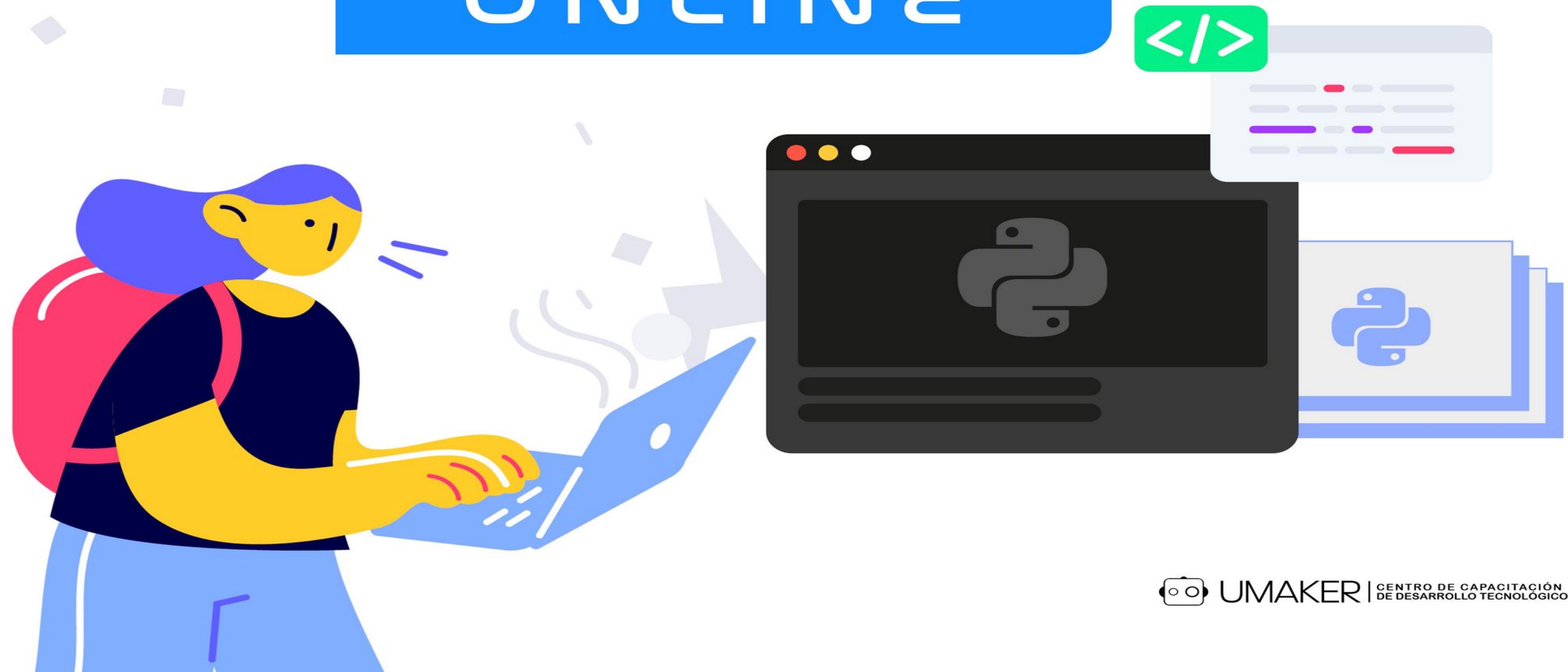


MASTER PYTHON

ONLINE

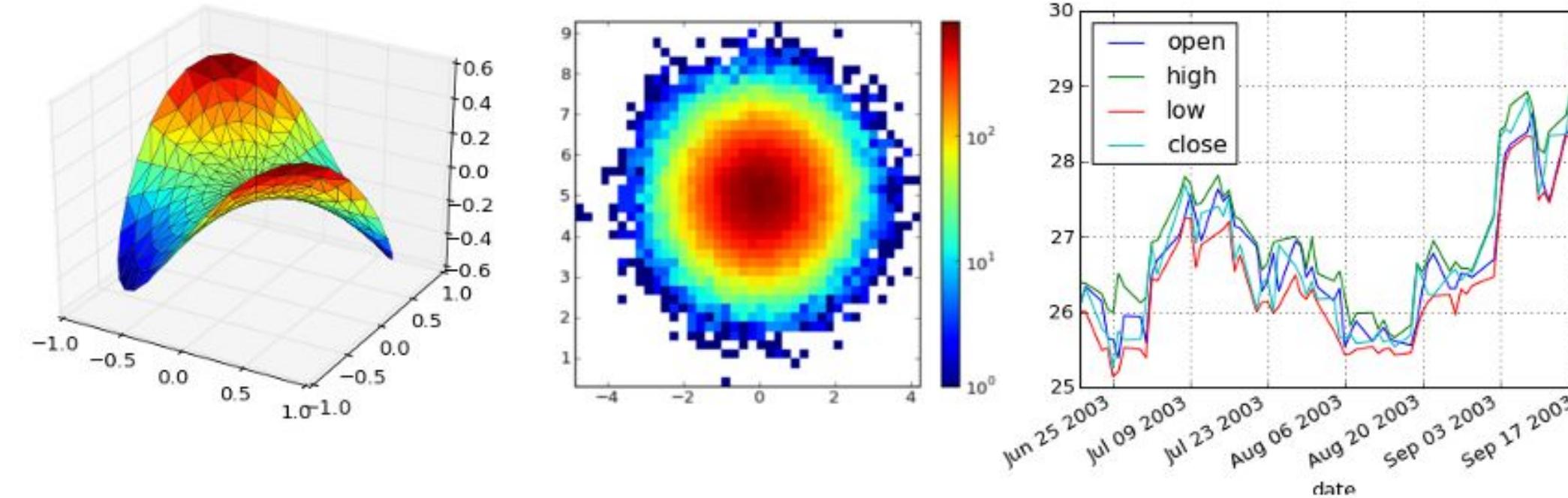




matplotlib

Matplotlib: visualización con Python

Matplotlib es una biblioteca completa para crear visualizaciones estáticas, animadas e interactivas en Python.



Características de Matplotlib

1. Crear

Gráficos con unas pocas líneas de código

Figuras interactivas que pueden hacer zoom, desplazarse, actualizar .

2. Personalizar

Control total de estilos de línea, propiedades de fuente, propiedades de ejes,etc.

Exportar e incrustar en varios formatos de archivo y entornos interactivos

3. Ampliar

Explora la funcionalidad personalizada proporcionada por paquetes de terceros

Extensa documentación de Matplotlib



Dependencias

Matplotlib requiere las siguientes dependencias:

- Python (> = 3.6)
- FreeType (> = 2.3)
- libpng (> = 1.2)
- NumPy (> = 1.11)
- herramientas de instalación
- cycladora (> = 0.10.0)
- dateutil (> = 2.1)
- kiwisolver (> = 1.0.0)
- pyparsing

Instalación

```
python -m pip install -U matplotlib
```

Nota

Para el soporte de otros marcos de GUI, renderizado de LaTeX, guardar animaciones y una mayor selección de formatos de archivo, es posible que necesite instalar dependencias adicionales .

Opcionalmente, también puede instalar una serie de paquetes para habilitar mejores kits de herramientas de interfaz de usuario.

Tk (> = 8.3,! = 8.6.0 o 8.6.1): para los backends basados en Tk;
PyQt4 (> = 4.6) o PySide (> = 1.0.3) [1] : para los backends basados en Qt4;
PyQt5 : para los backends basados en Qt5;
PyGObject : para los backends basados en GTK3 [2] ;
wxPython (> = 4) [3] : para los backends basados en wx;
cairoffi (> = 0.8) o pycairo : para los backends basados en el cairo;
Tornado : para el backend de WebAgg;



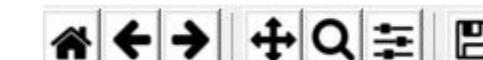
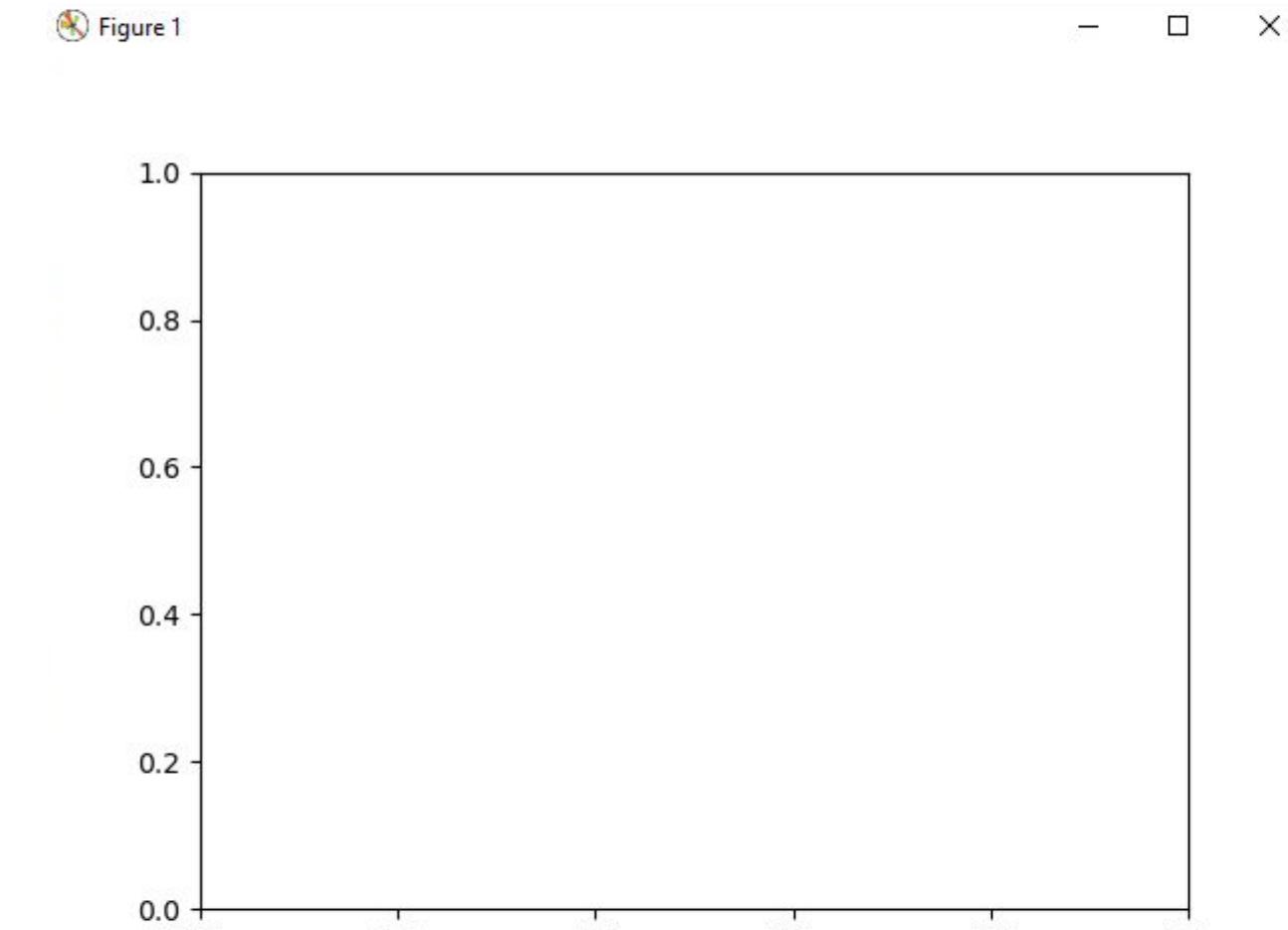
Iniciando

LA FUNCIÓN SUBPLOTS

La función `matplotlib.pyplot.subplots` crea una figura y uno (o varios) conjunto de ejes, devolviendo una referencia a la figura y a los ejes. Por defecto -si no se especifica otra cosa- crea un único conjunto de ejes:

```
import matplotlib.pyplot as plt  
fig,ax = plt.subplots()  
plt.show()
```

```
print(fig) # imprime Figure(640x480)  
print(ax) # imprime AxesSubplot(0.125,0.11;0.775x0.77)
```



Iniciando

LA FUNCIÓN SUBPLOTS

Si queremos crear un conjunto de matrices, por ejemplo, 2 filas y 3 columnas (es decir, 6 conjuntos), basta añadir estos valores como primeros argumentos de la función.

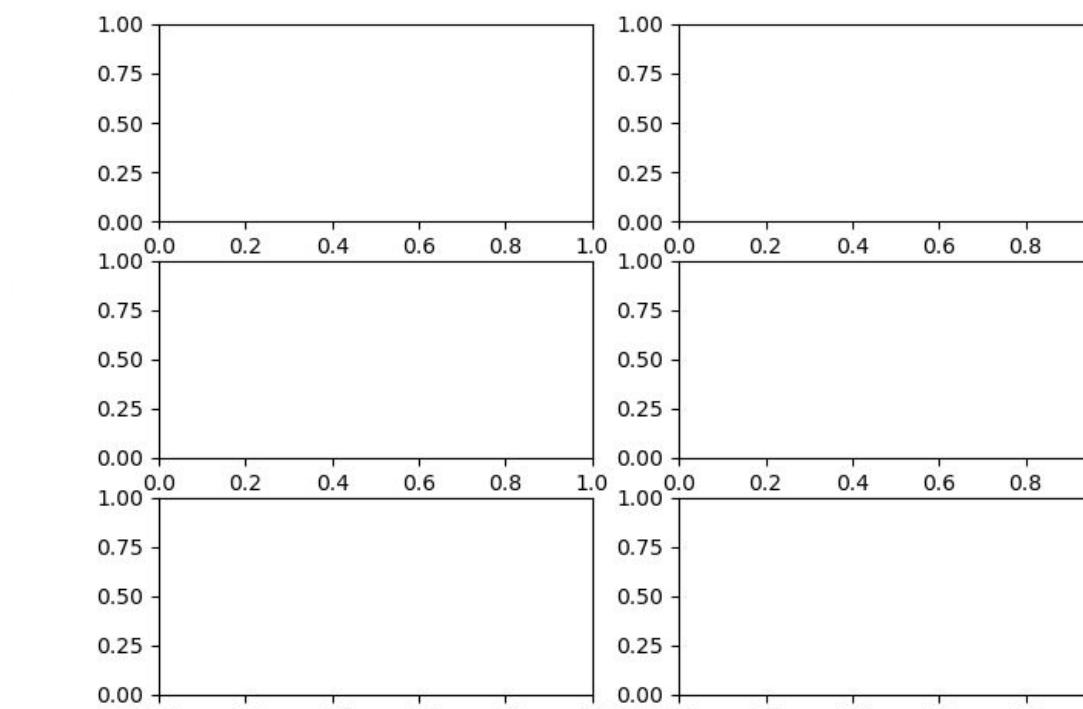
```
import matplotlib.pyplot as plt  
  
fig,ax = plt.subplots(3,2)  
fig.set_size_inches(8,6)  
plt.show()
```

El Método `set_size_inches(x,y)` redefine el tamaño de ventana ,además la referencia a la ventana pasa a ser una lista de objetos indicando su dirección de memoria y los ejes "ax" un array de numpy.

```
print(fig)  
print(ax)  
print(type(fig))  
print(type(ax))
```



```
Figure(800x600)  
[[<matplotlib.axes._subplots.AxesSubplot object at 0x000001721D671DC8>  
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001721FAD4A88>]  
 [<matplotlib.axes._subplots.AxesSubplot object at 0x000001721FB0EB88>  
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001721FB46D48>]  
 [<matplotlib.axes._subplots.AxesSubplot object at 0x000001721FB7DF08>  
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001721FBBE208>]]  
<class 'matplotlib.figure.Figure'>  
<class 'numpy.ndarray'>
```



Iniciando

LA FUNCIÓN SUBPLOTS

Ahora podríamos ejecutar el método plot asociado a cada uno de estos ejes para mostrar una gráfica. Por ejemplo, si quisiéramos mostrarla en la segunda fila (cuyo índice es 1) y primera columna (cuyo índice es 0), podríamos hacerlo del siguiente modo:

