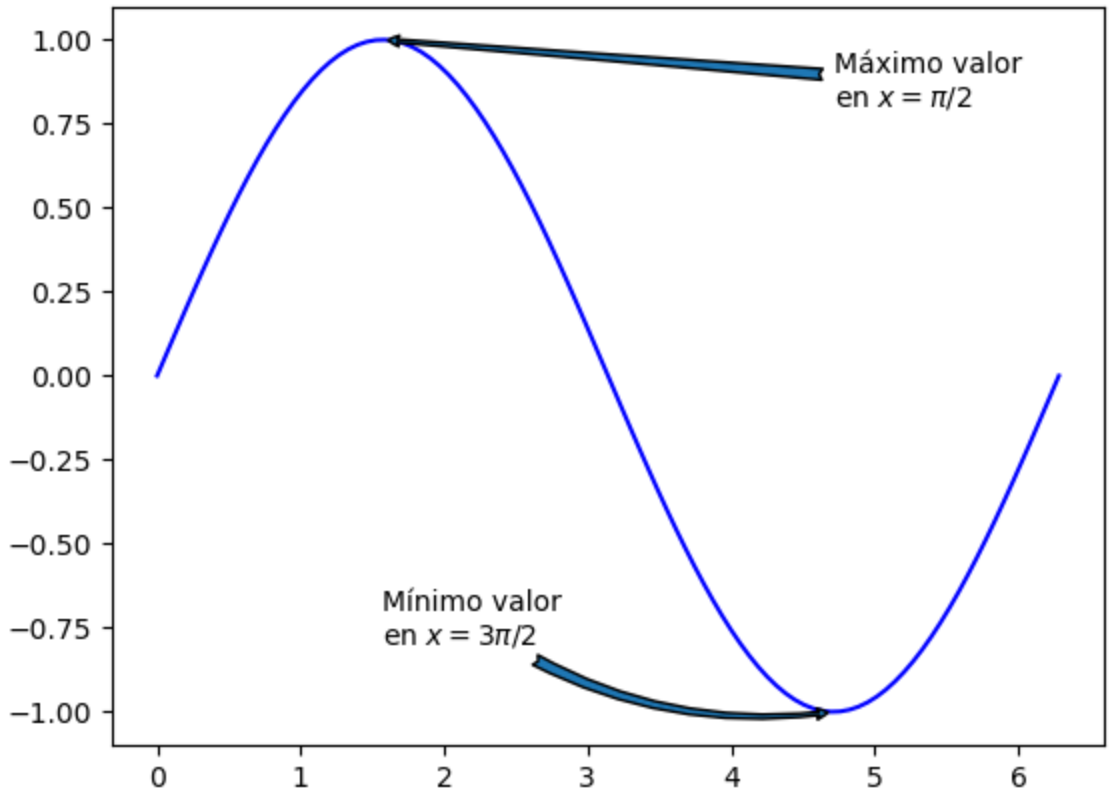
Si queremos combinar una flecha con un texto usaremos el comando `Axes.annotate()`. Este comando permite indicar el tipo de flecha y el texto a añadir de forma muy versátil. Esto se consigue con el parámetro `arrowprops` que es un diccionario.

Por ejemplo con `arrowprops = dict(arrowstyle='simple')` o podemos indicar una flecha curva con una longitud de arco con `arrowprops = dit(connectionstyle='arc3,rad=0.2')`. Se recomienda consultar la referencia oficial para ver otros estilos de flecha.

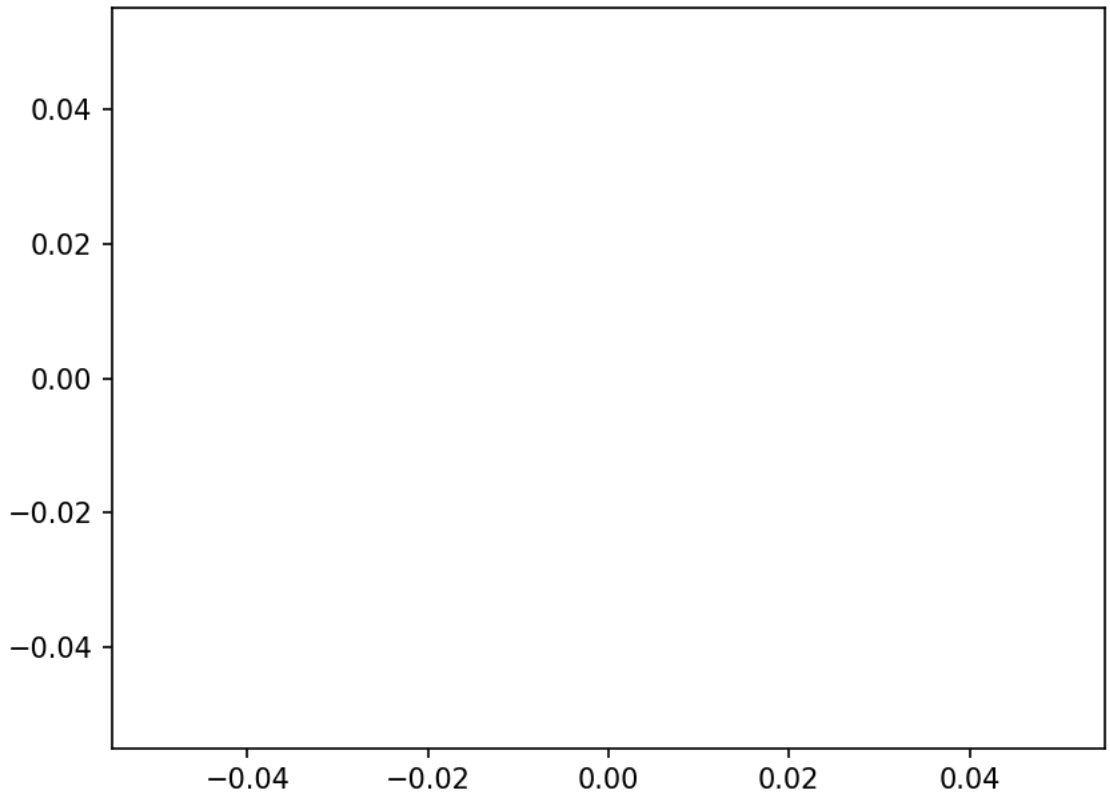
Ejercicio 04: Con los valores x, y calculados en los ejemplos anteriores, dibujar una gráfico de línea y añadir una anotación relativa al valor máximo en las coordeandas $(3\pi/2, 0.8)$ con una flecha apuntando a las coordenadas $(\pi/2, 1.0)$ y otra anotación relativa al valor mímimo en las coordenadas $(\pi/2, -0.8)$ con una flecha curva apuntando a las coordenadas $(3\pi/2, -1)$.

Utiliza el argumento `arrowprops` para indicar el estilo de línea 'fancy' y el rádio de la curva de la segunda flecha.

Intenta ajustar los parámetros para que el resultado sea lo más parecido posible a la siguiente figura.



In [5]:
```
fig = plt.figure()
ax = plt.axes()
ax.plot(x, y, "b-")

#Pon tu código aquí.
#Sugerencia: añade las fechas con el método annotate() del objeto
#Axes.
#Utiliza el argumento xy para indicar las coordenadas donde
#apunta la flecha y xytext con las coordenadas de la anotación.
#Usa el parámetro arrowprops (diccionario) para indicar las
#propiedades de las flechas: arrowstyle 'fancy',
#connectionstyle 'arc3,rad=0.2'

#
```



Out[5]: `[<matplotlib.lines.Line2D at 0x7fbf4524ddc0>]`

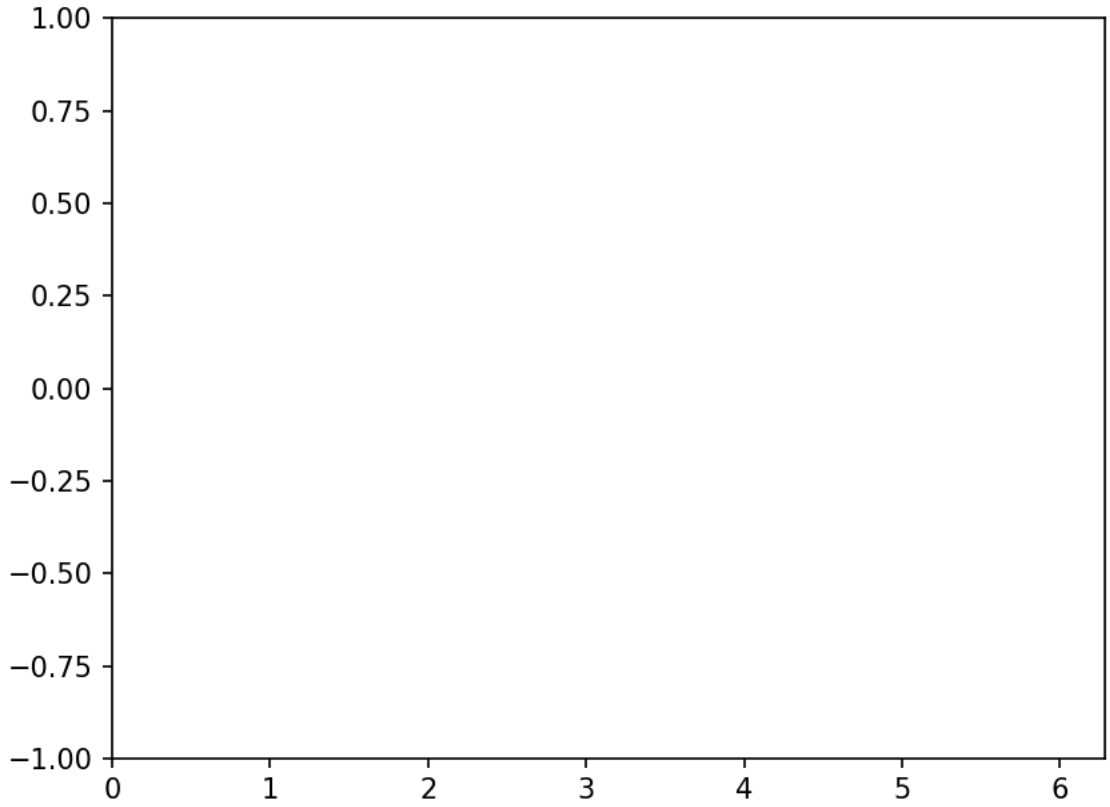
Mejorando los límites y marcas de los ejes.

Por defecto, los límites y las marcas mostrados en la leyenda de los ejes ('ticks') se seleccionan de forma automática a partir de los datos a mostrar y el aspecto obtenido es normalmente aceptable, pero habrá ocasiones donde queramos cambiar su aspecto.

Cada objeto Axes tiene dos atributos `Axes.xaxis` y `Axes.yaxis` que representan el eje X y el Y respectivamente.

Para ajustar los límites mostrados podemos usar los métodos `Axes.set_xlim()` y `Axes.set_ylim()` dependiendo del eje que se quiera modificar.

In [6]:
```
fig = plt.figure()
ax = plt.axes()
ax.set_xlim(0, 2*np.pi)
ax.set_ylim(-1, 1)
ax.plot(x, np.sin(x))
```



Out[6]:
```
[<matplotlib.lines.Line2D at 0x7fbf452240d0>]
```

Hay dos tipos de marcas en cada eje X/Y: marcas principales ('major') y las auxiliares ('minor').

Para representar estas marcas, cada objeto `Axes.xaxis` , `Axes.yaxis` tiene a su vez dos objetos asociados: un objeto `locator` que referencia el valor numérico en el eje y un objeto `formatter` que representa como dibujar el texto mostrado en las marcas.

Estos objetos se define en el módulo `matplotlib.ticker` . Veamos los valores por defecto que tienen estos objetos en los ejes generados en la figura anterior:

In [7]:
```
print('Objeto locator para marcas principales  :', ax.xaxis.get_major_locator())
print('Objeto formatter para marcas principales:', ax.xaxis.get_major_formatter())
print('Objeto locator para marcas auxiliares   :', ax.xaxis.get_minor_locator())
print('Objeto formatter para marcas auxiliares :', ax.xaxis.get_minor_formatter())
```

```
Objeto locator para marcas principales  : <matplotlib.ticker.AutoLocator object at 0x7fb
f451f0370>
Objeto formatter para marcas principales: <matplotlib.ticker.ScalarFormatter object at 0
x7fbf451f0940>
Objeto locator para marcas auxiliares   : <matplotlib.ticker.NullLocator object at 0x7fb
f451f0490>
Objeto formatter para marcas auxiliares : <matplotlib.ticker.NullFormatter object at 0x7
fbf452678b0>
```

Como vemos, por defecto, sólo se muestran los ticks principales ('major') por motivos de rendimiento. Pero podemos cambiar esta configuración por defecto.

Activando las marcas auxiliares.

Por ejemplo si sólo queremos mostrar los ticks auxiliares pero sin texto, podemos usar una objeto

`AutoMinorLocator()`:

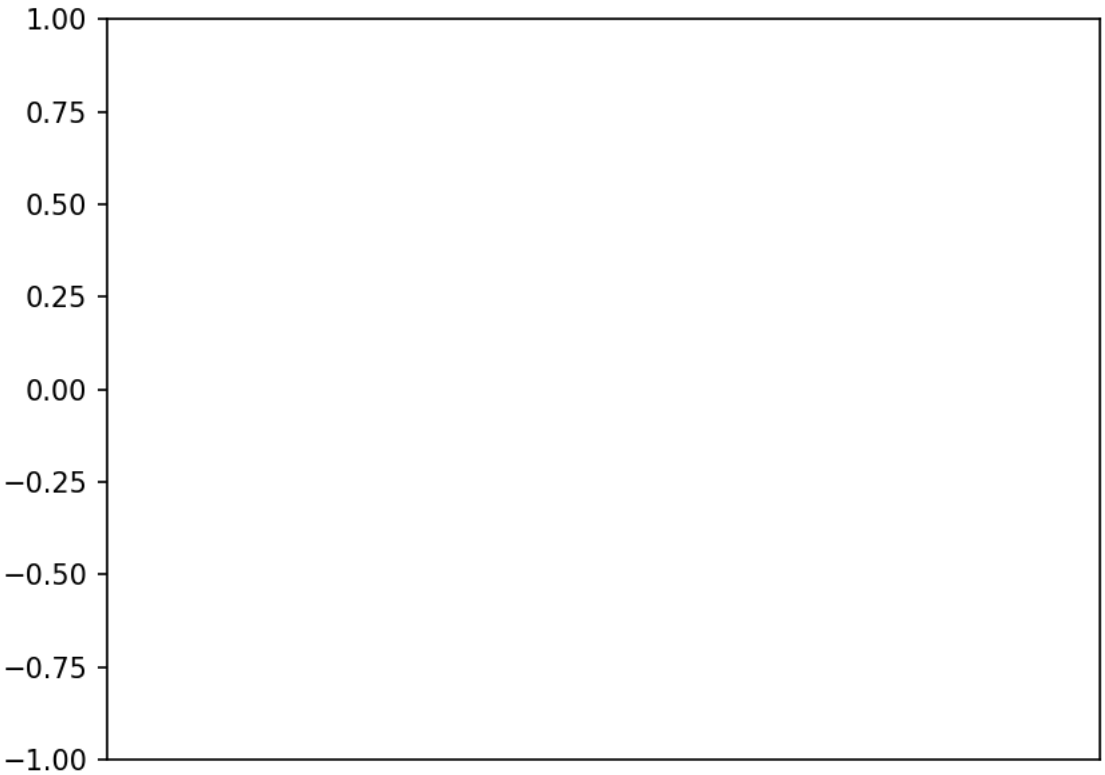
In [8]:
```
fig = plt.figure()
ax = plt.axes()
ax.set_xlim(0, 2.0*np.pi)
ax.set_ylim(-1, 1)
ax.xaxis.set_minor_locator(mpl.ticker.AutoMinorLocator())
ax.yaxis.set_minor_locator(mpl.ticker.AutoMinorLocator())
ax.plot(x, y)
ax.grid(which='major', linestyle='-', color='red' )
ax.grid(which='minor', linestyle=':', color='green')
```