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

y = np.random.randn(100).cumsum()

fig,ax = plt.subplots(3,2)
fig.set_size_inches(8,6)
ax[1,0].plot(y) #seleccionamos la fila 1 , columna 0

plt.show()
```

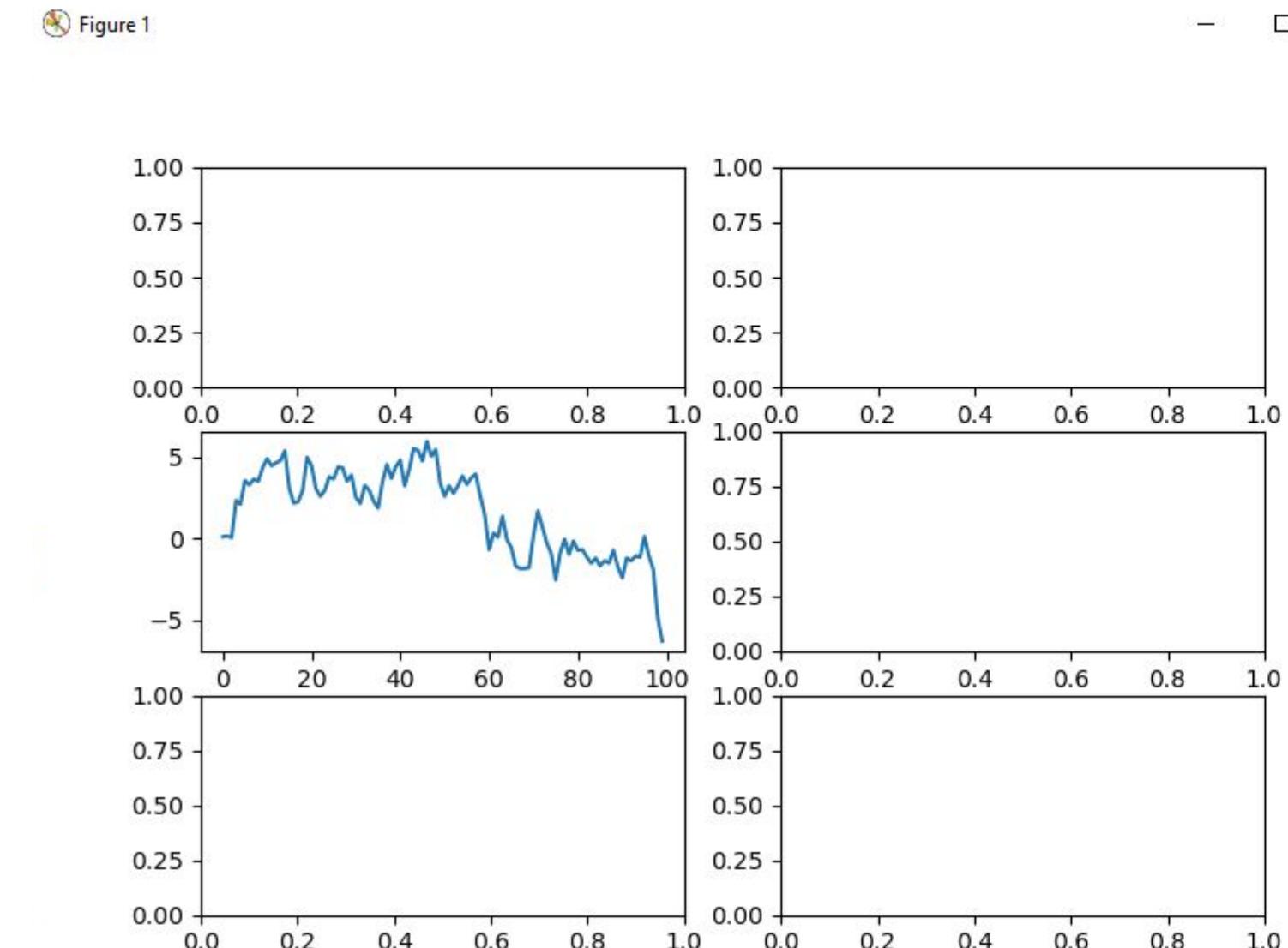
Las dos instrucciones son equivalentes :

fig,ax = plt.subplot()

fig,ax = plt.subplots(1,1)

Es decir, en el segundo caso no se devuelve una lista formada por un único elemento, sino el elemento directamente.

Método **cumsum ()** Devuelve la suma acumulativa de los elementos a lo largo de un eje dado

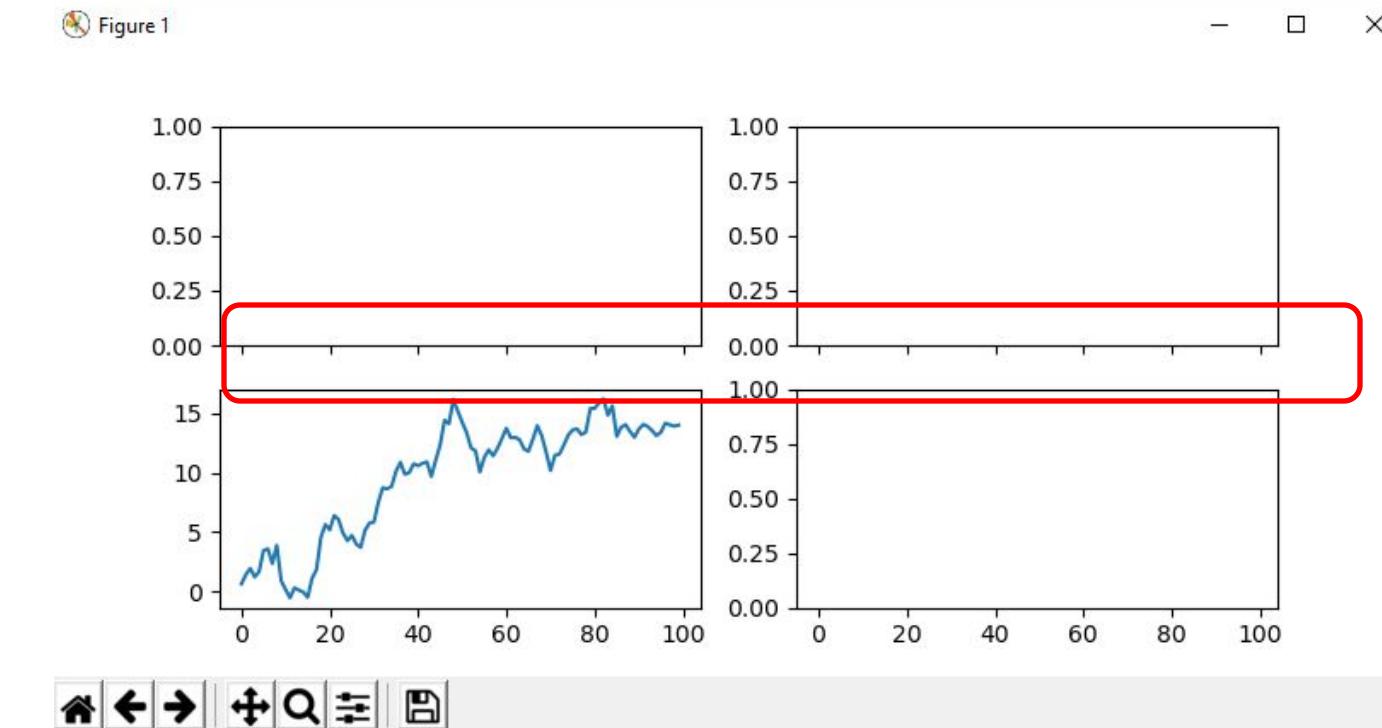


Iniciando

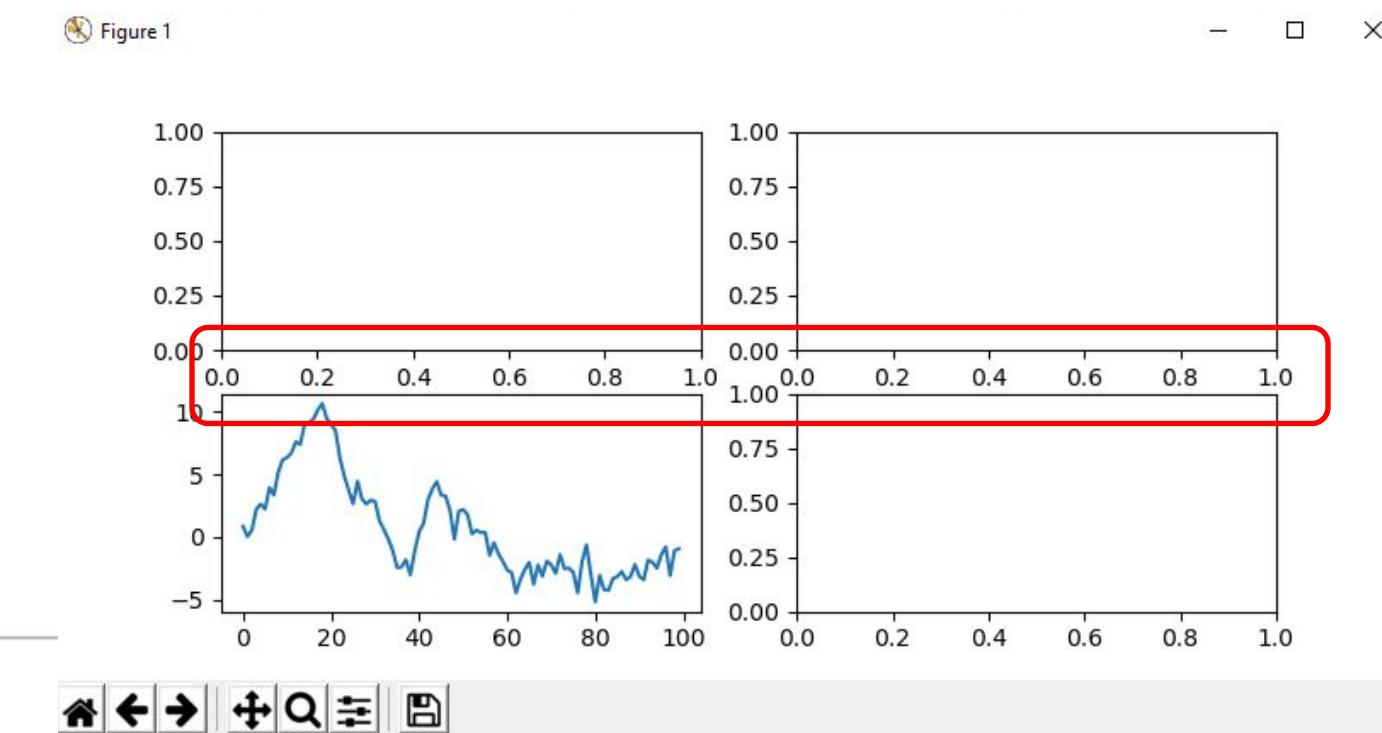
LA FUNCIÓN SUBPLOTS

Dos parámetros interesantes de la función subplots son **sharex** y **sharey**. Éstos controlan la compartición de las propiedades de los ejes. Por defecto toman el valor *False*, lo que supone que cada conjunto de ejes es independiente. Si, por ejemplo, el argumento *sharex* se fija a *True*, todos los ejes x de los diferentes conjuntos de ejes compartirán las mismas propiedades. Veámoslo en la práctica con una matriz de 2x2 conjuntos de ejes:

```
import numpy as np
import matplotlib.pyplot as plt
y = np.random.randn(100).cumsum()
fig,ax = plt.subplots(2,2,sharex = True)
fig.set_size_inches(8,4)
ax[1,0].plot(y)
plt.show()
```



```
import numpy as np
import matplotlib.pyplot as plt
y = np.random.randn(100).cumsum()
fig,ax = plt.subplots(2,2,sharex = False)
fig.set_size_inches(8,4)
ax[1,0].plot(y)
plt.show()
```



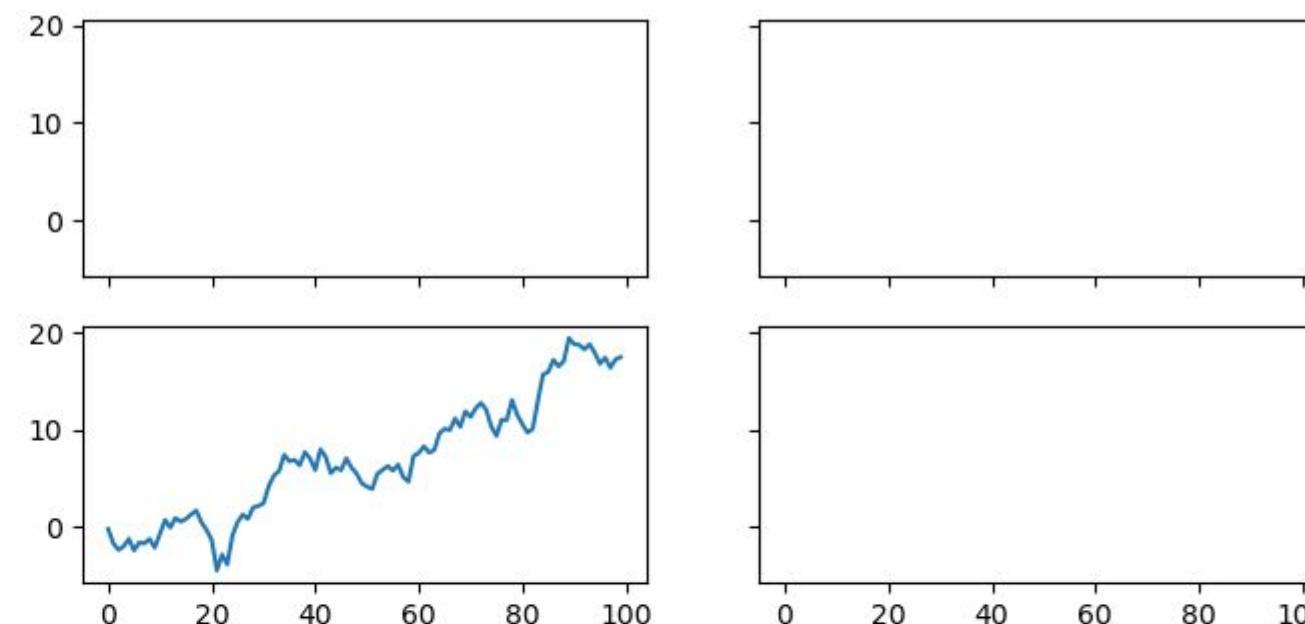
Iniciando

LA FUNCIÓN SUBPLOTS

Vemos ahora que los ejes son mostrados en la primera columna y en la última fila, indicando que el resto de los conjuntos de ejes comparten dichos ejes.

```
import numpy as np
import matplotlib.pyplot as plt
y = np.random.randn(100).cumsum()
fig,ax = plt.subplots(2,2,sharex = True , sharey = True)
fig.set_size_inches(8,4)
ax[1,0].plot(y)
plt.show()
```

Figure 1

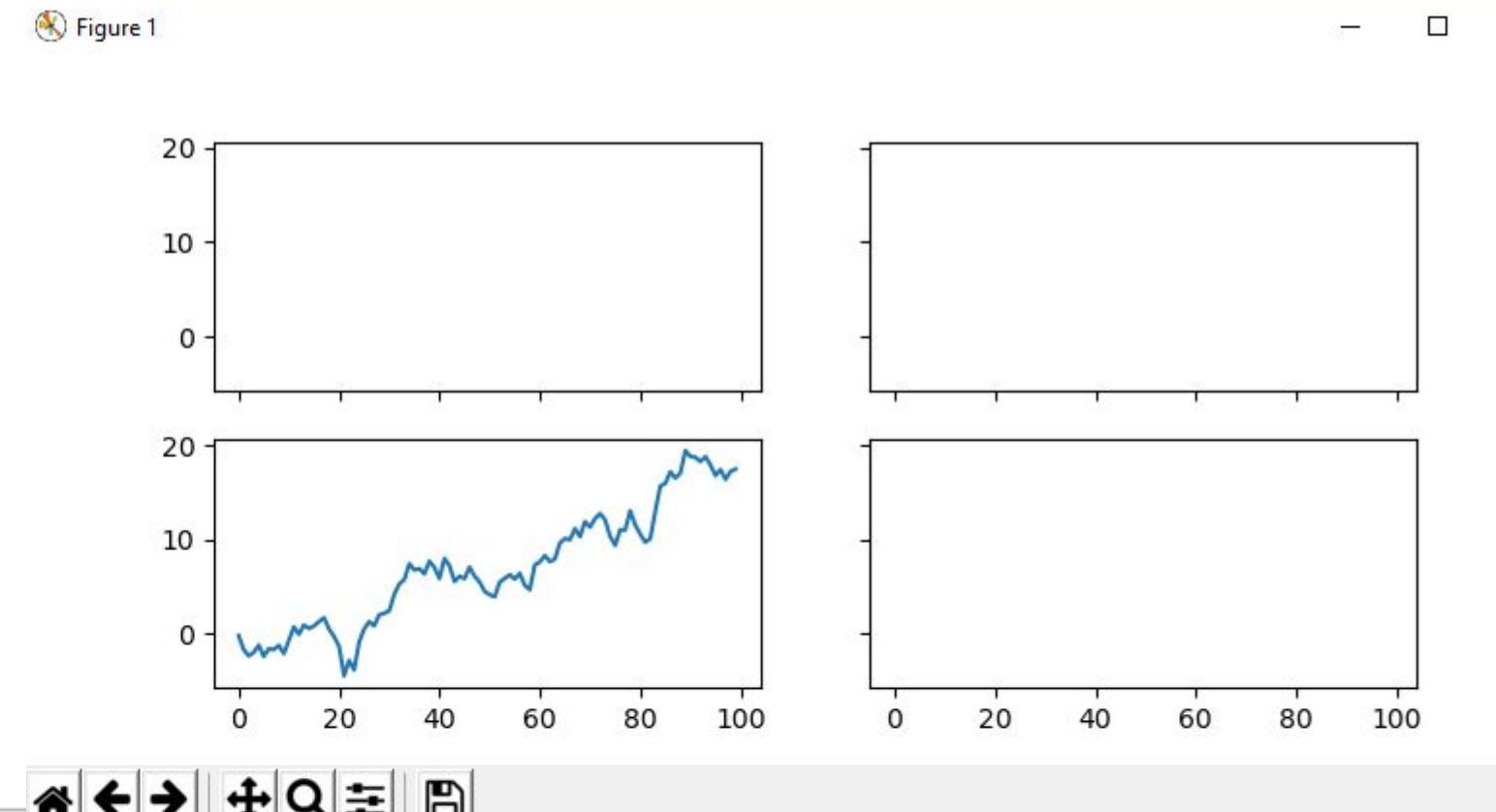


Iniciando

LA FUNCIÓN SUBPLOTS

Vemos ahora que los ejes son mostrados en la primera columna y en la última fila, indicando que el resto de los conjuntos de ejes comparten dichos ejes.