Como vemos, sólo se han activado mostrar las marcas auxiliares pero sin mostrar sus valores. También hemos activado la rejilla diferenciando las líneas para las marcas principales de la auxiliares.

Ocultando las marcas.

Si lo que queremos es que no aparezcan ticks de uno u otro tipo usaremos un localizador `NullLocator`.

In [9]:
```
fig = plt.figure()
ax = plt.axes()
ax.set_xlim(0, 2.0*np.pi)
ax.set_ylim(-1, 1)
ax.xaxis.set_major_locator(mpl.ticker.NullLocator())
ax.plot(x, np.sin(x))
```

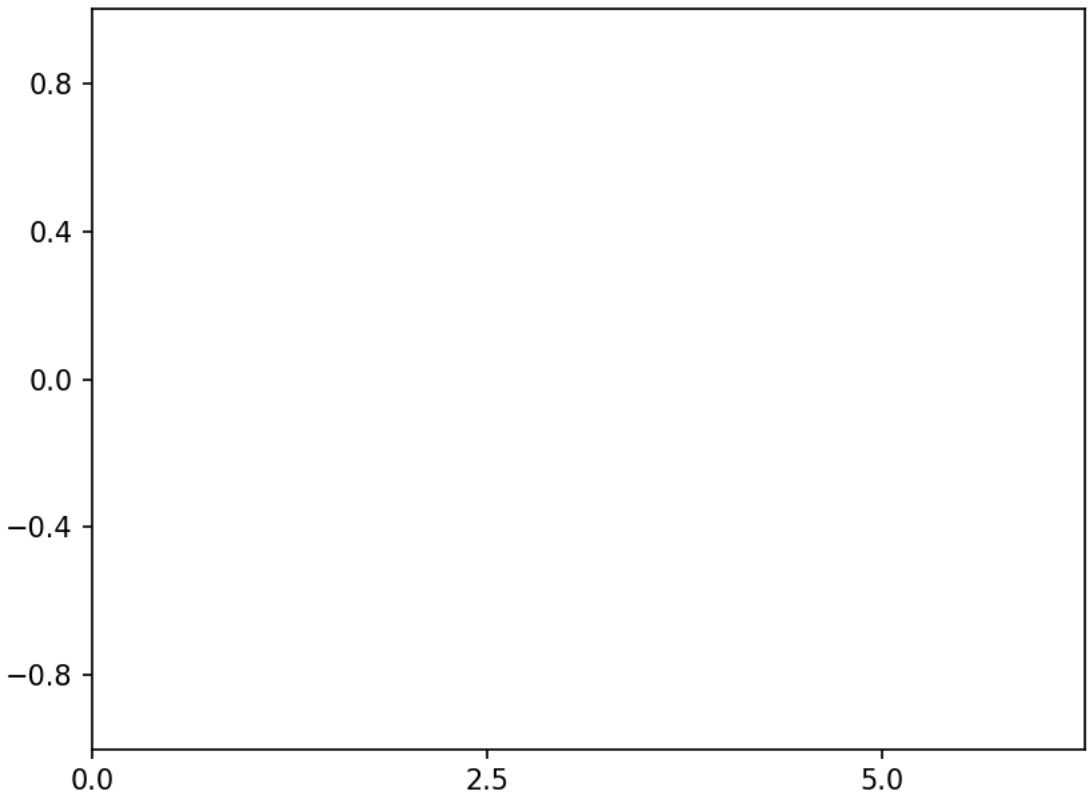


Out[9]:
```
[<matplotlib.lines.Line2D at 0x7fbf451a5460>]
```

Indicando el número máximo de marcas requeridas.

Por otro lado, si queremos ajustar el número de marcas a mostrar usamos un `MaxNLocator()` con el número de ticks.

In [10]:
```
fig = plt.figure()
ax = plt.axes()
ax.set_xlim(0, 2.0*np.pi)
ax.set_ylim(-1, 1)
ax.xaxis.set_major_locator(mpl.ticker.MaxNLocator(3))
ax.yaxis.set_major_locator(mpl.ticker.MaxNLocator(5))
ax.plot(x, np.sin(x))
```

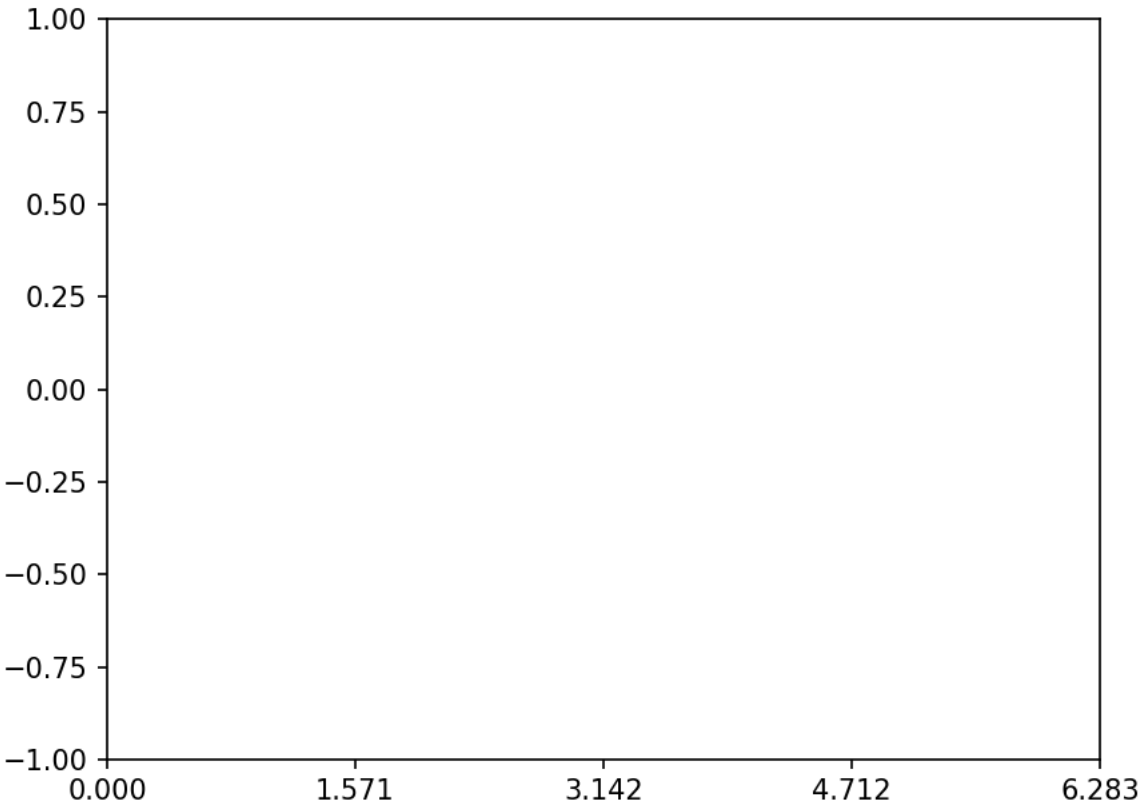


Out[10]:
```
[<matplotlib.lines.Line2D at 0x7fbf450ec6a0>]
```

Distribuyendo las marcas de forma uniforme.

Como vemos, por defecto el rango a mostrar se divide en N partes iguales. Otras veces puede ser interesante indicar el tamaño de las partes. Para ello se utiliza un `MultipleLocator()`. Por ejemplo vamos a crear partes con tamaño $\pi/2$.

In [11]:
```
fig = plt.figure()
ax = plt.axes()
ax.set_xlim(0, 2.0*np.pi)
ax.set_ylim(-1, 1)
ax.xaxis.set_major_locator(mpl.ticker.MultipleLocator(np.pi/2.0))
ax.plot(x, np.sin(x))
```

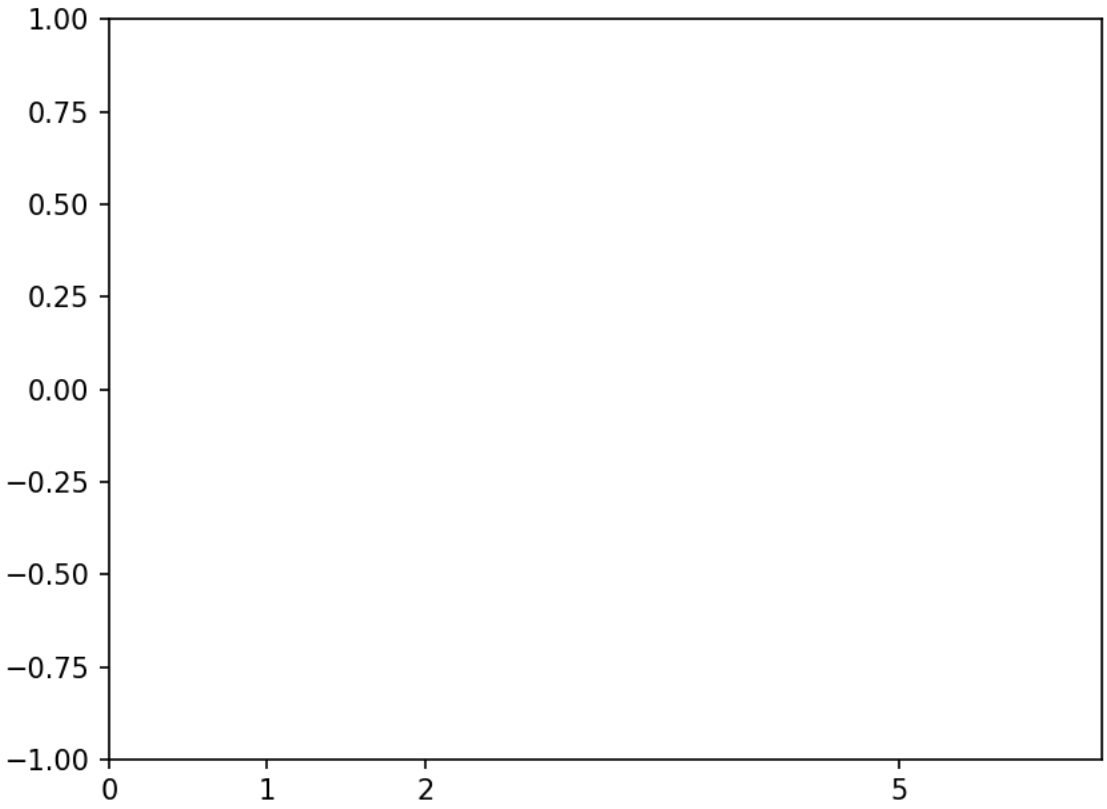


Out[11]:
```
[<matplotlib.lines.Line2D at 0x7fbf45127f70>]
```

Indicando posiciones fijas para las marcas.

También podemos indicar de forma explícita la posición de las marcas que queremos. Esto lo hacemos con un `FixedLocator()`. Veamos un ejemplo:

In [12]:
```
fig = plt.figure()
ax = plt.axes()
ax.set_xlim(0, 2.0*np.pi)
ax.set_ylim(-1, 1)
ax.xaxis.set_major_locator(mpl.ticker.FixedLocator([0, 1, 2, 5]))
ax.plot(x, np.sin(x))
```

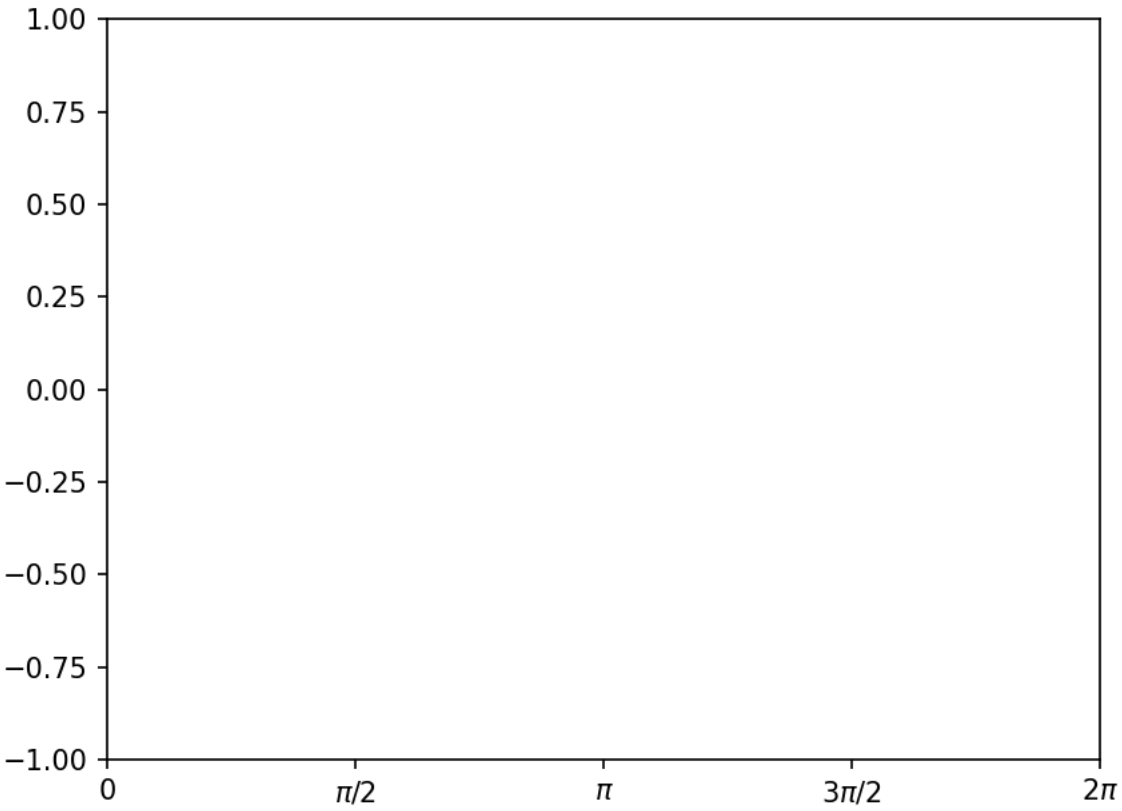


Out[12]:
```
[<matplotlib.lines.Line2D at 0x7fbf4506da30>]
```

Asignando etiquetas a posiciones.