```
import numpy as np
import matplotlib.pyplot as plt
y = np.random.randn(100).cumsum()
fig,ax = plt.subplots(2,2,sharex = True , sharey = True)
fig.set_size_inches(8,4)
ax[1,0].plot(y)
plt.show()
```



Iniciando

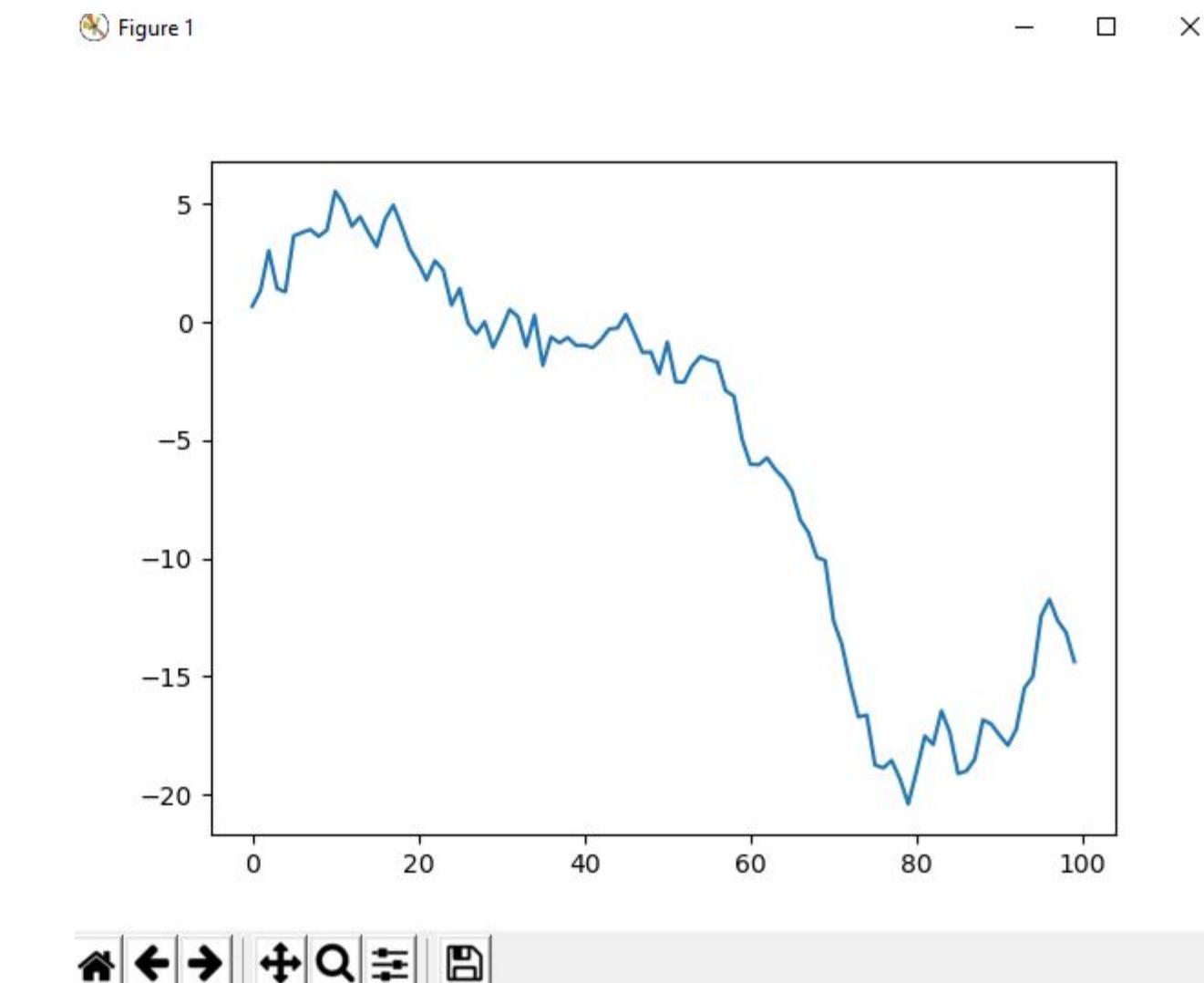
LA FUNCIÓN AXES

La función `matplotlib.pyplot.axes` es la segunda alternativa que vamos a ver. Esta función añade un conjunto de ejes en la figura actual, y fija el nuevo conjunto de ejes como actual (o, con otras palabras, lo marca como "activo"). Si no existe una figura, la crea y la marca como actual

```
import numpy as np
import matplotlib.pyplot as plt
y = np.random.randn(100).cumsum()

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

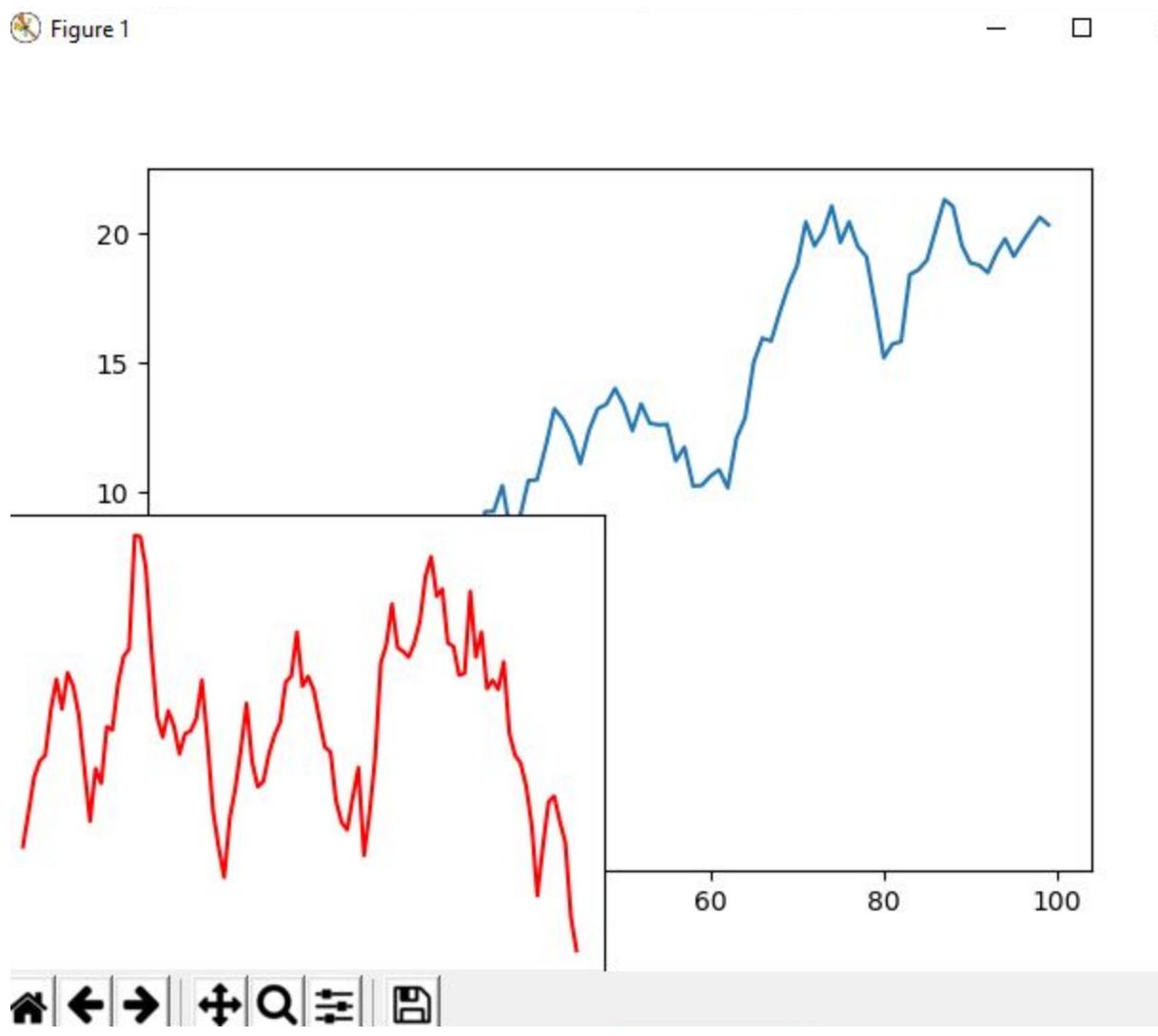
print(fig)
print(ax)
ax.plot(y)
plt.show()
```



Iniciando

Para fijar la posición y tamaño de un eje podemos pasar una tupla a la función como primer argumento. Esta tupla contendrá la siguiente información:

[Posición izquierda, posición derecha, ancho, alto]



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

y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()

fig = plt.figure()

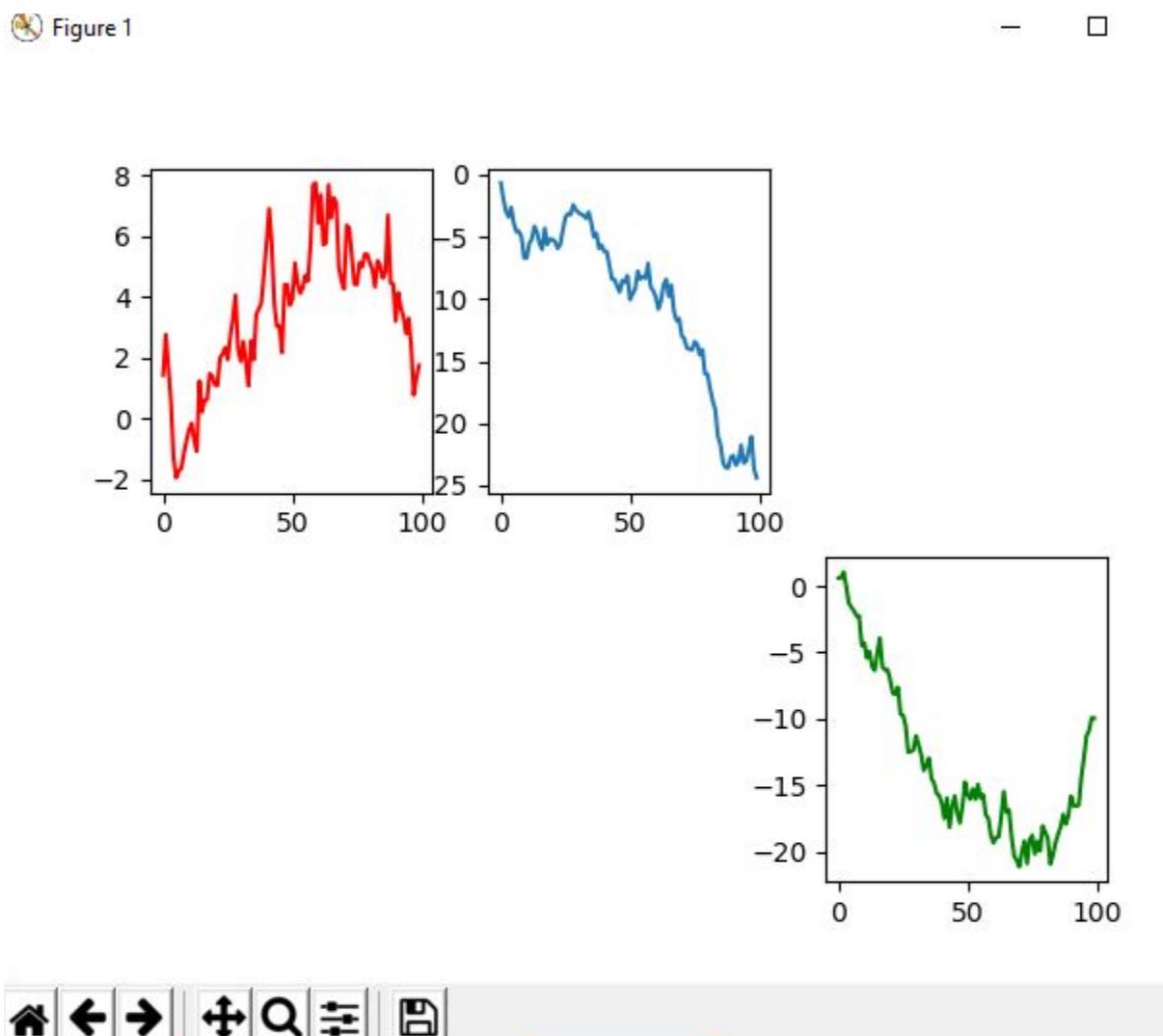
ax1 = plt.axes()
ax1.plot(y1)

ax2 = plt.axes([0,0,0.5,0.5])
ax2.plot(y2,color="red")

plt.show()
```

LA FUNCIÓN ADD_SUBPLOT

El método [add_subplot](#) asociado a una figura añade un conjunto de ejes a la misma pero suponiéndolo en una matriz de columnas y filas (de conjuntos de ejes) y creándolo en la posición indicada de dicha matriz.



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

y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()
y3 = np.random.randn(100).cumsum()

fig = plt.figure() 2 filas 3 columnas
fig.add_subplot(2,3,2)
plt.plot(y1)
fig.add_subplot(2,3,1)
plt.plot(y2, color = "red")
fig.add_subplot(2,3,6)
plt.plot(y3,color = "green")
plt.show()
```

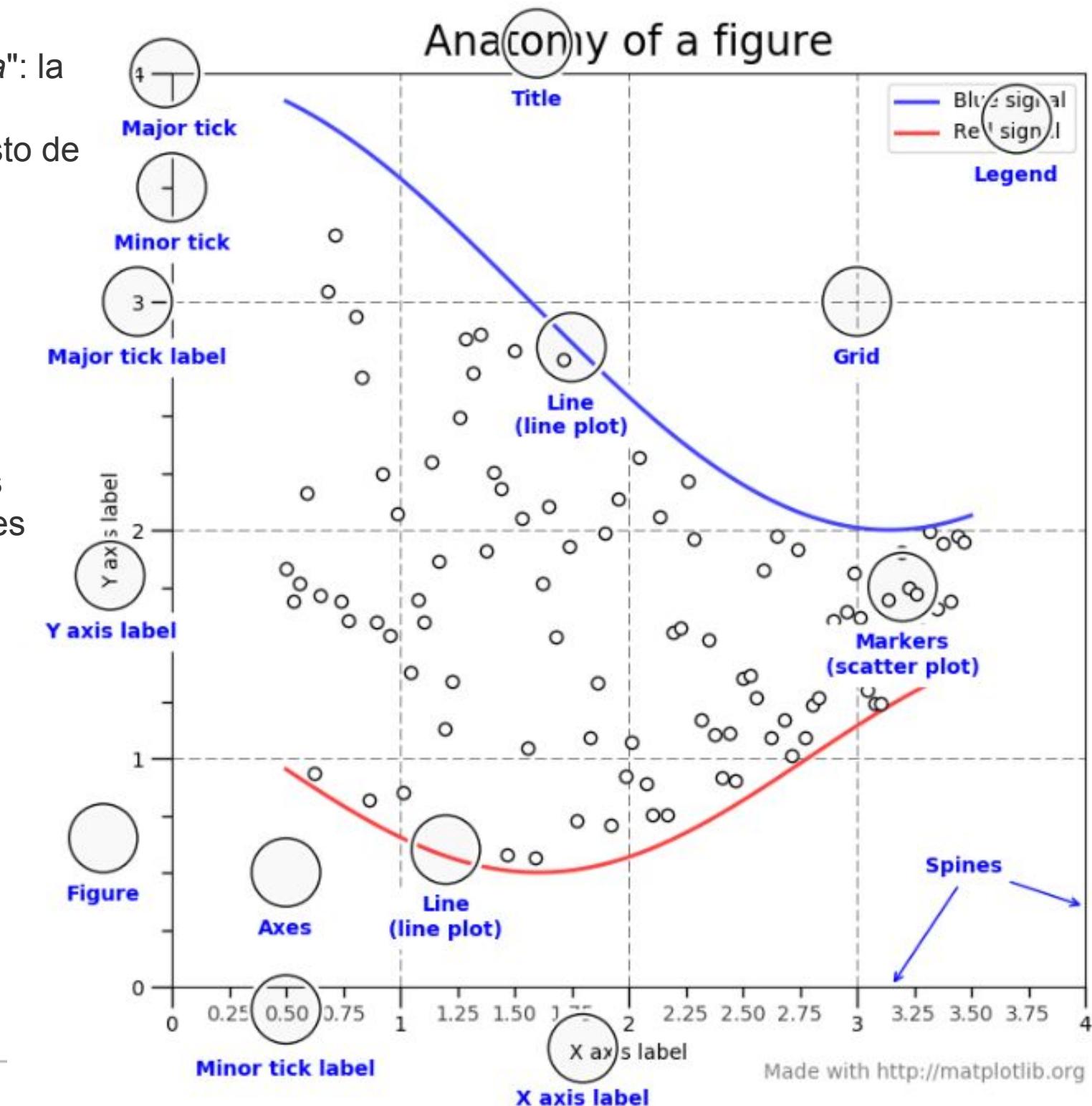
PERSONALIZACIÓN DE EJES

Identificamos en ella un elemento que depende de lo que llamamos "gráfica": la línea (*line*) -y su color, ancho, estilo...-.

Otro de los elementos mostrados (la leyenda) depende de la figura. Y el resto de los elementos dependen directamente del conjunto de ejes:

- **Title**: título a mostrar en la gráfica
 - **x axis label**: nombre del eje x
 - **y axis label**: nombre del eje y
 - **Major ticks**: marcas principales en los ejes
 - **Minor ticks**: marcas secundarios en los ejes
 - **Major tick labels**: etiquetas a mostrar en las marcas principales de los ejes
 - **Minor tick labels**: etiquetas a mostrar en las marcas secundarios de los ejes
 - **Spines**: bordes de la gráfica
 - **Grid**: rejilla a mostrar sobre la gráfica

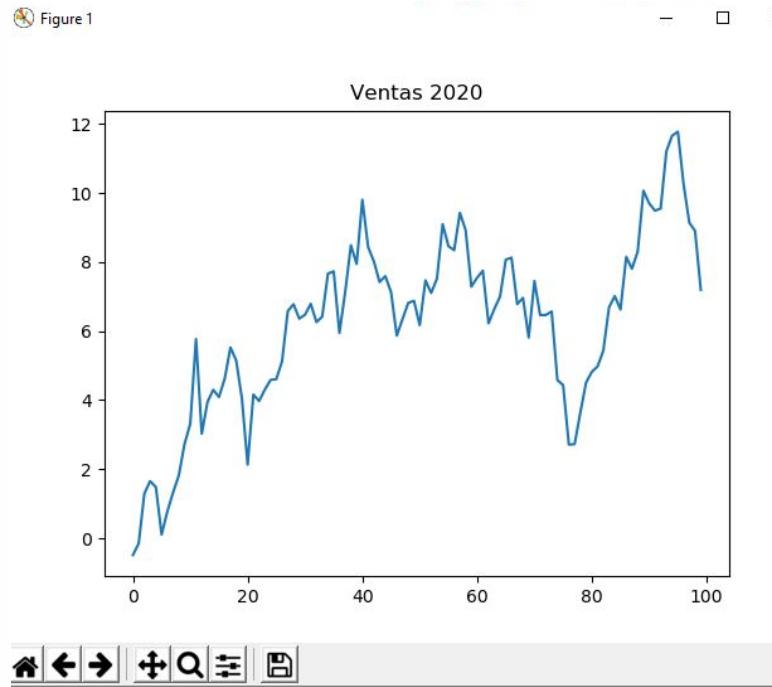
Veamos en las siguientes páginas cómo personalizar todos ellos



Iniciando

PERSONALIZACIÓN DE EJES

TÍTULO

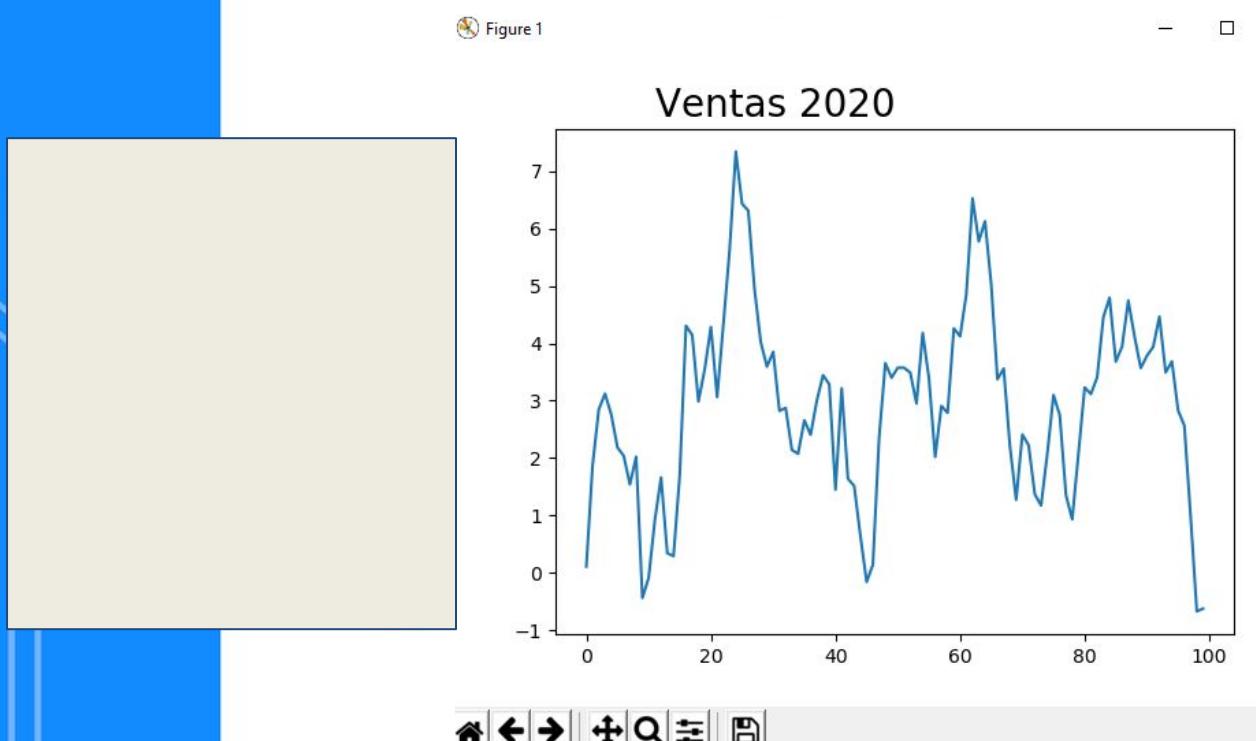