También podemos asignar etiquetas concretas a las posiciones fijas de las marcas con `FixedFormatter()`.

In [13]:
```
fig = plt.figure()
ax = plt.axes()
ax.set_xlim(0, 2.0*np.pi)
ax.set_ylim(-1, 1)
ax.xaxis.set_major_locator(mpl.ticker.FixedLocator(
    np.arange(5)*np.pi/2.0))
ax.xaxis.set_major_formatter(mpl.ticker.FixedFormatter(
    ['0', '$\pi/2$', '$\pi$', '$3\pi/2$', '$2\pi$']))
ax.plot(x, np.sin(x))
```

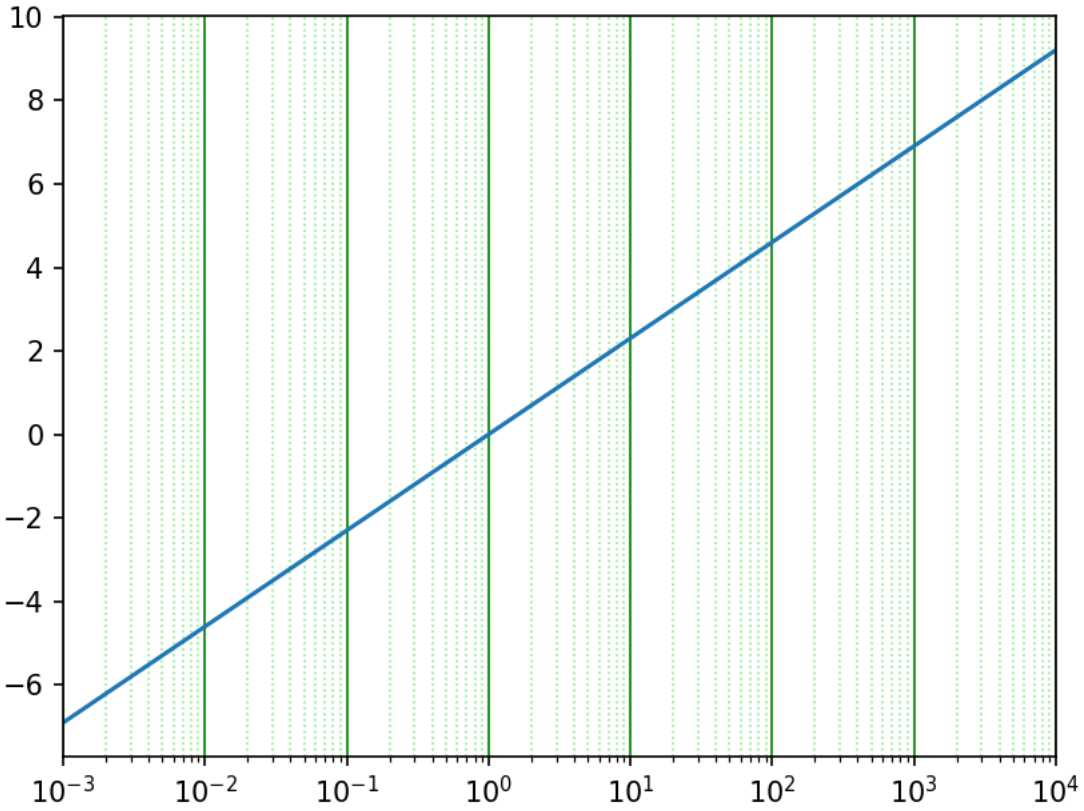


Out[13]:
```
[<matplotlib.lines.Line2D at 0x7fbf45033bb0>]
```

Escala logarítmica.

Hasta ahora la escala usada para mostrar los ejes ha sido lineal. Para activar una escala logarítmica basta con indicarlo con los métodos `Axes.set_xscale()` o `Axes.set_yscale()`. Estos métodos son equivalentes a indicar usar los objetos `LogLocator` y `LogFormatterSciNotation`.

In [14]:
```
x=np.linspace(0.001, 10000.0, 1000)
fig = plt.figure()
ax = plt.axes()
ax.set_xscale('log')
ax.grid(axis='x', which='major', color='green')
ax.grid(axis='x', which='minor', color='lightgreen', linestyle=':')
ax.plot(x, np.log(x))
ax.set_xlim(0.001, 10000.0)
print(ax.xaxis.get_major_locator())
print(ax.xaxis.get_major_formatter())
print(ax.xaxis.get_minor_locator())
print(ax.xaxis.get_minor_formatter())
```




```
<matplotlib.ticker.LogLocator object at 0x7fbf4504f340>
<matplotlib.ticker.LogFormatterSciNotation object at 0x7fbf4504a340>
<matplotlib.ticker.LogLocator object at 0x7fbf4504acd0>
<matplotlib.ticker.LogFormatterSciNotation object at 0x7fbf4504af70>
```

Activando marcas temporales.

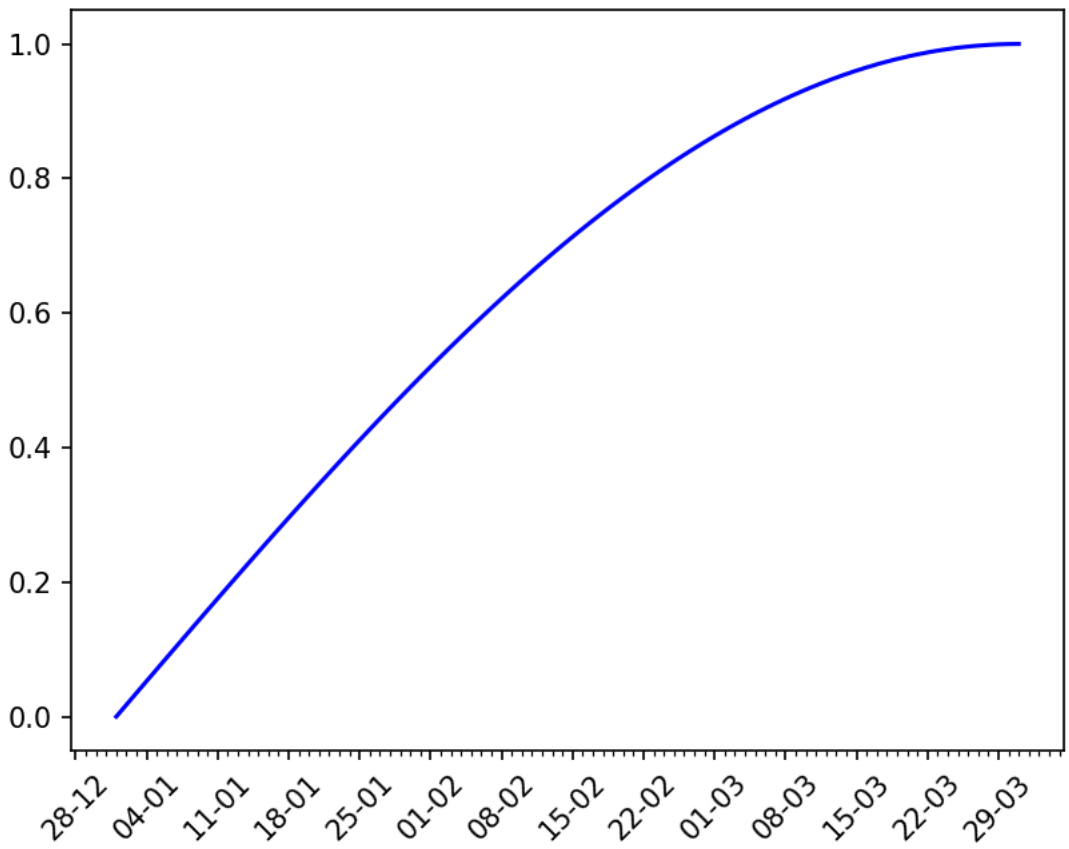
Cuando uno de los ejes maneja datos temporales, podemos indicarlo con `Axes.xaxis_date()` `Axes.yaxis_date()`.

Por defecto el formato de fecha se pone como 'yy-mm-dd'. Para tener un mayor control de los ticks, se puede utilizar los objetos definidos en el subpaquete `matplotlib.dates`.

Por ejemplo para localizar las marcas los días que son 'lunes' usaríamos el objeto `WeekdayLocator` y para dar el formato 'dd-mm' usamos el objeto `DateFormatter`.

A menudo las etiquetas de los ticks se sobreponen unas a las otras. Para impedir esto, podemos hacer que se dibujen rotando un ángulo dado. Para ello se usa el método `Axes.tick_params()` ajustando el parámetro `labelrotation`.

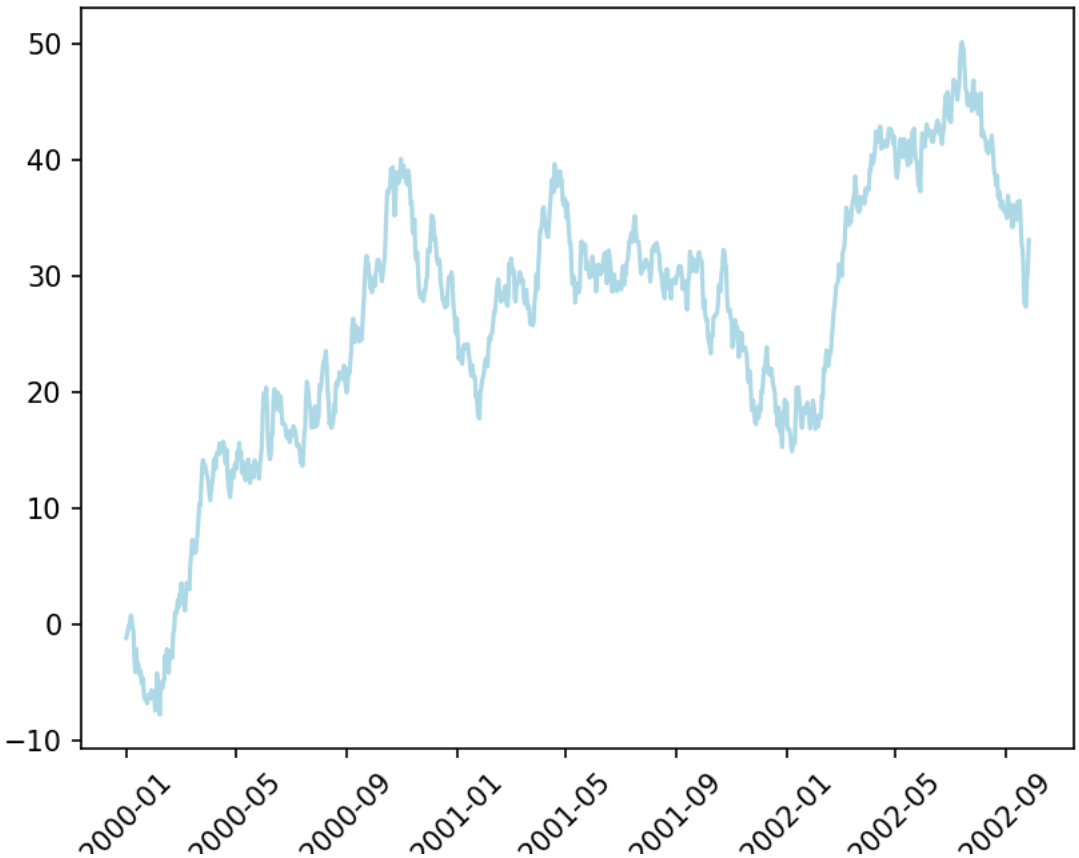
In [15]:
```
x=np.linspace(0, np.pi/2.0, 90)
x_date=pd.date_range('2021-01-01', periods=90, freq='D')
y=np.sin(x)
fig=plt.figure()
ax = plt.axes()
ax.xaxis_date()
ax.xaxis.set_major_locator(mpl.dates.WeekdayLocator(byweekday=mpl.dates.MO))
ax.xaxis.set_major_formatter(mpl.dates.DateFormatter("%d-%m"))
ax.tick_params(axis='x', labelrotation = 45)
ax.xaxis.set_minor_locator(mpl.dates.DayLocator())
ax.plot(x_date, y, 'b-')
```



Out[15]: `[<matplotlib.lines.Line2D at 0x7fbf44fcd070>]`

Si los datos temporales a utilizar como para las marcas de los ejes provienen de un objeto pandas.Series o pandas.DataFrame usando los tipos Pandas para definir fechas y tiempo, se recomienda utilizar los "formateadores" que proporciona Pandas ejecutando el método `pandas.plotting.register_matplotlib_converters()`).

In [16]:
```
pd.plotting.register_matplotlib_converters()
ts = pd.Series(np.random.randn(1000),
               index=pd.date_range("1/1/2000", periods=1000))
ts = ts.cumsum()
fig = plt.figure()
ax = plt.axes()
ax.xaxis_date()
ax.tick_params(axis='x', labelrotation = 45)
ax.plot(ts.index, ts.to_numpy(), linestyle='-', color='lightblue')
```



Out[16]: [<matplotlib.lines.Line2D at 0x7fbf44f4c070>]

Fijando un estilo general a los gráficos.

Por estilo entendemos como un conjunto general de parámetros con unos valores por defecto que ofrecen una aspecto determinado a nuestros gráficos. Hasta ahora estamos usando el estilo clásico '*classic*' pero hay muchos otros estilos que podemos usar. Podemos ver una lista de estilos disponibles (esto puede cambiar dependiendo de la versión y entorno de ejecución) ejecutando:

In [17]:
```
print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid', 'bmh',
'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seabor
n', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'sea
born-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 's
eaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'sea
born-whitegrid', 'tableau-colorblind10']
```

Para los ejemplos siguientes, vamos a crear una función que dibuje una figura con dos gráficos: un histograma y un gráfico de líneas.