```
import matplotlib.pyplot as plt
import numpy as np

y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
plt.plot(y)
plt.title("Ventas 2020")
plt.show()
```

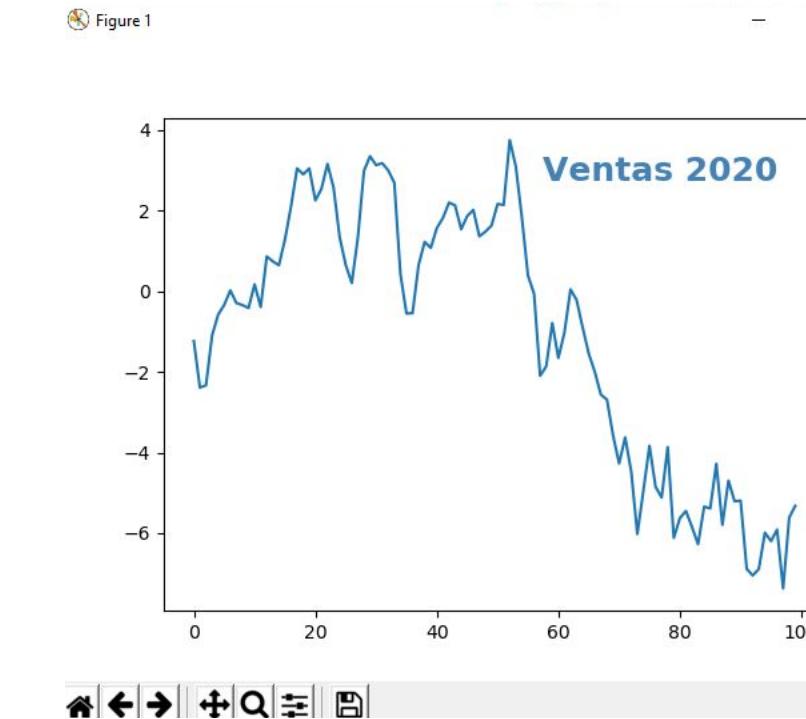
En el siguiente ejemplo especificamos una fuente tamaño 20 y alineamos el texto a la derecha (es decir, el extremo derecho del texto se muestra en el centro del área disponible):



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

y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
plt.plot(y)
plt.title("Ventas 2020", fontsize=20, horizontalalignment = "right")
plt.show()
```



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

y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
plt.plot(y)
plt.title("Ventas 2020",
          position =(0.75,0.85),
          fontsize=18,
          color = "SteelBlue",
          fontweight = "bold")

plt.show()
```

PERSONALIZACIÓN DE EJES

ETIQUETAS DE EJES

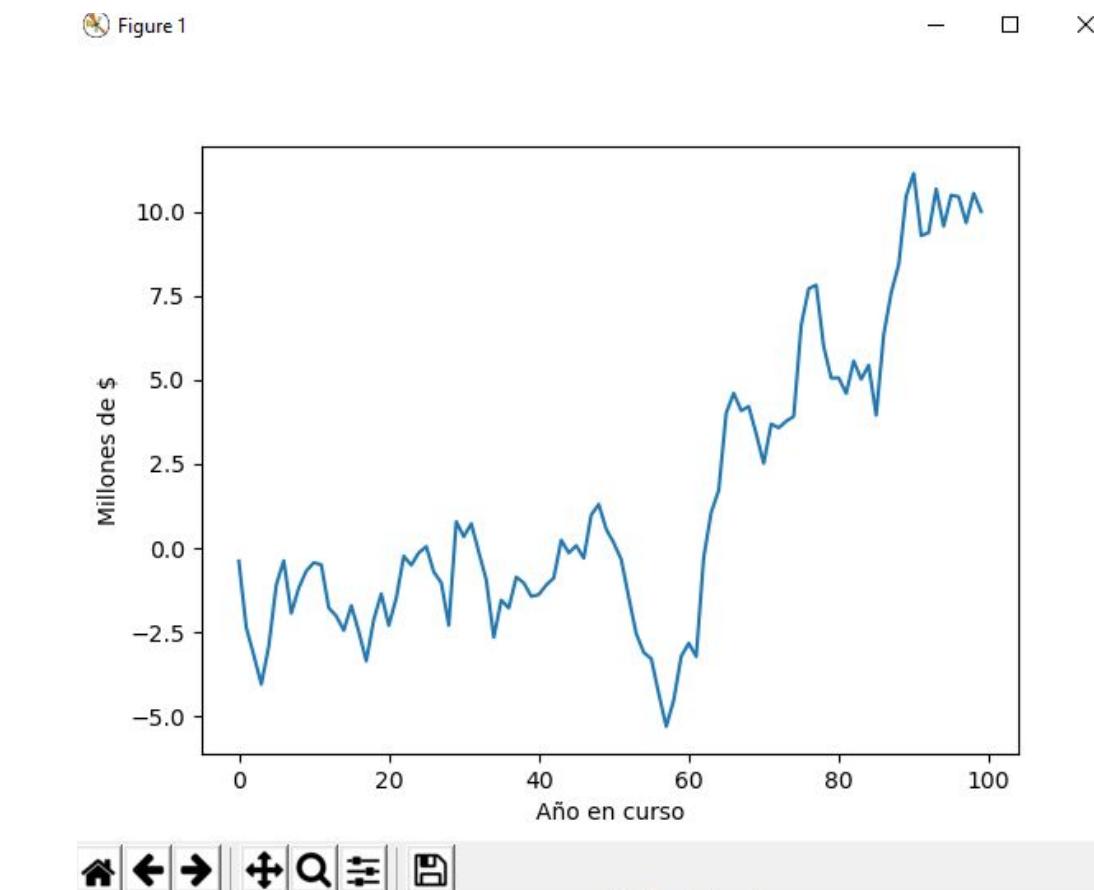
Es posible asignar etiquetas a los ejes x e y con las funciones `matplotlib.pyplot.xlabel` y `matplotlib.pyplot.ylabel`. Ambas funciones aceptan un primer argumento con el texto y aceptan también todos los atributos del texto que hemos visto para el título:

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

y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
plt.plot(y)

plt.xlabel("Año en curso")
plt.ylabel("Millones de $")
plt.show()
```



En el ejemplo de la imagen anterior simplemente ejecutamos las funciones mencionadas sin especificar atributos del texto adicionales. En el siguiente ejemplo se aplica cierto formato a ambas etiquetas:

PERSONALIZACIÓN DE EJES

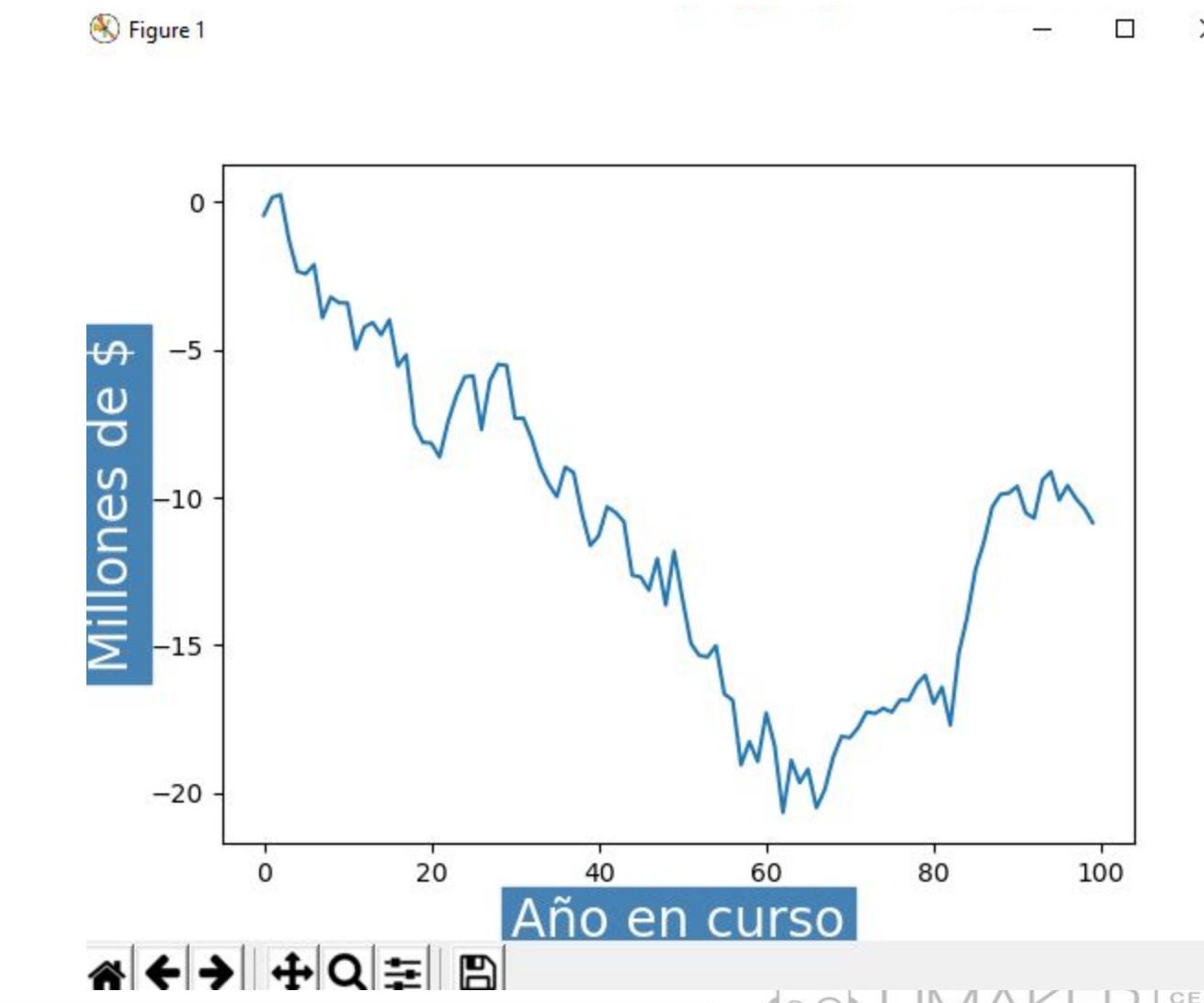
ETIQUETAS DE EJES

En el ejemplo de la imagen anterior simplemente ejecutamos las funciones mencionadas sin especificar atributos del texto adicionales. En el siguiente ejemplo se aplica cierto formato a ambas etiquetas:

```
y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
plt.plot(y)

plt.xlabel("Año en curso",
           fontsize = 20,
           color = "w",
           fontstretch = 0,
           backgroundcolor = "SteelBlue")
plt.ylabel("Millones de $",
           fontsize = 20,
           color = "w",
           fontstretch = 0,
           backgroundcolor = "SteelBlue")
plt.show()
```



PERSONALIZACIÓN DE EJES

MARCAS DE EJES

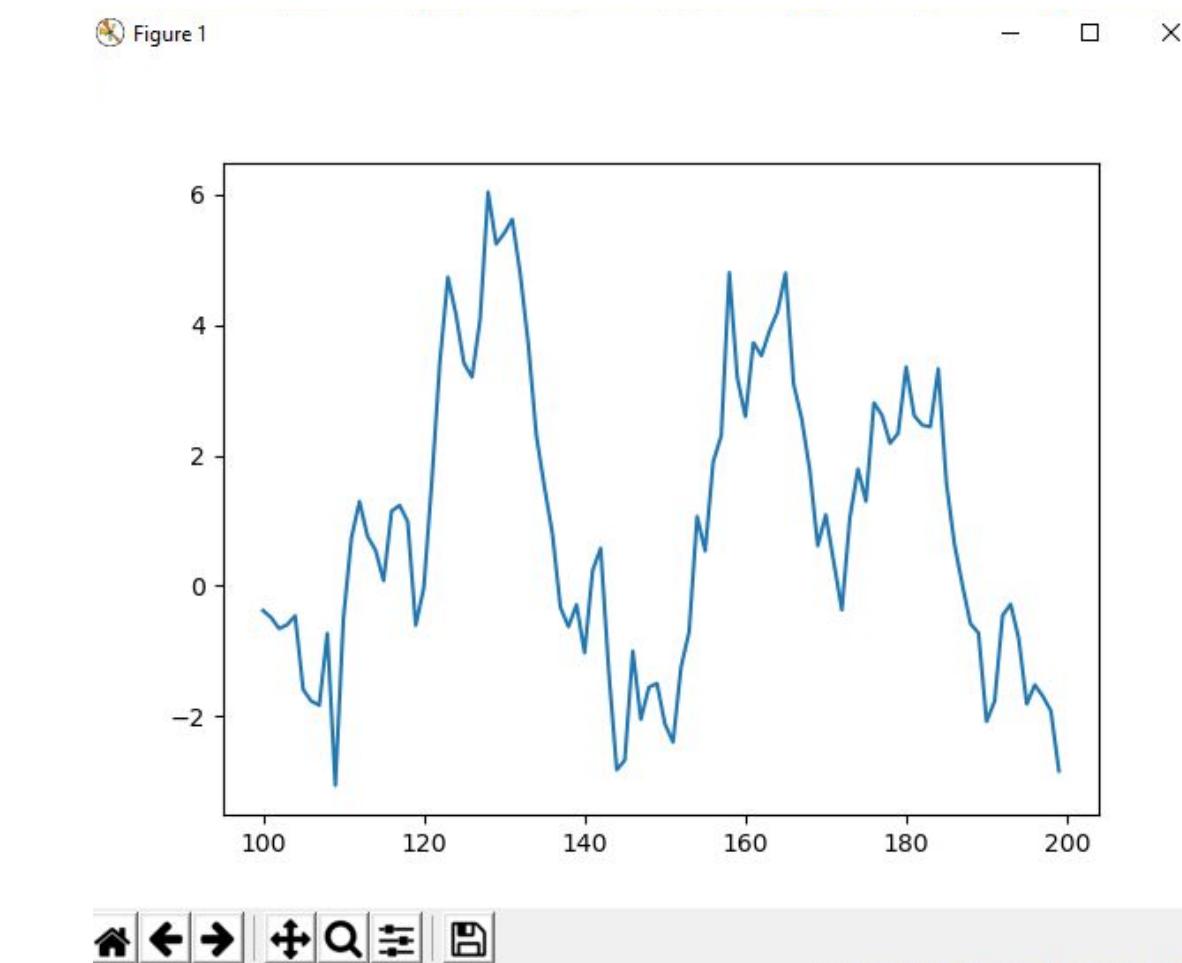
Supongamos que partimos de los siguientes datos:

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

x = range(100,200)
y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
plt.plot(x,y)

plt.show()
```



Vemos que matplotlib hace un más que razonable esfuerzo en mostrar tanto en el eje x como en el eje y marcas suficientemente significativas: en eje x las sitúa de 20 en 20 entre el número 100 y el 200 (es decir, cubriendo todo el rango de nuestros valores x) y en el eje y las sitúa de 2 en 2 entre los valores -4 y 10 (aunque este último valor no lo muestra).

PERSONALIZACIÓN DE EJES

MARCAS DE EJES

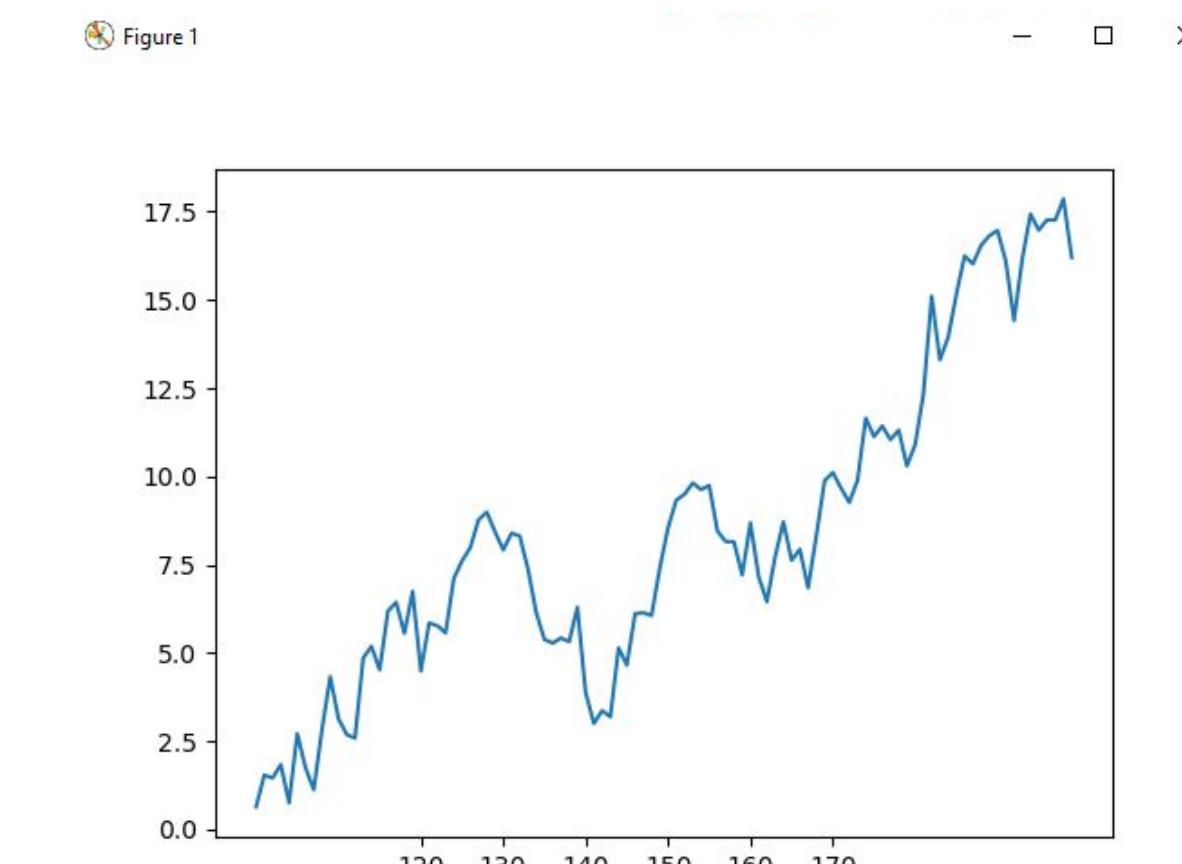
Pero ¿y si queremos mostrar las marcas entre dos valores distintos? ¿o con una frecuencia diferente? Para esto tenemos las funciones `matplotlib.pyplot.xticks` y `matplotlib.pyplot.yticks`. Estas funciones permiten obtener o establecer las marcas en los ejes. Por ejemplo, supongamos que queremos mostrar las marcas del eje x solo entre los valores 120 y 180, pero de 10 en 10. Bastaría con ejecutar la siguiente función:

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

x = range(100,200)
y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
plt.plot(x,y)
plt.xticks(range(120,180,10))

plt.show()
```



PERSONALIZACIÓN DE EJES

MARCAS DE EJES

Si, por otro lado, quisiéramos que las marcas se mostrasen incluyendo valores de x no incluidos en nuestro rango de datos, por ejemplo entre 0 y 250 de 20 en 20, usaríamos la misma función con los valores correspondientes:

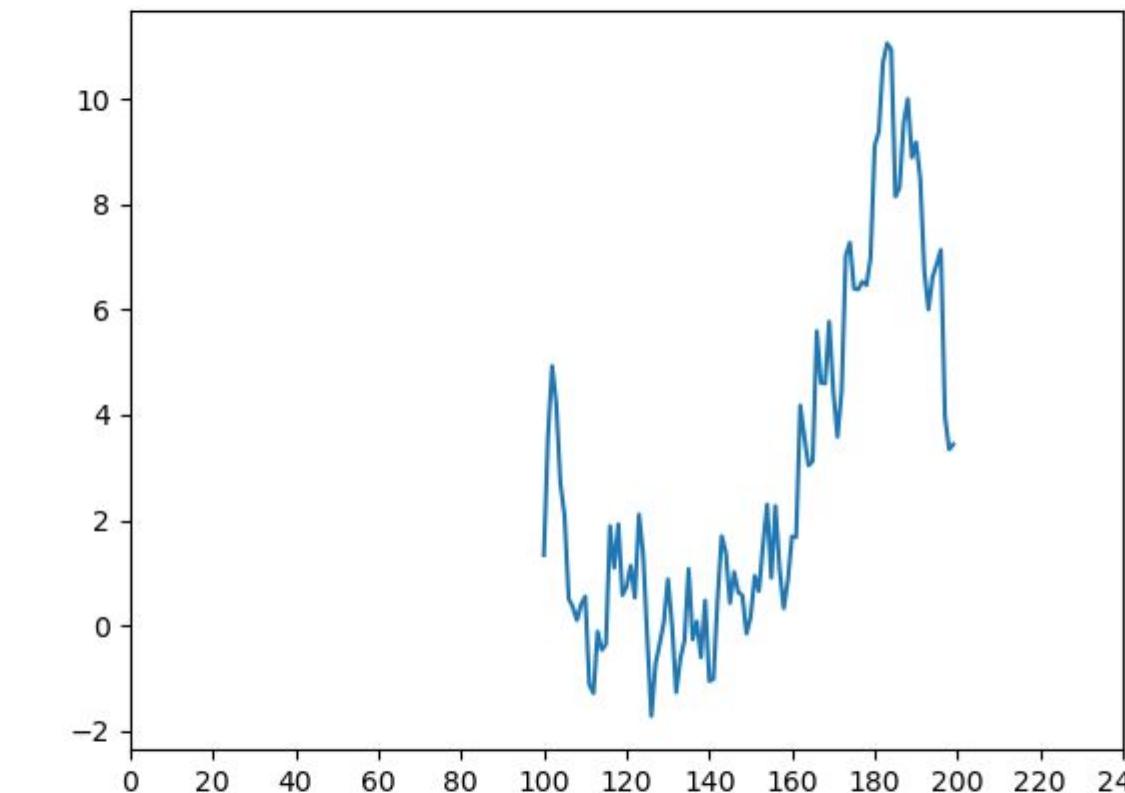
```
import matplotlib.pyplot as plt
import numpy as np

x = range(100,200)
y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
plt.plot(x,y)
plt.xticks(range(0,250,20))

plt.show()
```

Figure 1



PERSONALIZACIÓN DE EJES

MARCAS DE EJES

Vemos que, ahora, la gráfica ocupa un espacio menor para dejar sitio a los valores para los que hemos establecido las marcas.

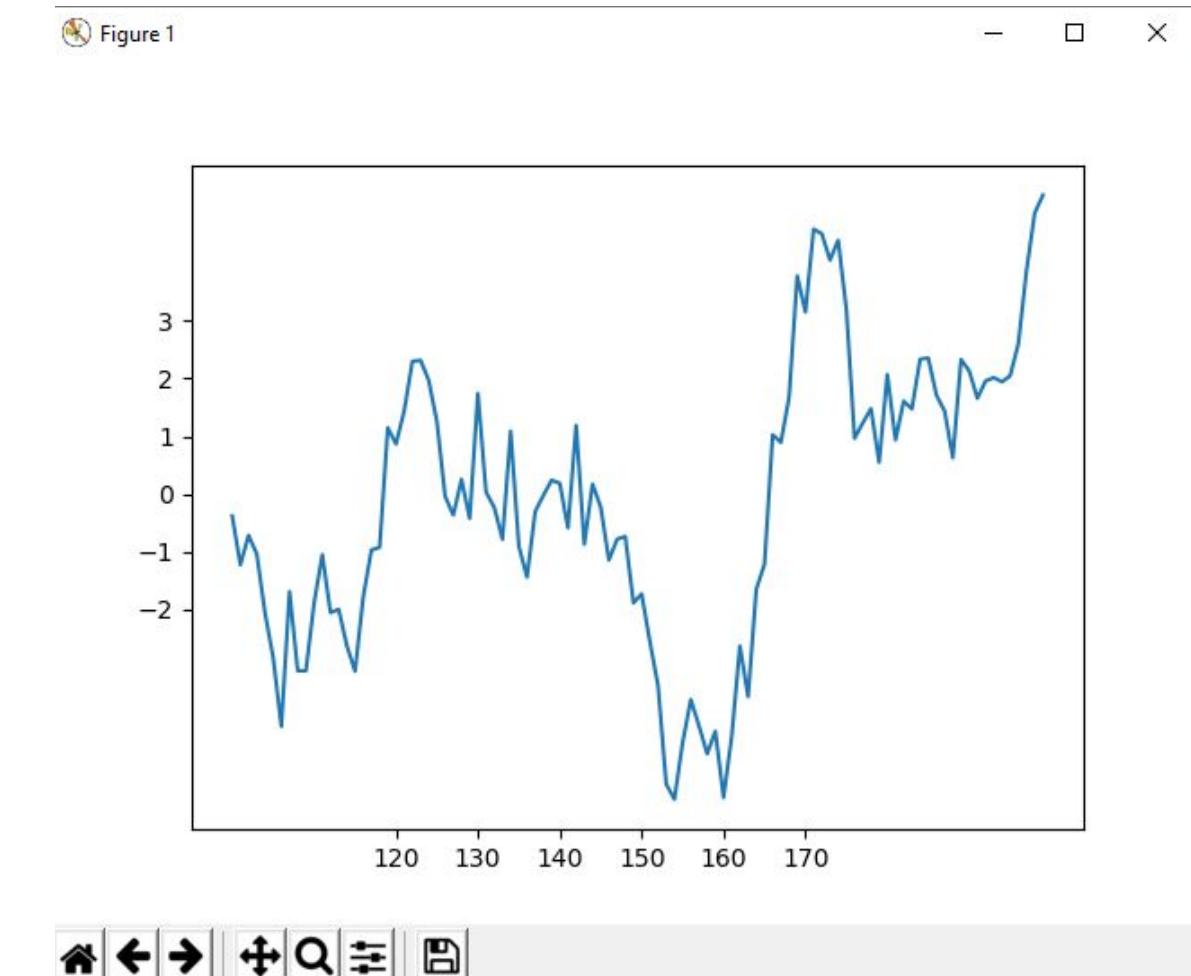
Para el eje y el proceso es exactamente el mismo. Mostremos marcas, por ejemplo solo entre los valores 120 y 180 de 10 en 10 en el eje x, y entre -2 y 4 de 1 en 1 en el eje y:

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

x = range(100,200)
y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
plt.plot(x,y)
plt.xticks(range(120,180,10))
plt.yticks(range(-2,4))

plt.show()
```



PERSONALIZACIÓN DE EJES

MARCAS DE EJES

En realidad, las funciones `plt.xticks` y `plt.yticks` no solo pueden fijar las marcas en los ejes, también pueden fijar las etiquetas de dichas marcas. Por ejemplo, en este último ejemplo que hemos visto, las etiquetas mostradas en el eje x son 120, 130, 140, 150, 160 y 170, pero podríamos querer fijarlas en algo como 2010, 2011, 2012, 2013, 2014 y 2015:

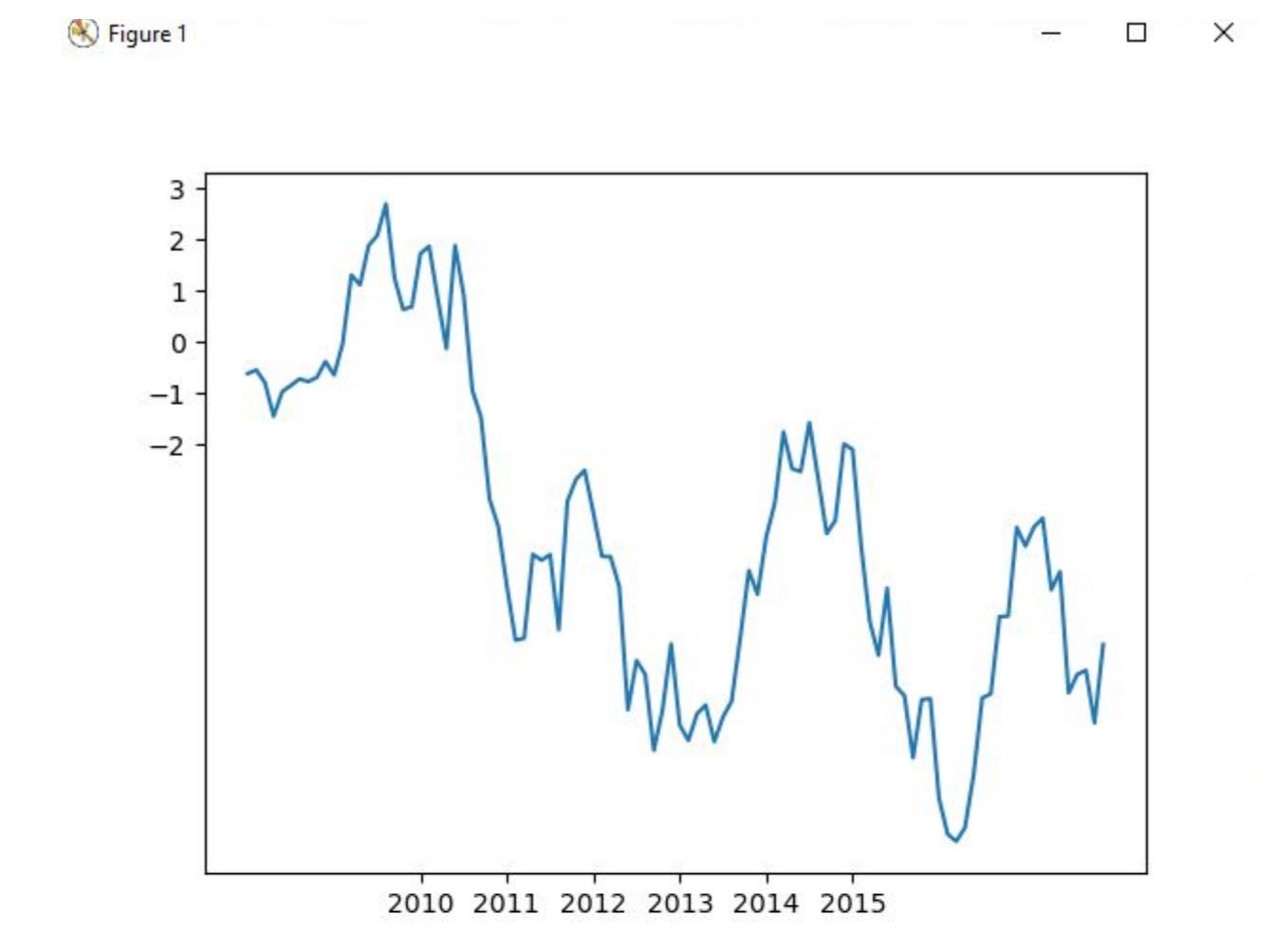
```
import matplotlib.pyplot as plt
import numpy as np

x = range(100,200)
y = np.random.randn(100).cumsum()

xtick_labels = [2010,2011,2012,2013,2014,2015]

fig,ax = plt.subplots()
plt.plot(x,y)
plt.xticks(range(120,180,10),xtick_labels)
plt.yticks(range(-2,4))

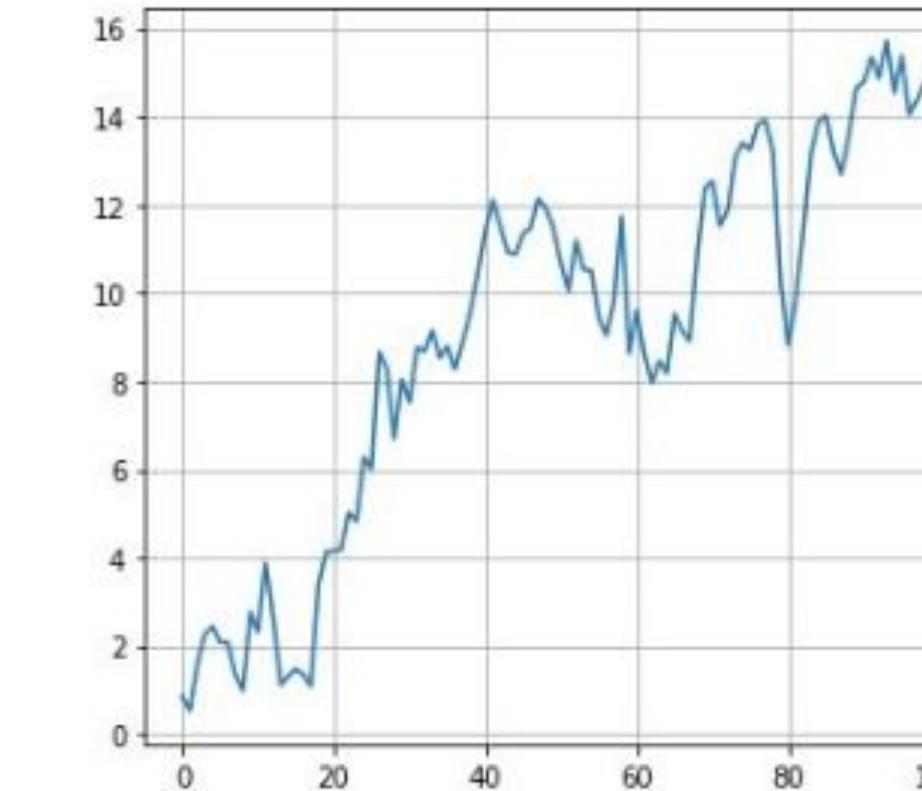
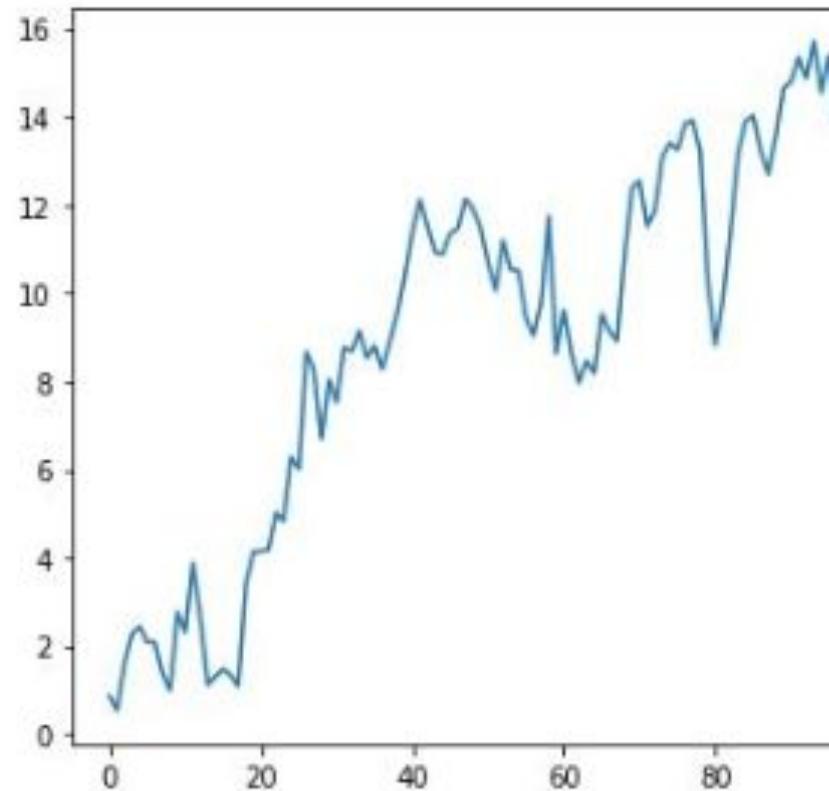
plt.show()
```



PERSONALIZACIÓN DE EJES

GRID

Frecuentemente resulta conveniente mostrar sobre nuestra gráfica un grid (o rejilla) que sirva de referencia para facilitar la interpretación de los datos. Compárense, por ejemplo, estas dos gráficas:



No cabe duda de que resulta más sencillo hacerse una idea de dónde están los máximos y mínimos locales con la ayuda de esta rejilla. Por supuesto, no siempre va a interesar transmitir esta información pero, cuando interese, mostrar el grid y personalizarlo resulta especialmente sencillo. Para mostrarlo tenemos la función `matplotlib.pyplot.grid` o, en el estilo OO, el método `grid` a ejecutar sobre una variable referenciando un conjunto de ejes.

PERSONALIZACIÓN DE EJES

GRID

Hay que mencionar que el grid se muestra sobre las marcas visibles. Esto significa que si deseamos ver el grid también sobre las marcas secundarias, deberemos mostrarlas previamente.

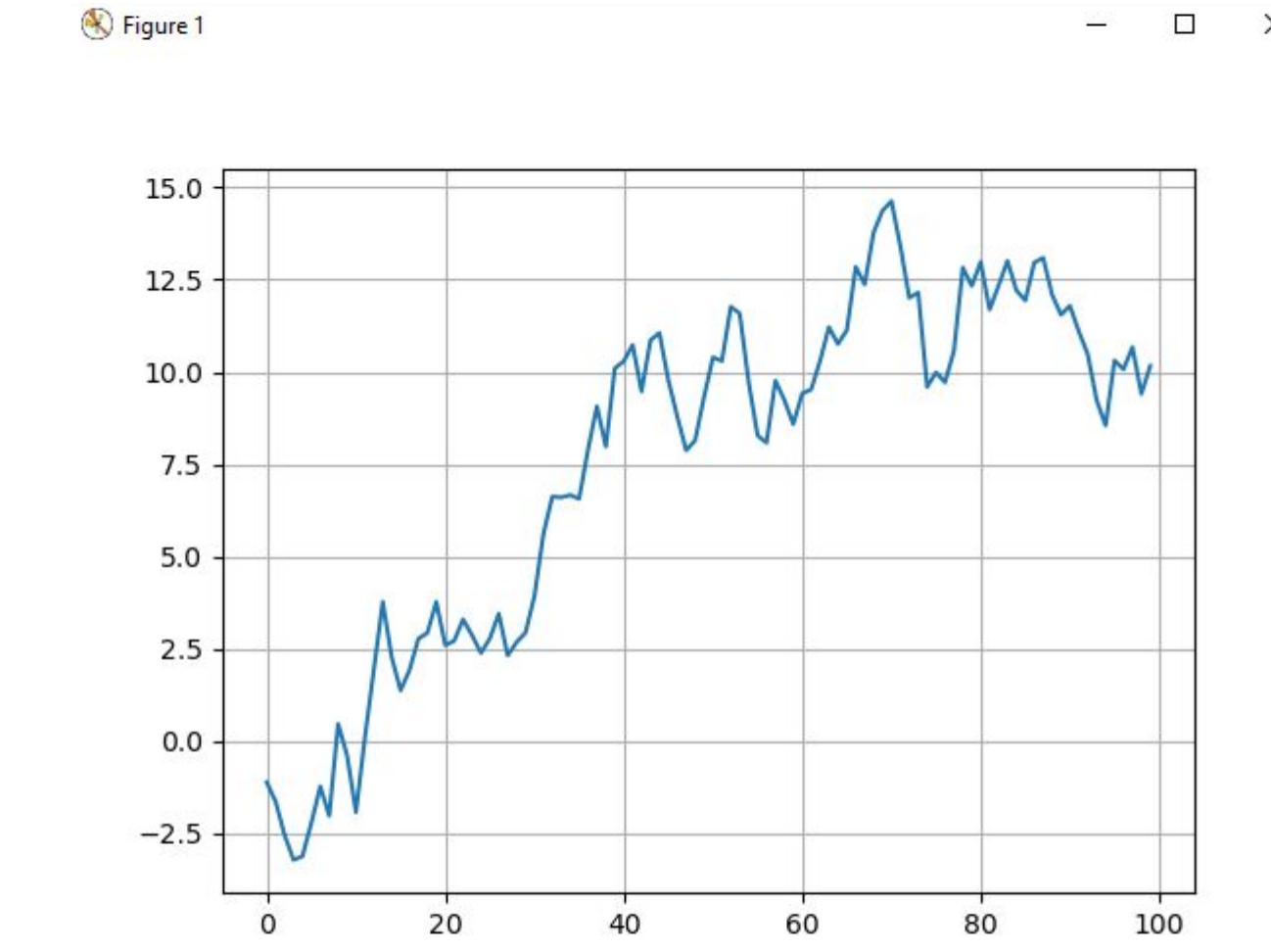
Comencemos con un sencillo ejemplo de uso de la función `grid`:

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

y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
ax.plot(y)
ax.grid()

plt.show()
```



PERSONALIZACIÓN DE EJES

GRID

En los enlaces anteriores que apuntan a la documentación de la función y del método `grid` tenemos acceso a los diferentes parámetros que nos permiten configurarlo, destacando:

- **`alpha`**: grado de transparencia
- **`color`**: color de las líneas del grid
- **`linestyle`**: estilo de las líneas ('-', '--', '-.', ':', etc.)
- **`linewidth`**: ancho de las líneas

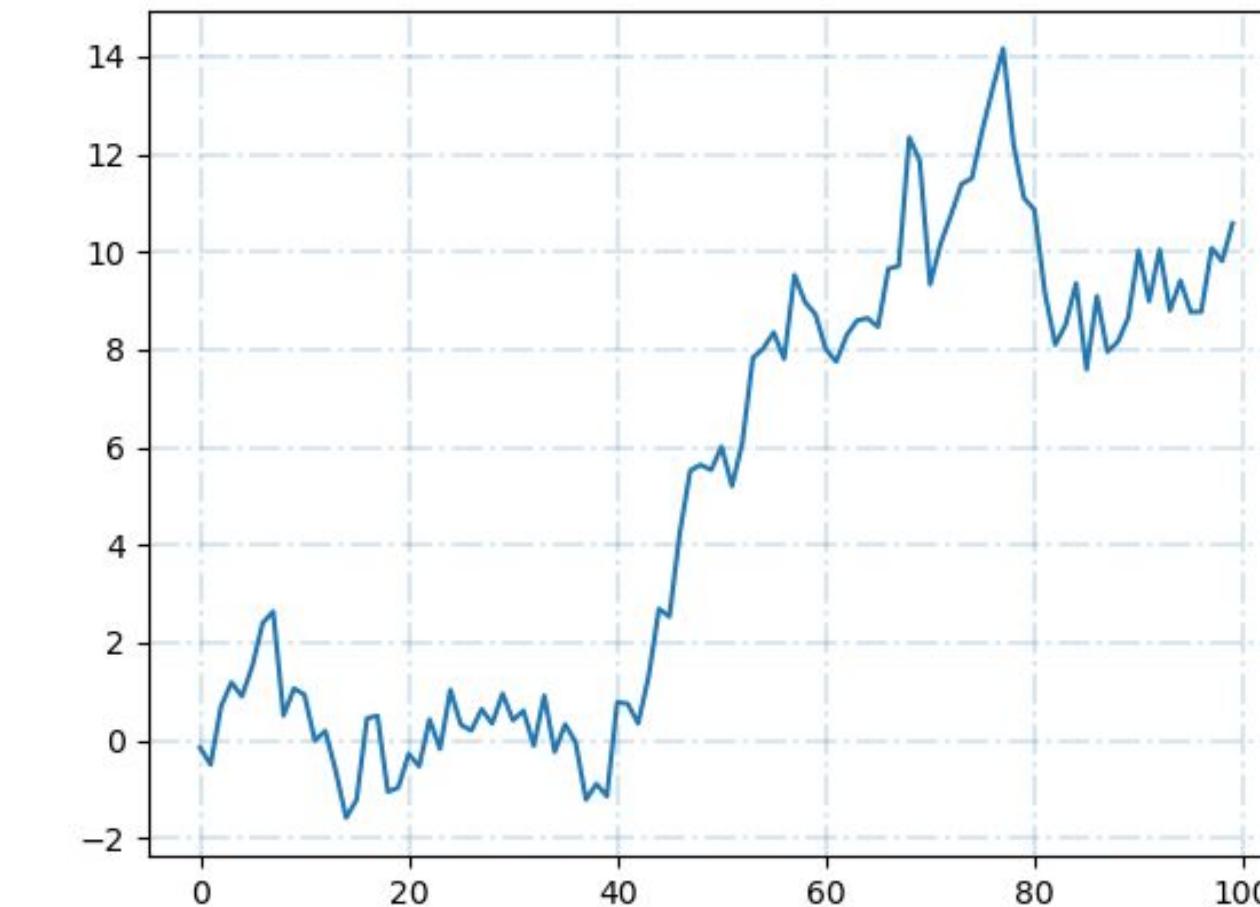
```
import matplotlib.pyplot as plt
import numpy as np

y = np.random.randn(100).cumsum()

fig ,ax = plt.subplots()
ax.plot(y)
ax.grid(alpha = 0.2,
        color = "SteelBlue",
        linestyle = "-.",
        linewidth = 1.5)

plt.show()
```

Figure 1



PERSONALIZACIÓN DE EJES

GRID

Tanto la función como el método grid aceptan, además de los parámetros que personalizan los diferentes atributos, varios parámetros adicionales:

- b**: booleano que indica si se muestran o no las líneas
 - which**: este parámetro puede tomar los valores 'major', 'minor' o 'both', indicando si estamos configurando el grid correspondiente a las marcas principales, a las secundarias o a ambas.
 - axis**: parámetro que puede tomar los valores 'both', 'x' o 'y', y que indica a qué eje vamos a aplicar la configuración.
- Esto nos permite dar formato al grid con un alto grado de detalle. En el siguiente ejemplo estamos mostrando las marcas principales y las secundarias tanto para el eje x como para el eje y, y estableciendo un formato diferente para cada uno de los conjuntos de marcas (eje x/principales, eje x/secundarias, eje y/principales y eje y/secundarias):

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import MultipleLocator

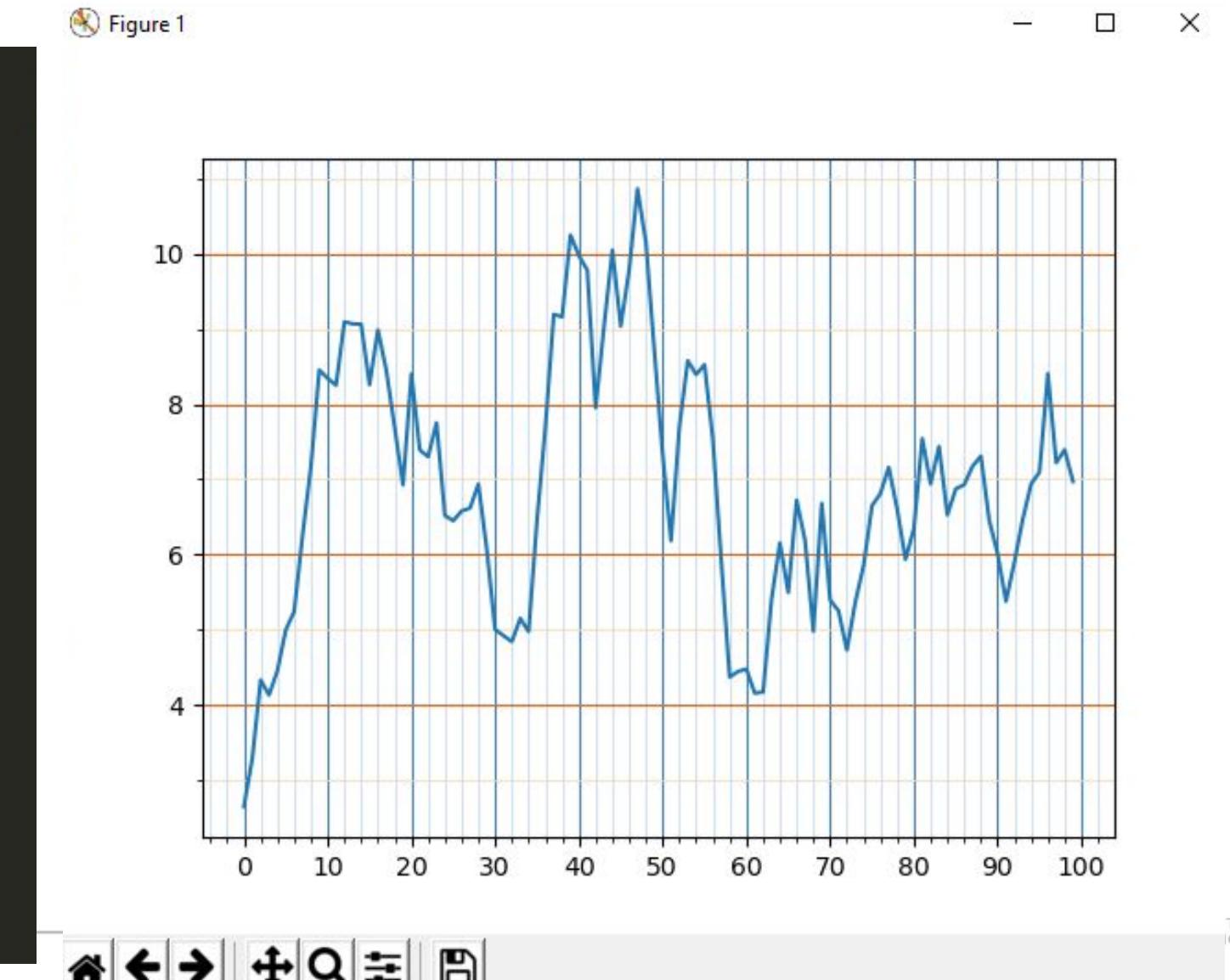
y = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
ax.plot(y)

#Eje x
ax.xaxis.set_major_locator(MultipleLocator(10))
ax.xaxis.set_minor_locator(MultipleLocator(2))
#Eje y
ax.yaxis.set_major_locator(MultipleLocator(2))
ax.yaxis.set_minor_locator(MultipleLocator(1))

ax.grid(which = "major",axis = "x",color = "SteelBlue")
ax.grid(which = "minor",axis = "x",color = "LightSteelBlue",alpha = 0.5)

ax.grid(which = "major",axis = "y",color = "Chocolate")
ax.grid(which = "minor",axis = "y",color = "wheat",alpha = 0.8)
plt.show()
```



PERSONALIZACIÓN DE EJES

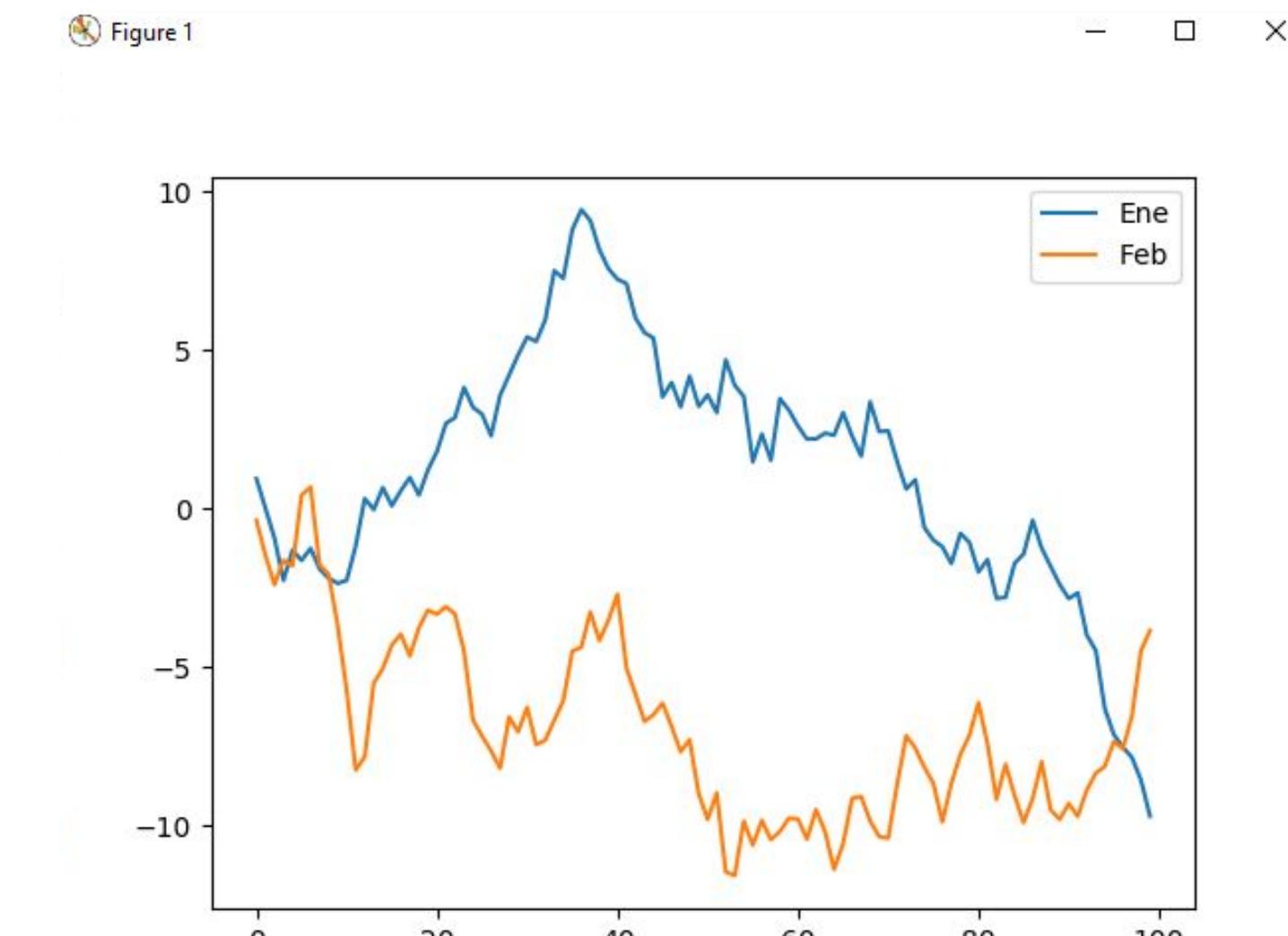
LEYENDA

La función `matplotlib.pyplot.legend` y el método `legend` de un conjunto de ejes muestra la leyenda en el gráfico. Vimos en su momento que uno de los parámetros que podemos pasar a la función `plot` es `label`. Esta etiqueta es la que se mostrará en la leyenda representando a la gráfica. Para ver cómo funciona, partamos de dos conjuntos de datos a representar en sendas gráficas, y mostremos éstas con su label correspondiente:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import MultipleLocator

y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
ax.plot(y1, Label = "Ene")
ax.plot(y2, Label = "Feb")
ax.legend()
plt.show()
```



PERSONALIZACIÓN DE EJES

LEYENDA

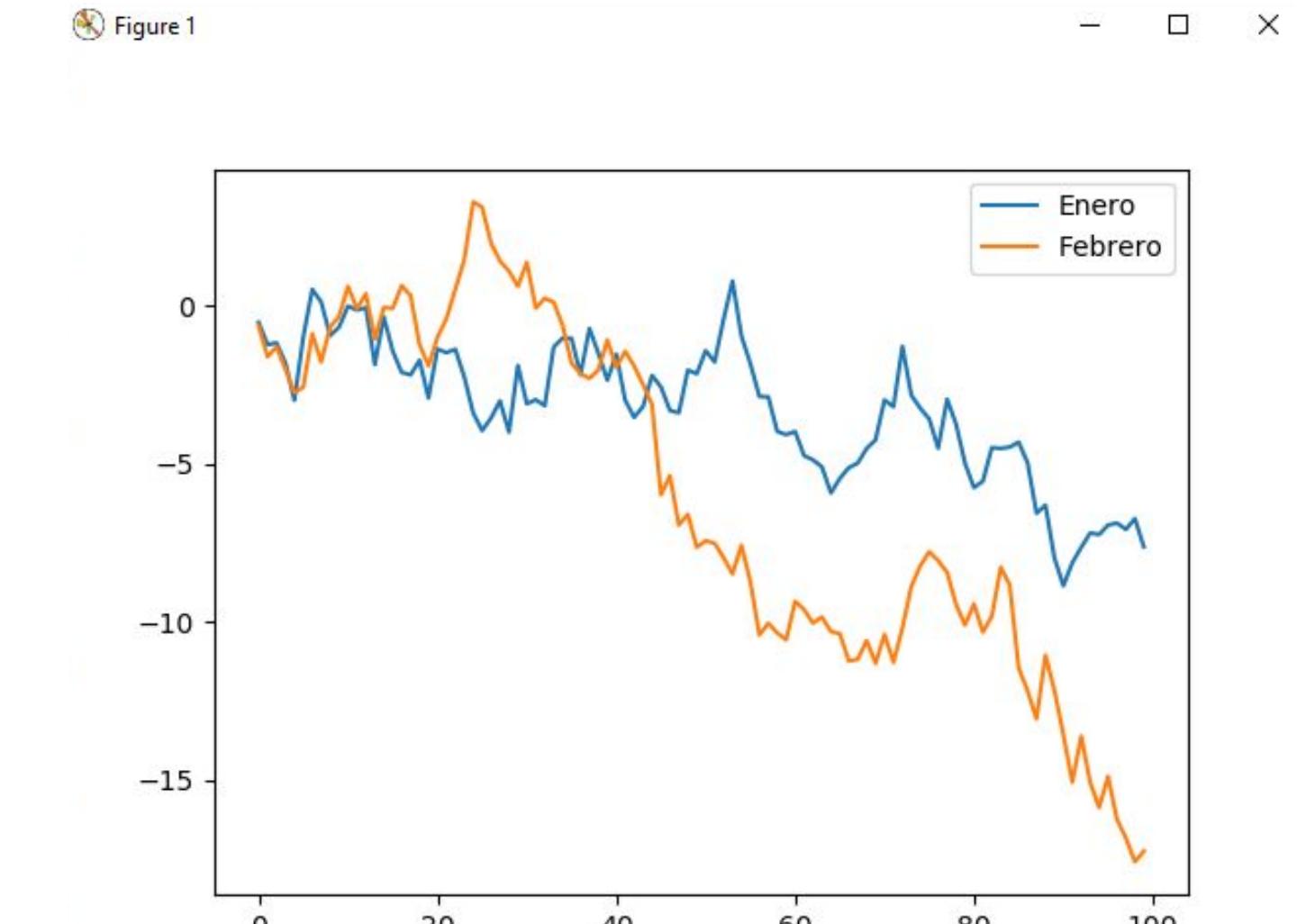
Se ha aplicado a la primera gráfica la etiqueta "Ene" y a la segunda "Feb", y vemos que son precisamente estas etiquetas las que se muestran en la leyenda.

Estas etiquetas se pueden sobreescribir al llamar a la función o método *legend* si pasamos los nuevos valores como argumentos:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import MultipleLocator

y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
ax.plot(y1, label = "Ene")
ax.plot(y2, label = "Feb")
ax.legend(["Enero","Febrero"])
plt.show()
```



PERSONALIZACIÓN DE EJES

LEYENDA

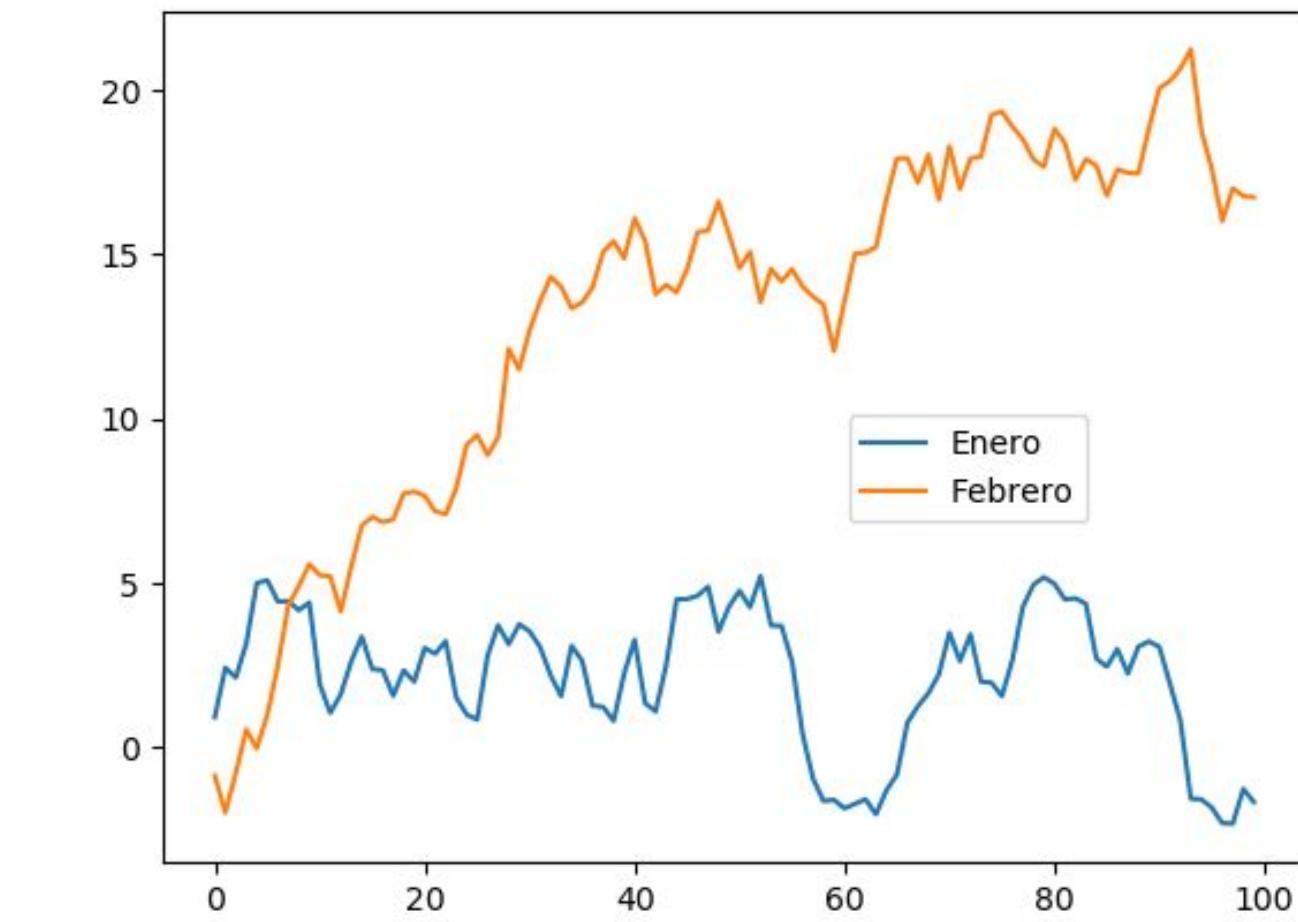
También podemos pasar al argumento col una tupla con la posición x e y de la leyenda con respecto al ancho y alto del área de la gráfica. Por ejemplo:

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

y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
ax.plot(y1, label = "Ene")
ax.plot(y2, label = "Feb")
ax.legend(["Enero","Febrero"], loc = (0.6,0.4))
plt.show()
```

Figure 1



MASTER PYTHON

LEYENDA

Hay, en todo caso, otros parámetros de esta función (y método) interesantes:

- **ncol**: este parámetro permite especificar el número de columnas en las que se van a mostrar las etiquetas en la leyenda (por defecto es 1):

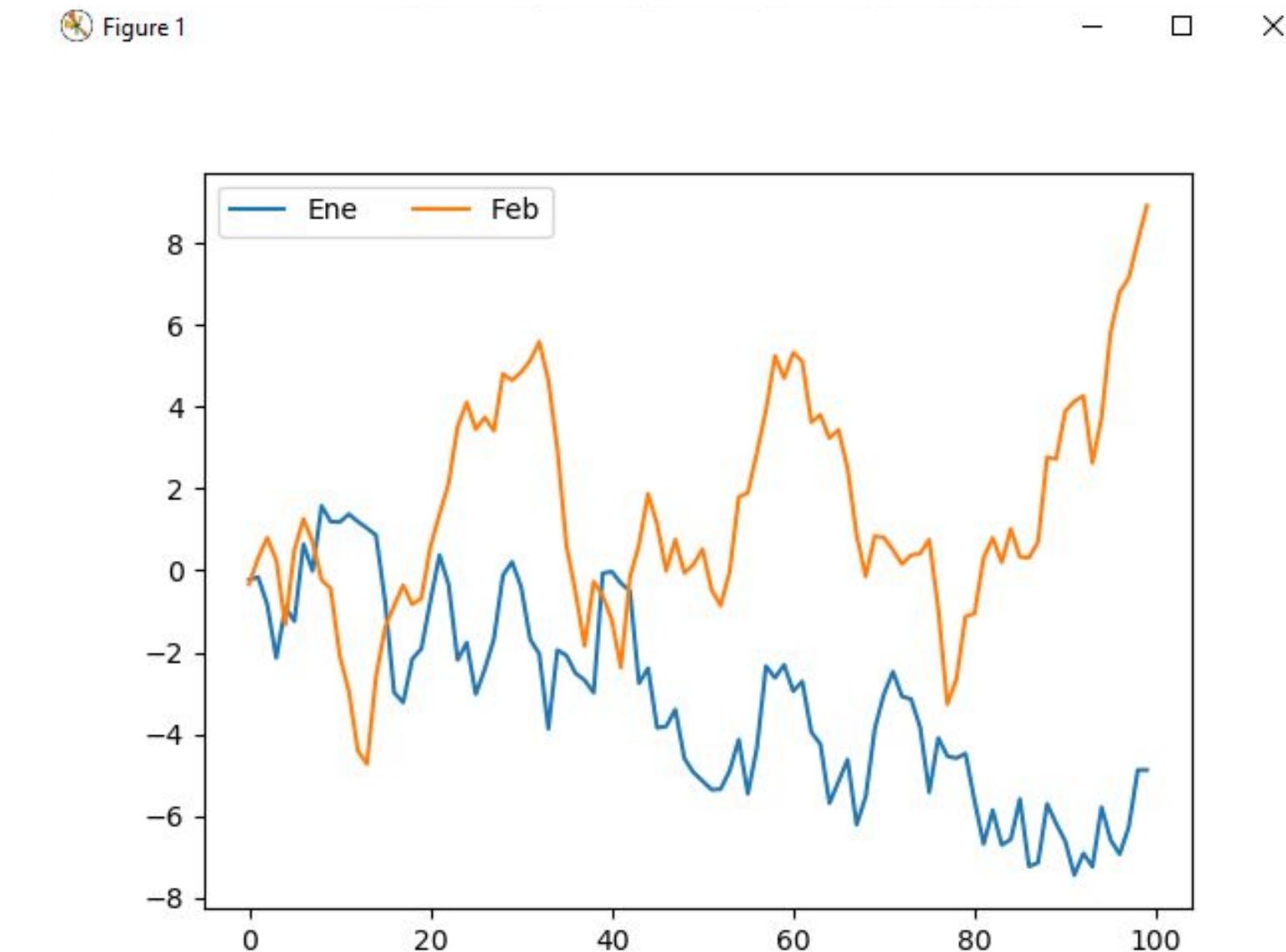
```
import matplotlib.pyplot as plt
import numpy as np

y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
ax.plot(y1, label = "Ene")
ax.plot(y2, label = "Feb")
ax.legend(ncol = 2)
plt.show()
```



MASTER PYTHON



LEYENDA

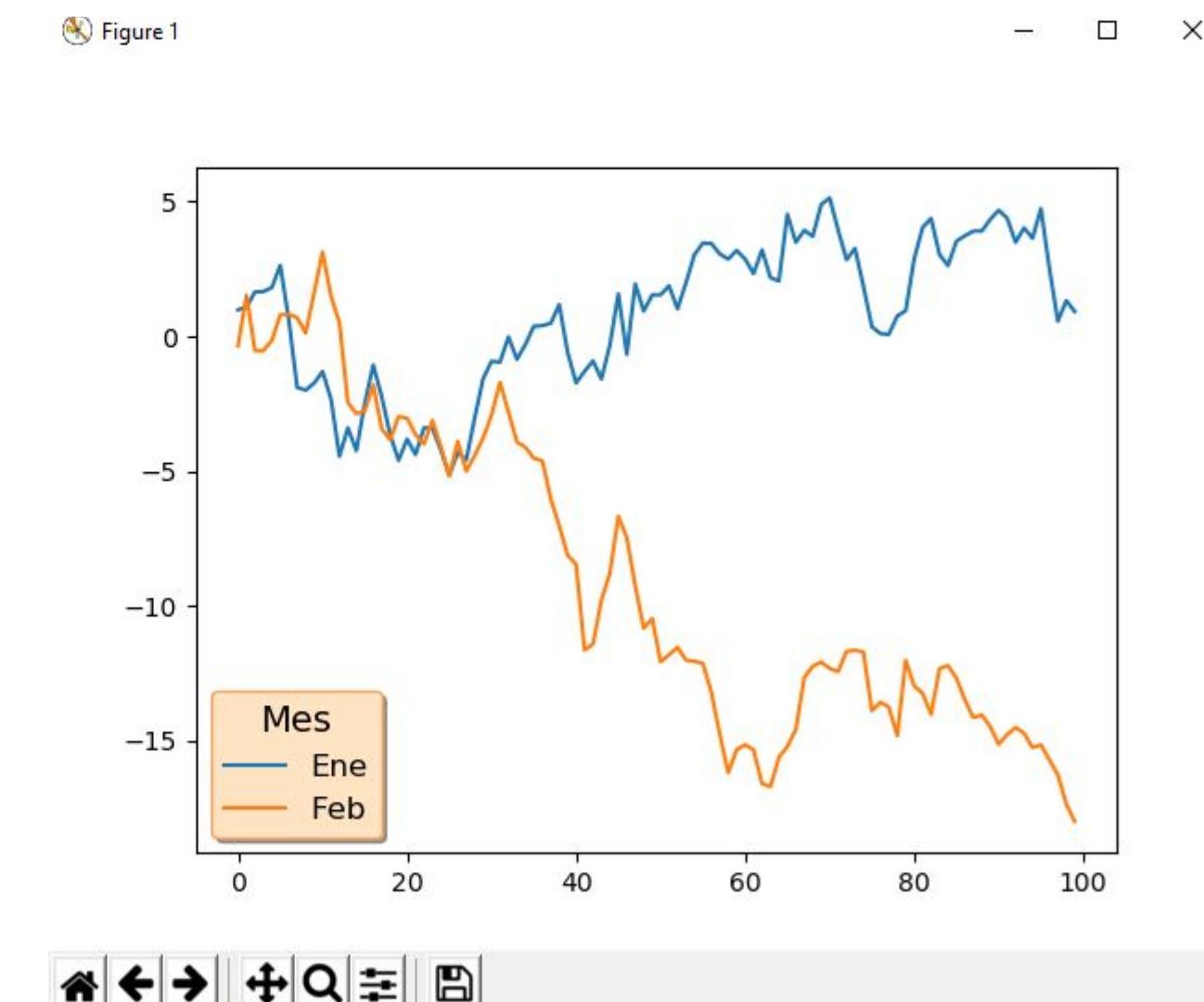
- **fontsize**: este parámetro determina el tamaño de la fuente.
 - **shadow**: muestra u oculta una sombra alrededor de la leyenda.
 - **facecolor**: establece el color de fondo de la leyenda.
 - **edgecolor**: establece el color del borde de la leyenda.
 - **title**: fija un título para la leyenda
 - **title_fontsize**: establece el tamaño del título de la leyenda
- En el siguiente ejemplo se hace uso de estos parámetros:

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

y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()

fig,ax = plt.subplots()
ax.plot(y1, label = "Ene")
ax.plot(y2, label = "Feb")
ax.legend(fontsize = 12,
          shadow = True,
          facecolor = "Bisque",
          edgecolor = "SandyBrown",
          title = "Mes",
          title_fontsize = 14)

plt.show()
```



MASTER PYTHON

PERSONALIZACIÓN DE EJES

ESTILOS Y MAPAS DE COLOR

Dos recursos gráficos a los que podemos recurrir son los estilos -mapas de atributos que definen el estilo gráfico en matplotlib- y los mapas de color (cmap), diccionarios que transforman unos colores en otros.

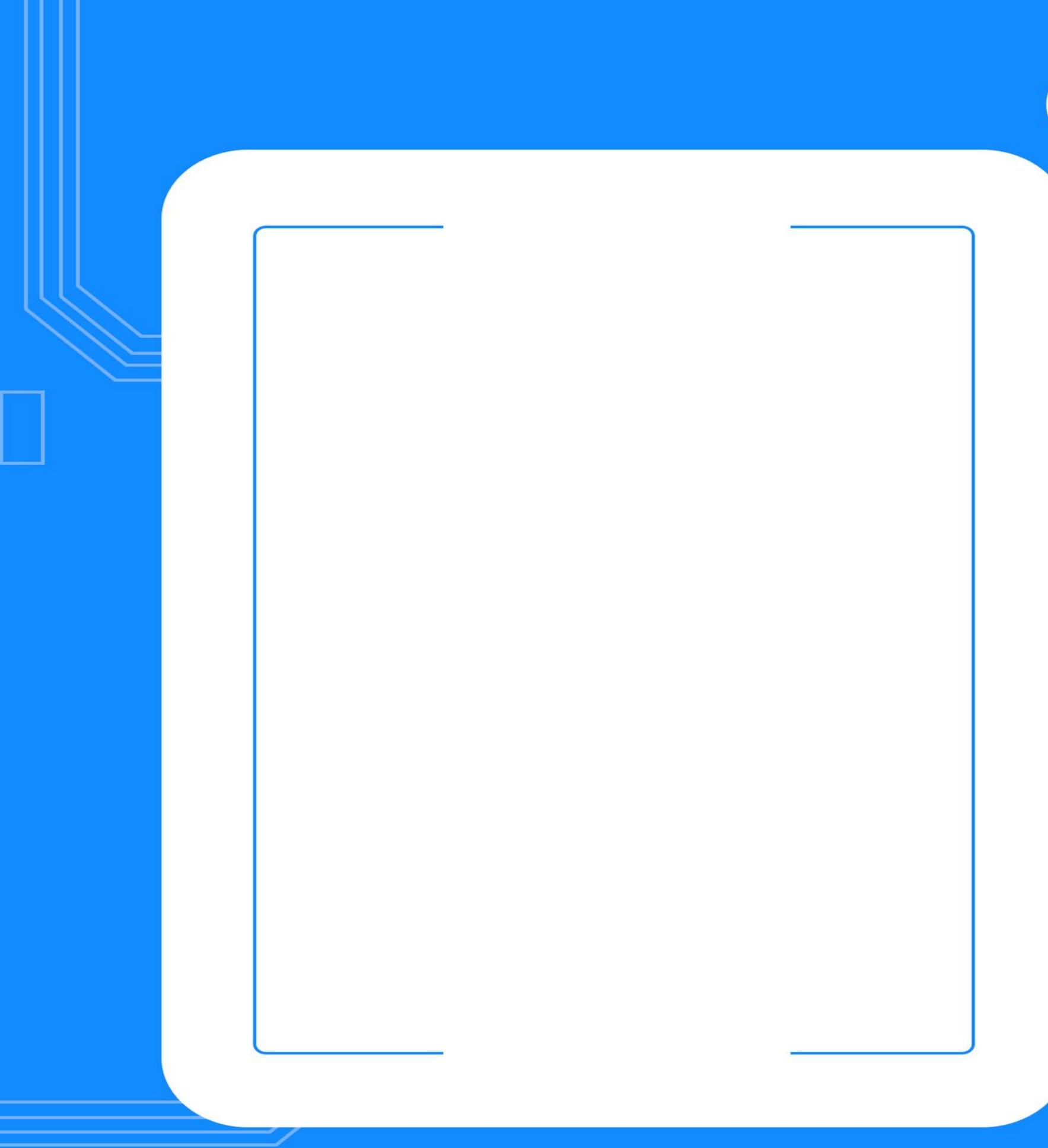
Las versiones más recientes de matplotlib permiten la aplicación de "estilos" que mejoran la apariencia de las gráficas creadas o que las modifican de forma tal que las hacen más amigables en ciertos entornos.

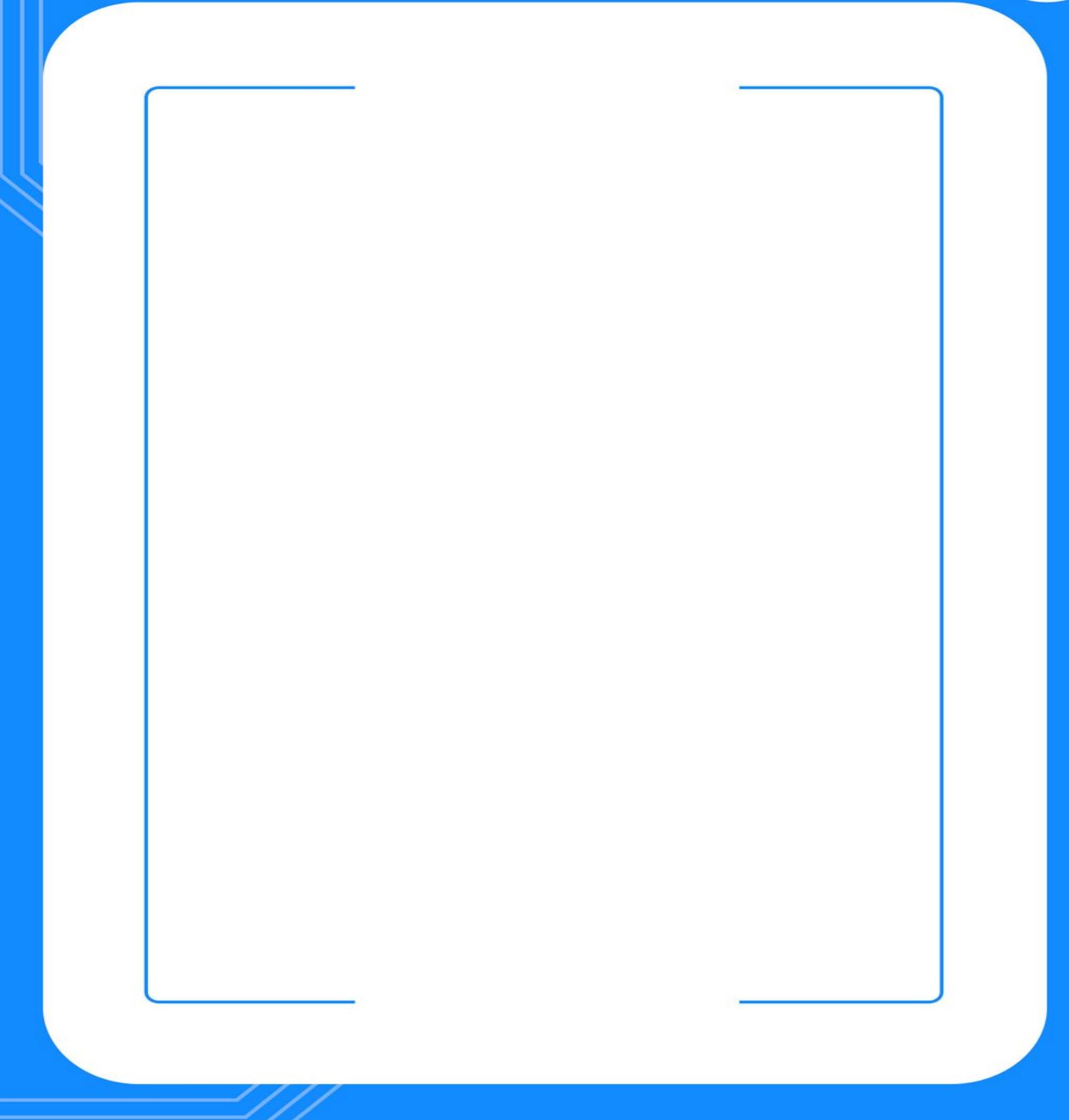
Podemos comprobar los estilos disponibles con la instrucción `plt.style.available`:

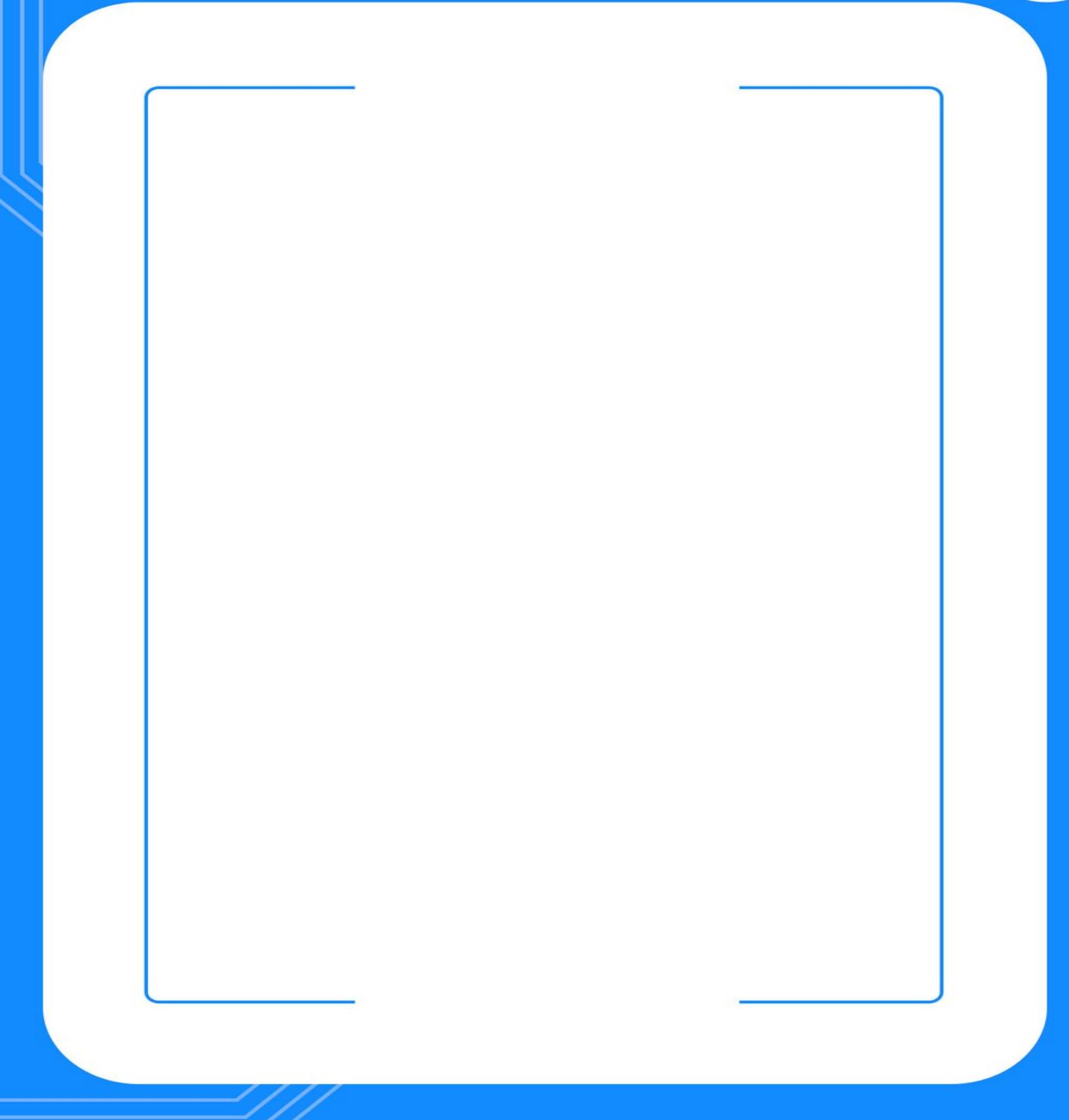
```
>>> import matplotlib.pyplot as plt
>>> plt.style.available
['bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark-palette', 'seaborn-dark', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'seaborn', 'Solarize_Light2', 'tableau-colorblind10', '_classic_test']
>>> _
```

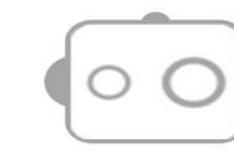
Para usar un estilo, por ejemplo, "seaborn", basta ejecutar la siguiente instrucción











UMAKER | CENTRO DE CAPACITACIÓN
DE DESARROLLO TECNOLÓGICO