In [18]:
```
def hist_and_lines():
    np.random.seed(1) #para que siempre se genere la misma figura.
    fig, axes = plt.subplots(1, 2, figsize=(8, 4))
    axes[0].grid(True)
    axes[0].set_title('Un histograma')
    axes[0].hist(np.random.randn(1000), bins=10, histtype='bar')
    axes[1].set_title('Unas líneas')
    for i in range(5):
```

```
        axes[1].plot(np.random.rand(5), label=str(i))
    axes[1].legend(loc='best')
```

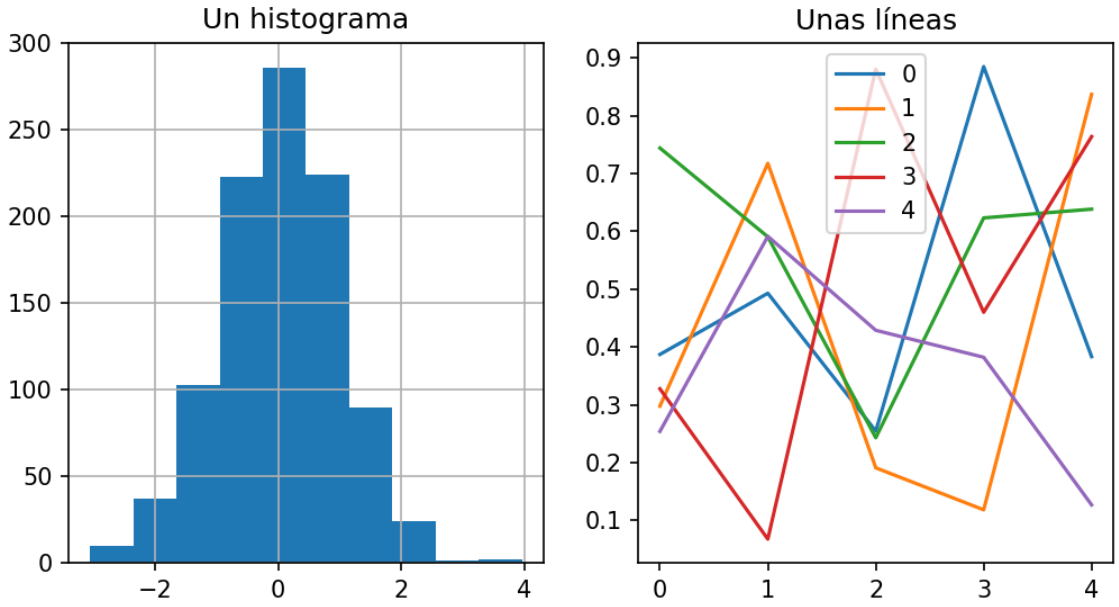
Para fijar un estilo de forma general para una sesión utilizamos una vez `plt.style.use('estilo-a-usar')`. De esta forma los cambios afectan a toda la sesión.

Otra forma alternativa es utilizar un contexto temporal, de forma que los parámetros fijados se restauran a sus valores originales antes de crear el contexto cuando este ya no sea necesario. Esto se hace usando el método plt.style.context('estilo-a-usar')

Para nuestros ejemplos usaremos un contexto temporal, sin embargo, en general, lo normal es fijar el estilo deseado al principio de la sesión/script una vez importando el módulo matplotlib.pyplot.

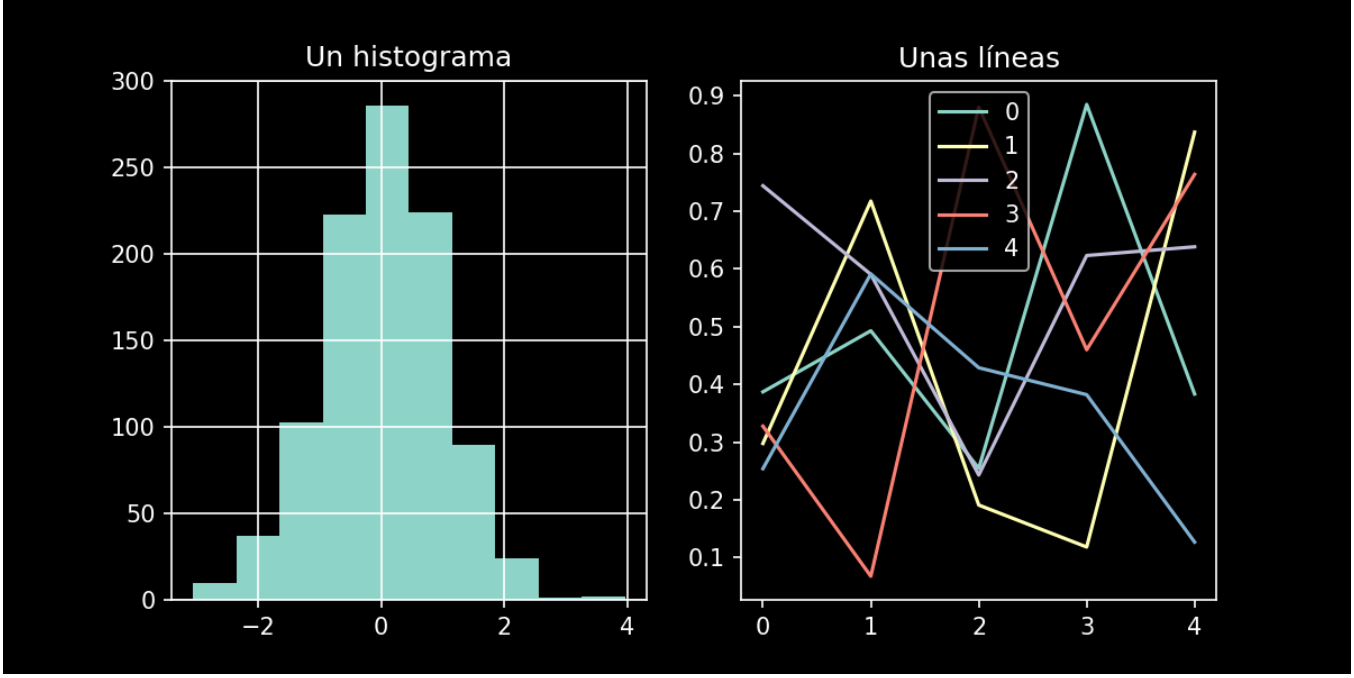
Veamos primero cómo se vería nuestro gráfico con el estilo por defecto del sistema (normalmente 'classic').

In [19]:
```
plt.rcdefaults() #Reset los parámetros por defecto del sistema
hist_and_lines()
```



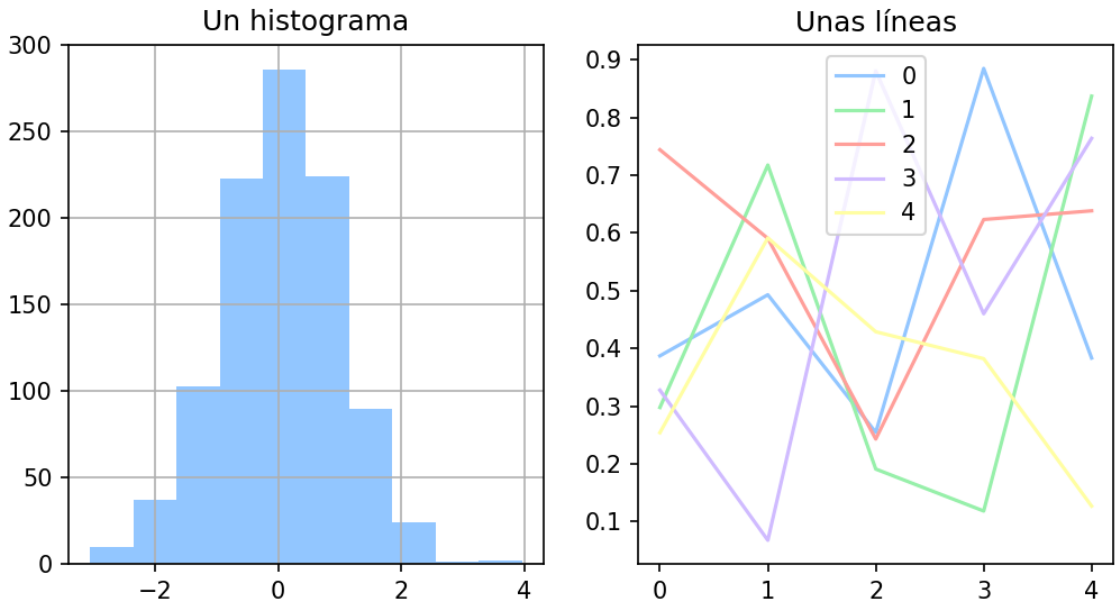
Veamos qué pasa si utilizamos el estilo 'dark-background'

In [20]:
```
with plt.style.context('dark_background'):
    hist_and_lines()
```



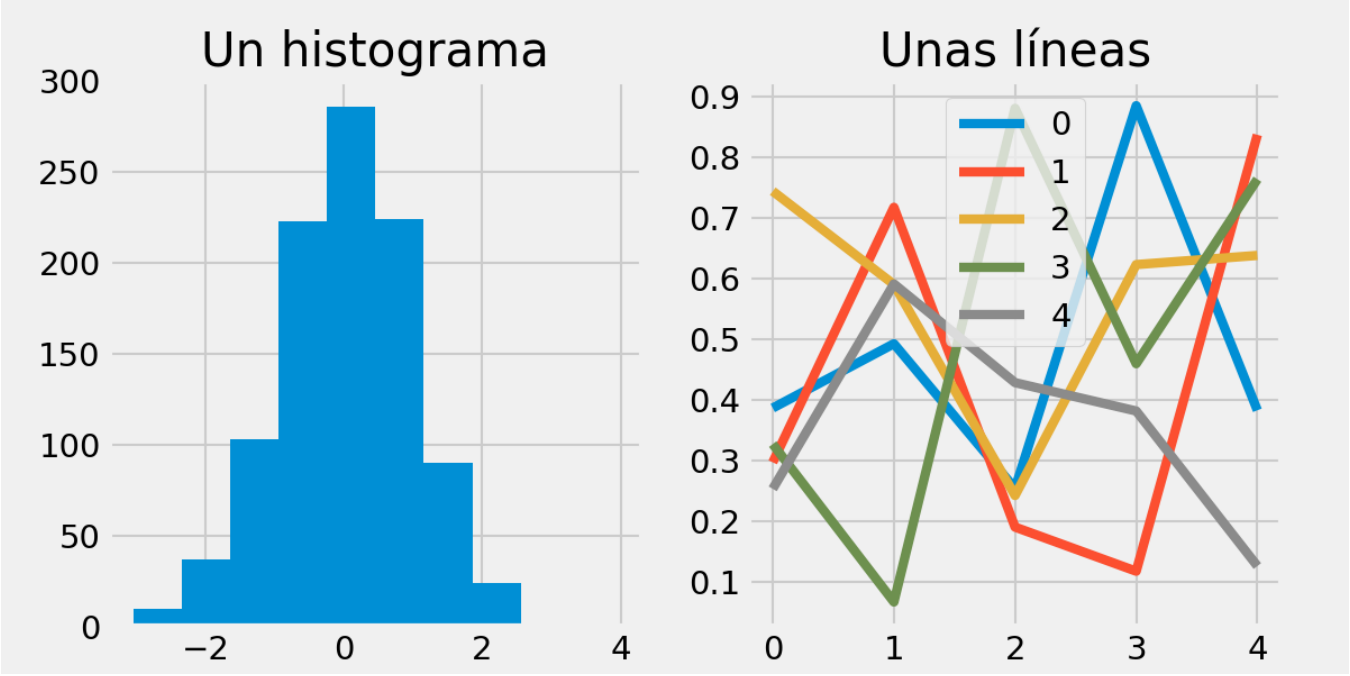
Veamos ahora el aspecto usando el estilo 'seaborn-pastel'

In [21]:
```
with plt.style.context('seaborn-pastel'):
    hist_and_lines()
```



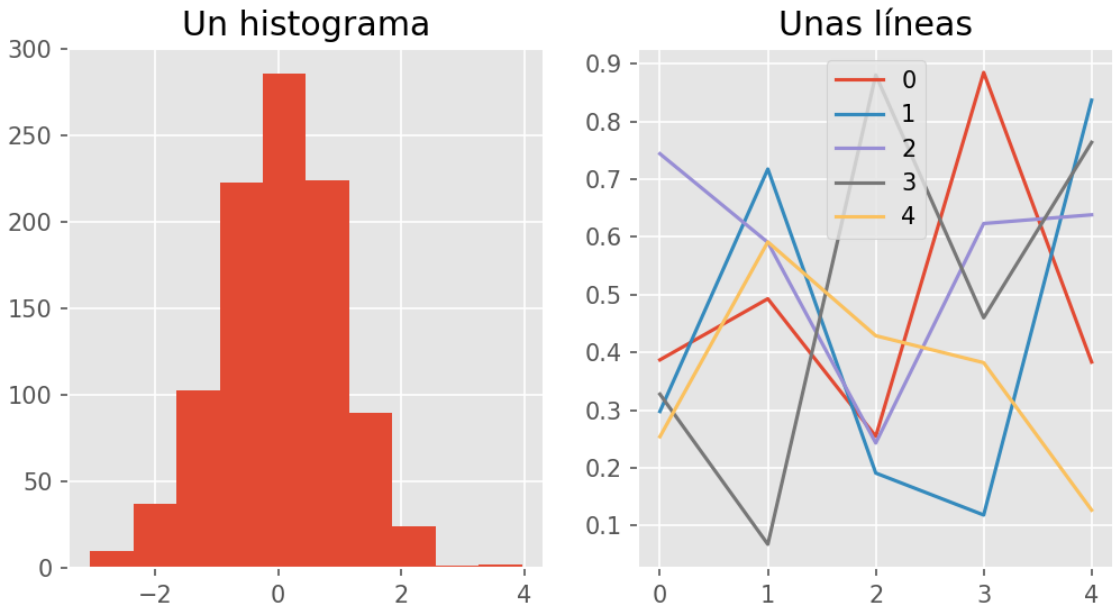
Otro estilo es 'fivethirtyeight', veamos cómo queda:

In [22]:
```
with plt.style.context('fivethirtyeight'):
    hist_and_lines()
```



Ggplot es un paquete usando con lenguaje R para visualizar datos estadísticos y de uso muy extendido. Podemos usar el estilo 'gglot' para obtener gráficos con un estilo similar.

In [23]:
```
with plt.style.context('ggplot'):
    hist_and_lines()
```



Además de seleccionar una configuración general podemos fijar parámetros concretos modificando el atributo `mpl.rcParams` para ajustarlos a nuestros gustos. Para esto se recomienda leer la referencia oficial.

Por ejemplo para generar figuras con mayor tamaño y densidad de la normal usamos los atributos `figure.dpi` y `figure.figsize`.

In [24]:
```
mpl.rcParams['figure.dpi'] = 300
mpl.rcParams['figure.figsize'] = (11.0+3.0/4.0, 8.0+1.0/4.0) #A4 landscape.
```

Generando gráficos desde Pandas

Pandas ofrece el módulo `pandas.plotting` para dibujar gráficos en combinación con `Matplotlib`.

Se recomienda leer esta referencia para ver los distintos tipos de gráficos proporcionados.

In [ ]: