Matplotlib

Unidad 6

Apunte de cátedra

Pensamiento computacional (90)
Cátedra: Camejo

.UBAXXI

Estructura de este apunte

Este apunte fue armado primero como un Google Colab antes de ser pasado a PDF. Esto significa que pueden ir al siguiente link y conseguir una versión más interactiva si así prefieren. Es importante destacar que no van a poder modificar el archivo porque solo tiene permisos de lectura, deben crear una copia primero. Una vez hecha la copia van a poder cambiar el código y probar cosas por su cuenta.

¿Qué es Matplotlib?

Matplotlib es probablemente la biblioteca de Python más usada para crear gráficos en 2D, también llamados plots. Provee una forma rápida de graficar datos en varios formatos de alta calidad que pueden ser compartidos y/o publicados. En esta sección vamos a ver los usos más comunes de matplotlib. En este video encontrarán una introducción breve a esta sección.

pyplot

pyplot proporciona una interfaz a la biblioteca de matplotlib. Pyplot está diseñada siguiendo el estilo de Matlab y la mayoría de los comandos para graficar en pyplot tienen análogos en Matlab con argumentos similares. Explicaremos las instrucciones más importantes con ejemplos interactivos.

```
from matplotlib import pyplot as plt
```

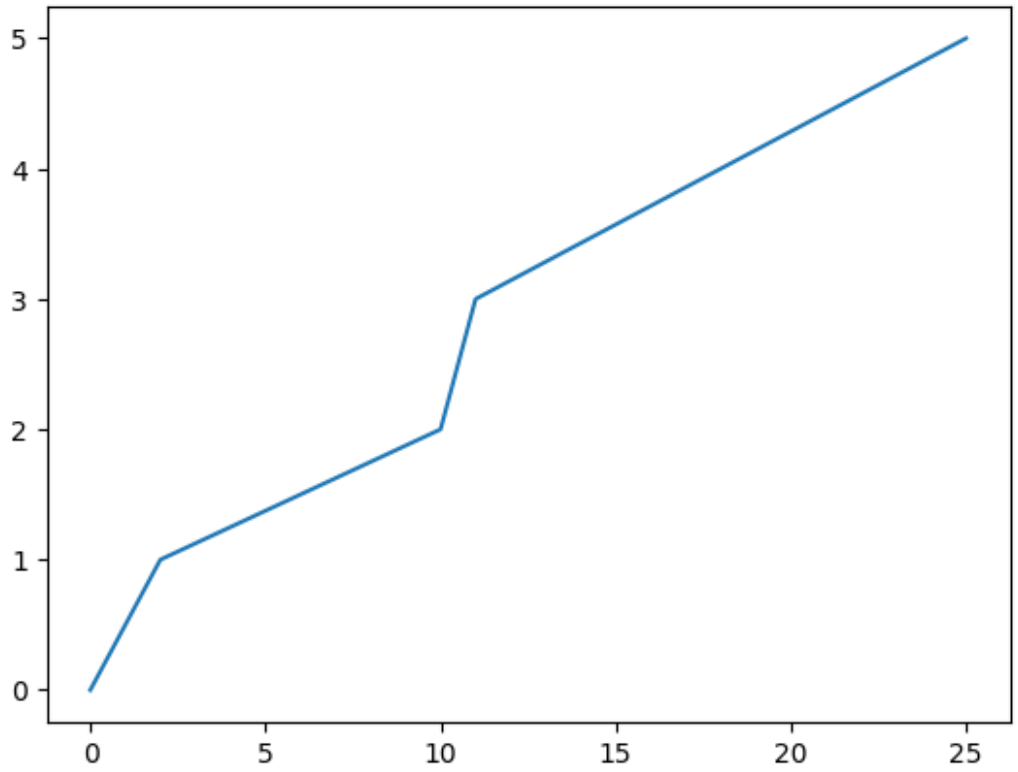
Las primeras funciones que vamos a ver son `plot` y `show`.

La función `plot` recibe 2 arreglos que tienen que ser del mismo tamaño que representan una serie de puntos en el plano cartesiano. El primer arreglo son las coordenadas X y el segundo son las coordenadas Y. Finalmente lo que hace la función es unir todos estos puntos con lineas para que en el dibujo podamos ver el gráfico de la función.

`show` simplemente crea la imagen con todos los gráficos definidos anteriormente.

```
x = [0,2,10,11,18,25]
y = [0,1,2,3,4,5]

plt.plot(x, y)

plt.show()
```

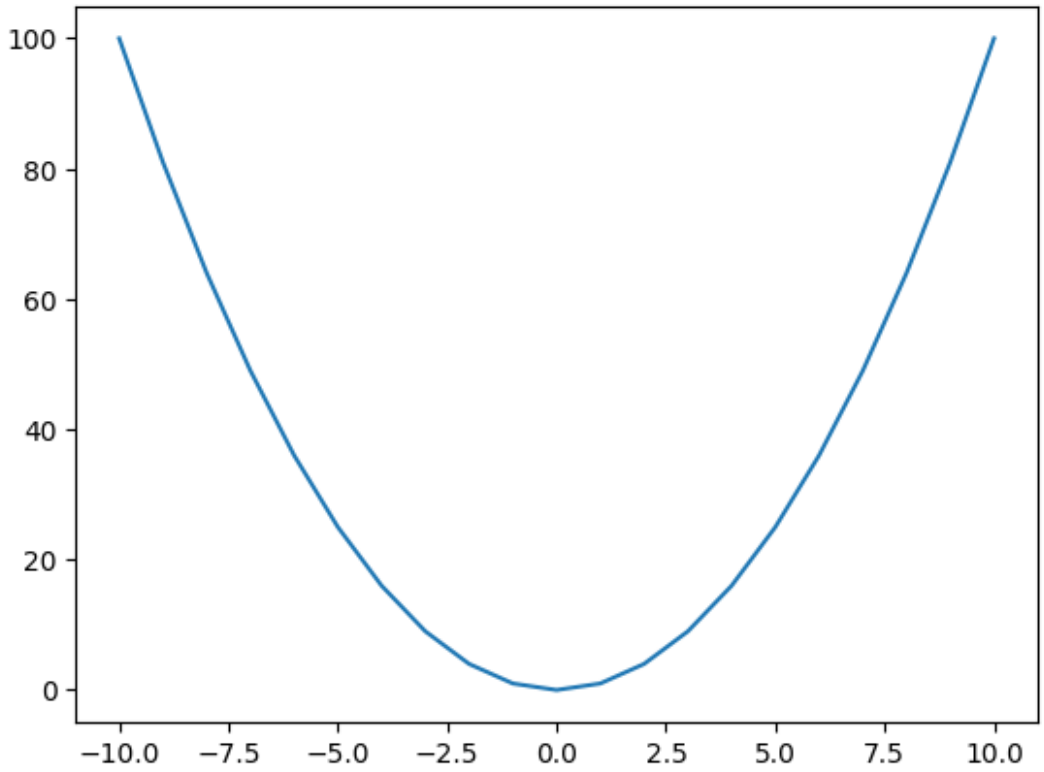
Resultado:



Si aumentamos la cantidad de puntos entonces nuestra linea va a ser más suave. En este caso podemos usar 21 puntos para graficar $f(x) = x^2$

```
x = [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [100,81,64,49,36,25,16,9,4,1,0,1,4,9,16,25,36,49,64,81,100]

plt.plot(x, y)

plt.show()
```

Resultado:



Poniendo títulos

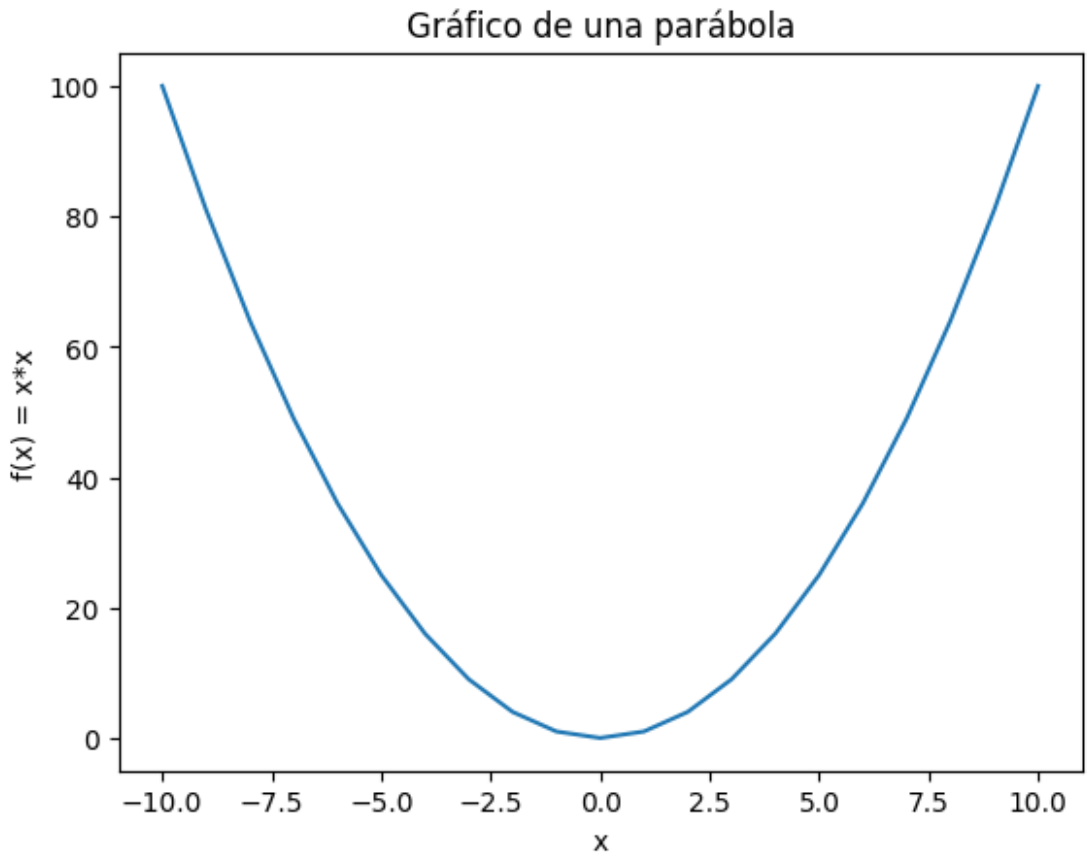
Una de las partes más importantes para que un gráfico se pueda entender es ponerle un título y explicar que significa cada eje.

Eso se hace con las funciones `xlabel`, `ylabel` y `title`. Cada una recibe un string que se usará como etiqueta del eje X, etiqueta del eje Y o título, respectivamente.

A continuación se muestra un ejemplo con nuestra parábola anterior.

```
x = [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [100,81,64,49,36,25,16,9,4,1,0,1,4,9,16,25,36,49,64,81,100]

plt.plot(x, y)

plt.xlabel('x')
plt.ylabel('f(x) = x*x')

# displaying the title
plt.title("Gráfico de una parábola")

plt.show()
```

Resultado:



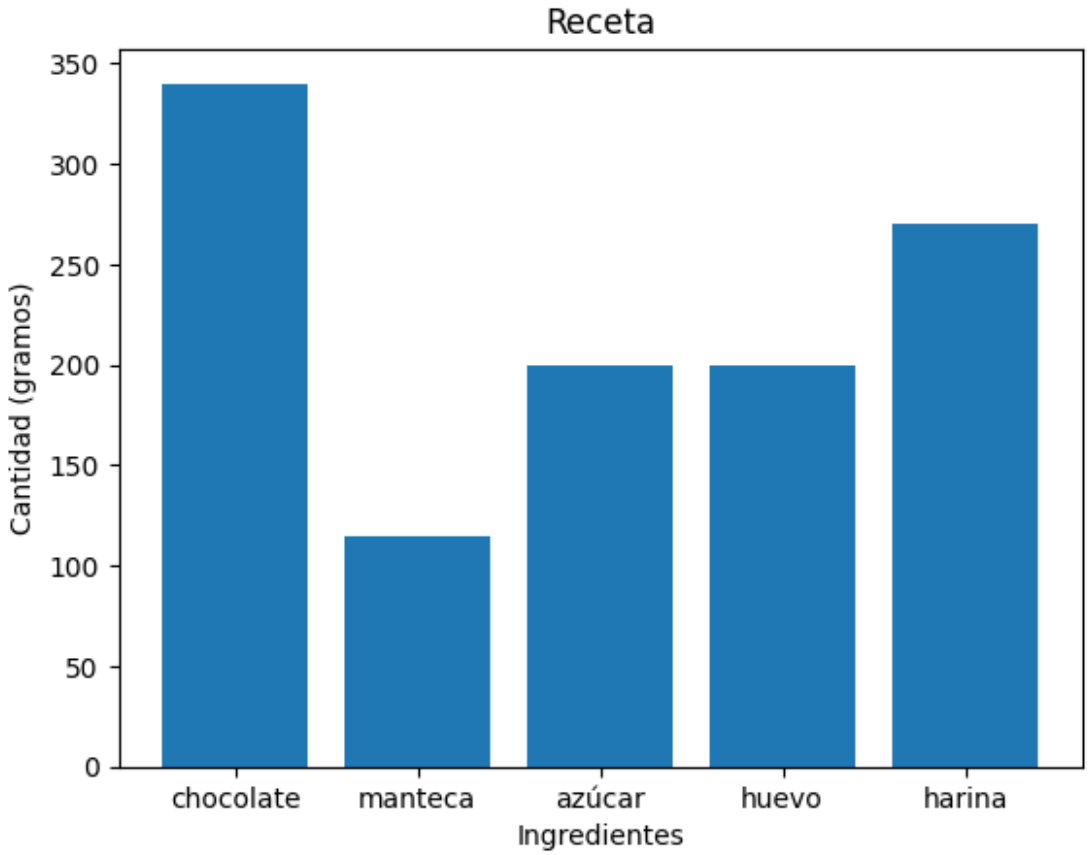
Más tipos de gráficos

Ahora vamos a ver gráficos diferentes a una curva. La fórmula siempre es la misma, primero se define el tipo de gráfico y después se usa `show` para mostrarlo.

Para hacer un gráfico de barras se usa `bar`. Esta función primero recibe un arreglo con las etiquetas de las barras que se van a mostrar y después otro arreglo con la altura de cada una de estas barras.

```
peso = [340, 115, 200, 200, 270]
ingredientes = ['chocolate', 'manteca', 'azúcar', 'huevo', 'harina']

plt.bar(ingredientes, peso)

plt.xlabel('Ingredientes')
plt.ylabel('Cantidad (gramos)')

plt.title("Receta")

plt.show()
```

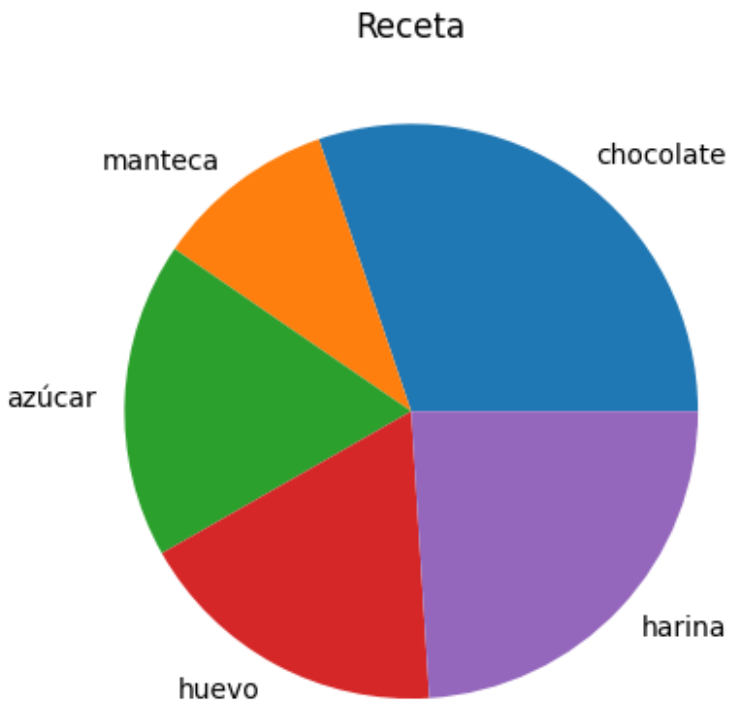
Resultado:



Esto mismo podríamos representarlo con un gráfico de torta como para tener una mejor noción de la proporción de cada elemento. Esto se hace con la función `pie`. Esta función podría solamente recibir un arreglo de números pero es útil también saber que simboliza cada parte, para eso se usa el parámetro `labels` que le asigna una etiqueta a cada porción.

```
peso = [340, 115, 200, 200, 270]
ingredientes = ['chocolate', 'manteca', 'azúcar', 'huevo', 'harina']

plt.pie(peso, labels= ingredientes)

plt.title("Receta")

plt.show()
```

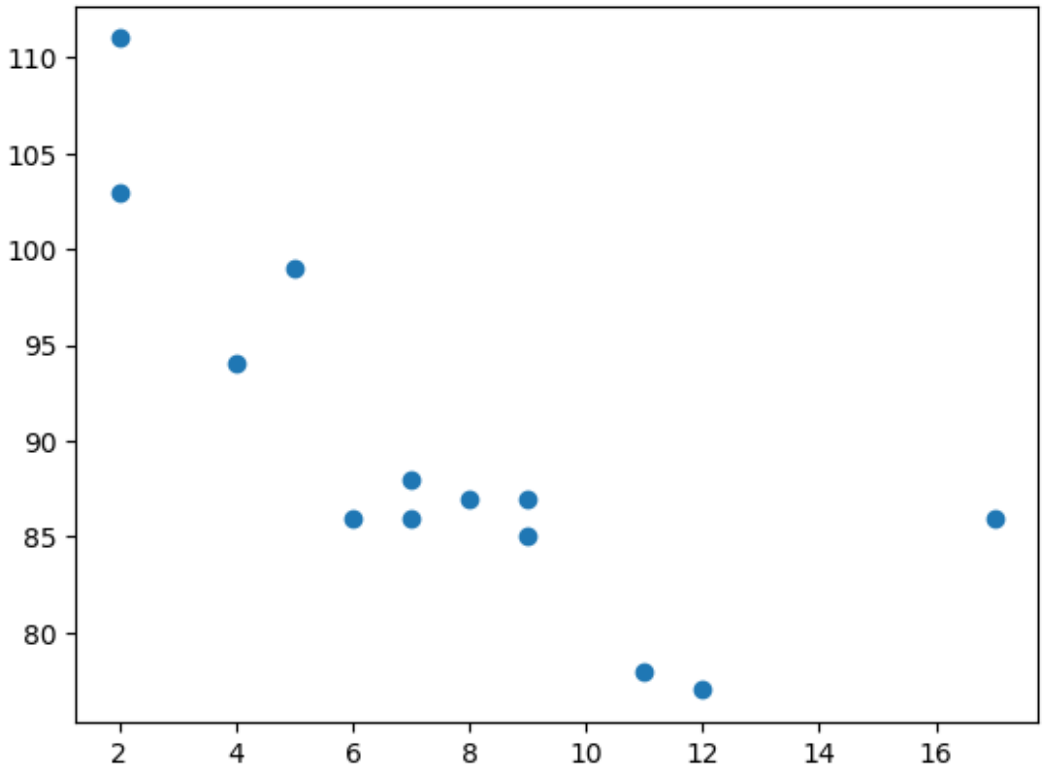
Resultado:



Finalmente tenemos el gráfico de dispersión que se crea con la función `scatter`. Es muy parecido a la función `plot` en cuanto a los parámetros que recibe pero solo dibuja los puntos en vez de unirlos con líneas.

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```

Resultado:



Cambiando las propiedades por defecto del gráfico

En Matplotlib los gráficos tienen una configuración por omisión. Cambiándolas podés configurar muchas propiedades del gráfico. Podés cambiar el tamaño de la figura, los DPI (viene de dots per inch, puntos por pulgada, y determina la resolución), el tamaño, color y estilo del trazo, las propiedades de los ejes y el cuadriculado, los textos y sus propiedades, etc.

Vamos a plotear las funciones *seno* y *coseno* en el mismo gráfico. Partiendo de la configuración básica, vamos a ir cambiando el gráfico paso por paso para que quede como queremos.

Primero hay que obtener los datos para graficar:

```
import numpy as np

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

No importa mucho si no se entienden las funciones lo importante es que ahora tenemos un array llamado `X` con 256 valores que van desde -π a +π (incluído). `C` tiene los valores del coseno (256 valores) y `S` tiene los valores del seno (256 valores). Podemos usar lo que aprendimos de Pandas y ponerlos en una tabla para verlos mejor.

```
import pandas as pd

df=pd.DataFrame({'x':X, 'cos(x)': C, 'sin(x)': S})
df
```

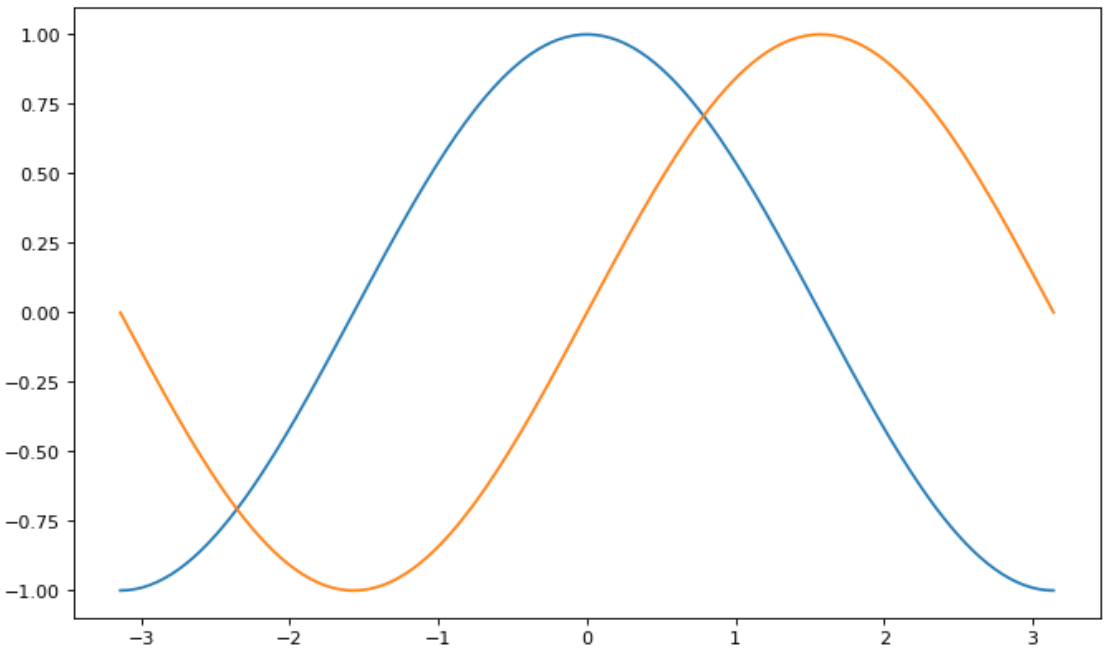
Resultado:

```
            x    cos(x)        sin(x)
0   -3.141593 -1.000000 -1.224647e-16
1   -3.116953 -0.999696 -2.463745e-02
2   -3.092313 -0.998786 -4.925994e-02
3   -3.067673 -0.997269 -7.385253e-02
4   -3.043033 -0.995147 -9.840028e-02
..        ...       ...           ...
251  3.043033 -0.995147  9.840028e-02
252  3.067673 -0.997269  7.385253e-02
253  3.092313 -0.998786  4.925994e-02
254  3.116953 -0.999696  2.463745e-02
255  3.141593 -1.000000  1.224647e-16

[256 rows x 3 columns]
```

Ahora podemos graficar ambas funciones en una misma imagen usando dos `plot` seguidos.

```
# Crea una figura nueva, de 8x6 pulgadas, con 80 puntos por pulgada. Se usa para
cambiar el tamaño de la imagen
plt.figure(figsize=(10, 6), dpi=80)

plt.plot(X, C)
plt.plot(X, S)

plt.show()
```

Resultado:

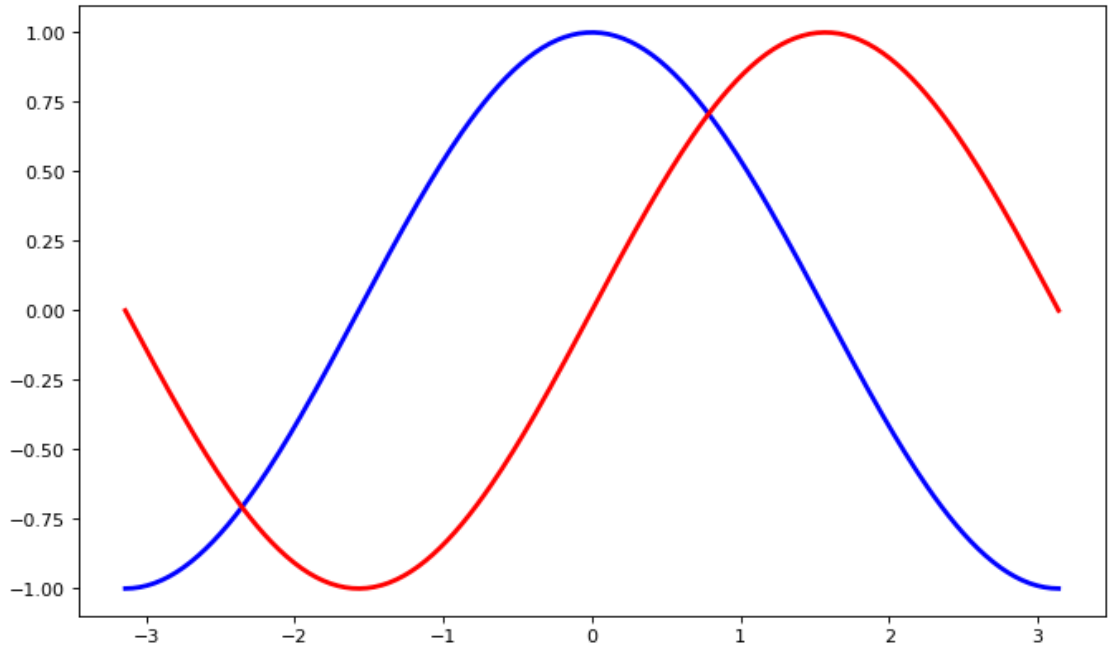


Cómo cambiar los colores y ancho de los trazos

Ahora vamos a modificar el gráfico para que quede un poco mejor. Primero, queremos trazar el coseno en azul y el seno en rojo, y ambos con una línea algo más gruesa. Además, vamos a cambiar un poco el tamaño de la figura para hacerla apaisada. Corré el siguiente código y compará el resultado con la figura anterior.

```
plt.figure(figsize=(10, 6), dpi=80)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
plt.show()
```

Resultado:

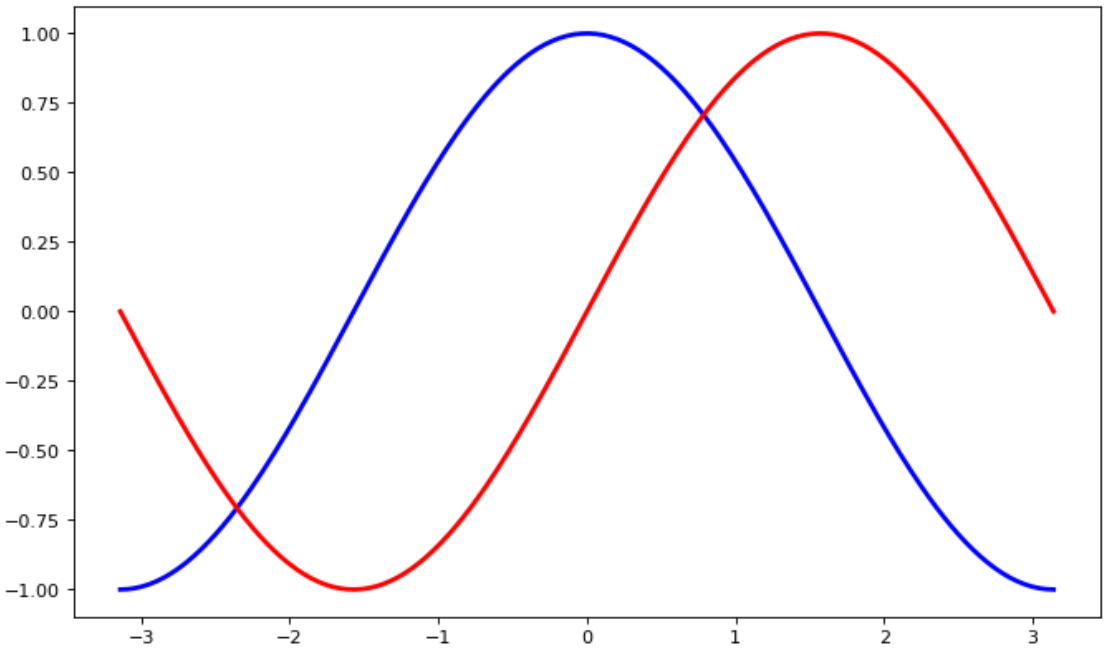


Límites de los ejes

El rango de valores de los ejes es un poco angosto y necesitamos más espacio alrededor para ver claramente todos los puntos.

```
plt.figure(figsize=(10, 6), dpi=80)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.ylim(C.min() * 1.1, C.max() * 1.1)
plt.show()
```

Resultado:

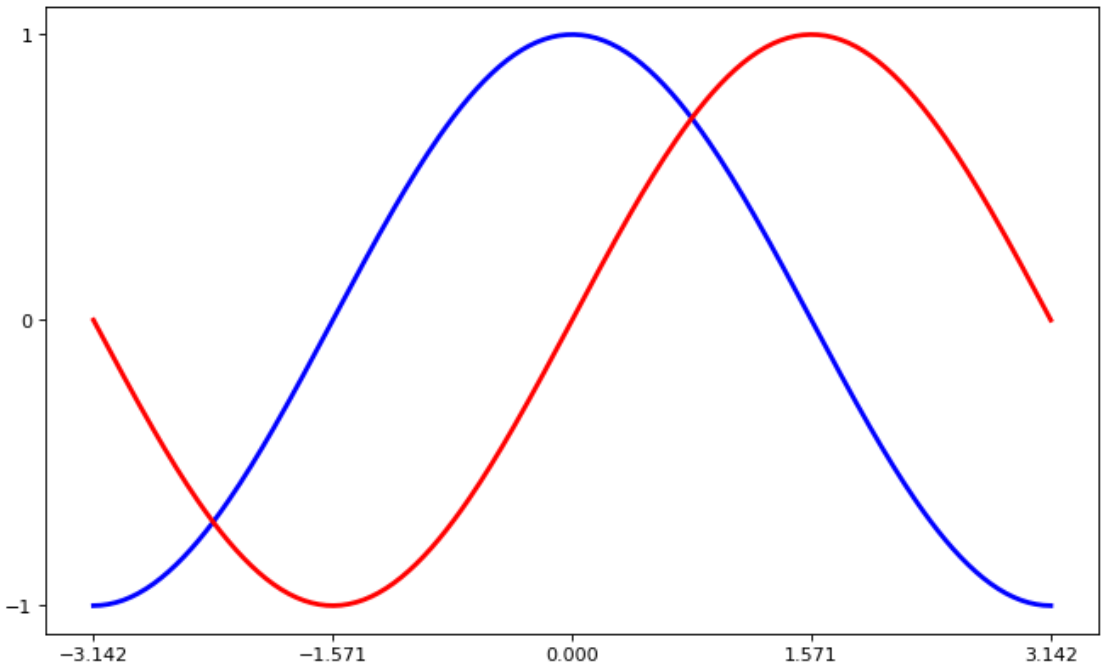


Marcas en los ejes

Así como están, las marcas sobre los ejes no son lo más útil. Sería bueno destacar los valores interesantes para seno y coseno (+/-π,+/-π/2). Cambiémoslos para mostrar únicamente esos valores.

```
plt.figure(figsize=(10, 6), dpi=80)

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.yticks([-1, 0, +1])
plt.show()
```

Resultado:



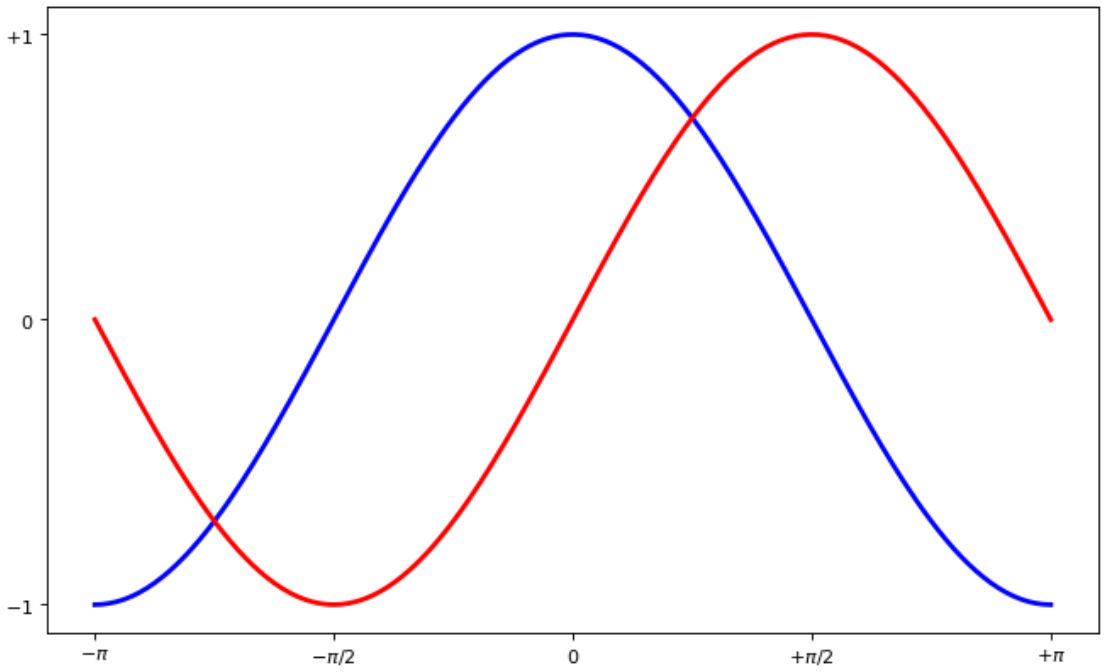
Texto de las marcas en los ejes

Las marcas en los ejes ahora están donde los queremos, pero el texto no es muy explícito. Aunque podemos darnos cuenta que 3.142 es π sería mejor dejarlo explícito.

Al definir un valor para las marcas en los ejes podemos proveer un texto en la segunda lista de argumentos para usar como etiqueta. Fijate que vamos a usar *LaTeX* para hacer que los símbolos tengan mejor pinta

```
plt.figure(figsize=(10, 6), dpi=80)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
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],
           [r'$-1$', r'$0$', r'$+1$'])
plt.show()
```

Resultado:

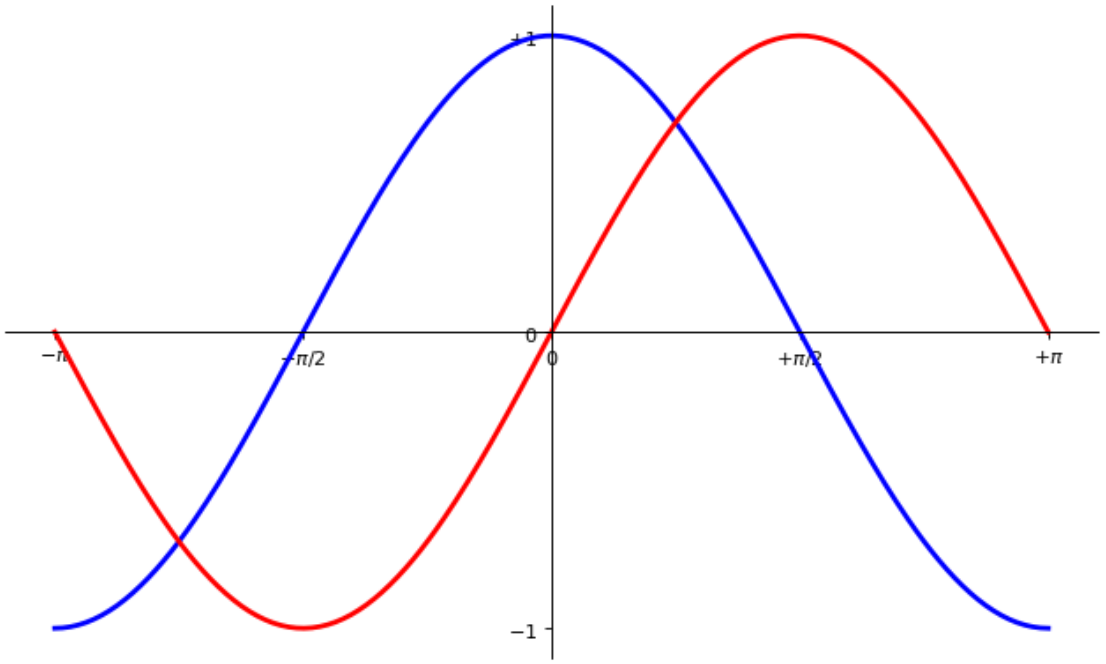


Movamos el contorno

El contorno es el conjunto de líneas que delimitan el área de graficación y que unen todas las marcas en los ejes. Podemos ubicarlas en cualquier posición y, hasta ahora, han estado en el extremo de cada eje. Cambiemos eso, así las ubicamos en el centro. Como hay cuatro (arriba, abajo, izquierda y derecha) vamos a esconder dos de ellas dándoles color none y vamos a mover la de abajo y la de la izquierda a la posición 0 del espacio de coordenadas.

```
plt.figure(figsize=(10, 6), dpi=80)

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
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],
           [r'$-1$', r'$0$', r'$+1$'])
ax = plt.gca()  # gca es 'get current axis' ó 'tomar eje actual'
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
plt.show()
```

Resultado:



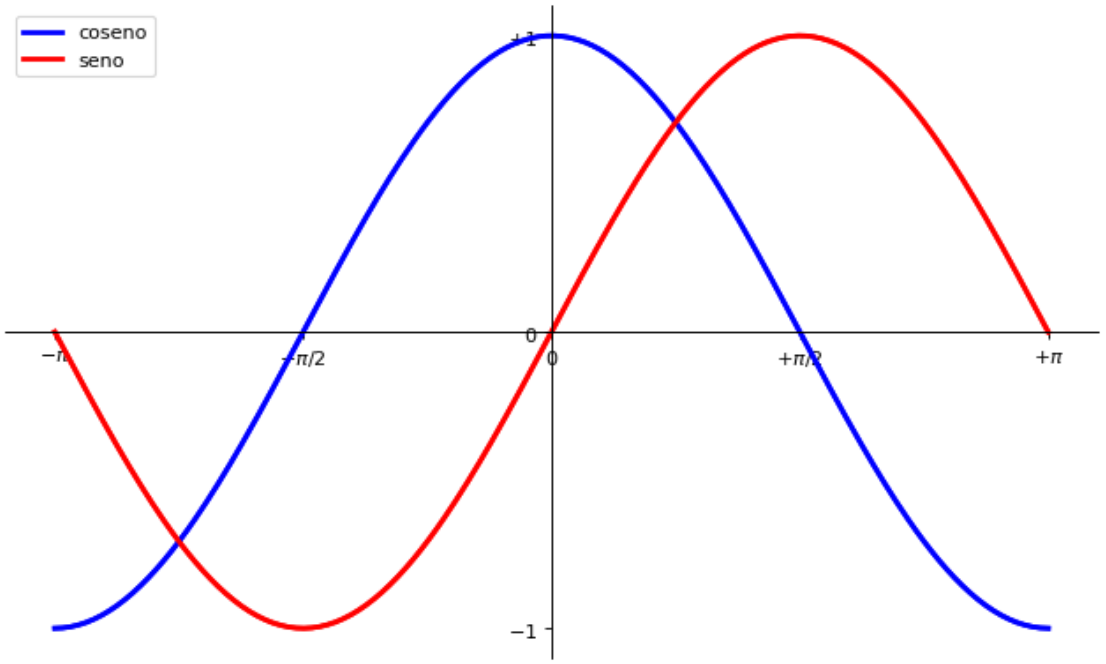
Pongámosle un título

Pongámosle nombres a los trazos al gráfico en la esquina superior izquierda. Para esto alcanza con agregar a la instrucción `plot` la palabra clave `label` y ese texto será usado para el recuadro con los nombres.

```
plt.figure(figsize=(10, 6), dpi=80)

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
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],
           [r'$-1$', r'$0$', r'$+1$'])
ax = plt.gca()  # gca es 'get current axis' ó 'tomar eje actual'
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="coseno")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-", label="seno")
plt.legend(loc='upper left')
```

Resultado:

```
<matplotlib.legend.Legend at 0x7f7fb078faf0>
```



Anotando cosas

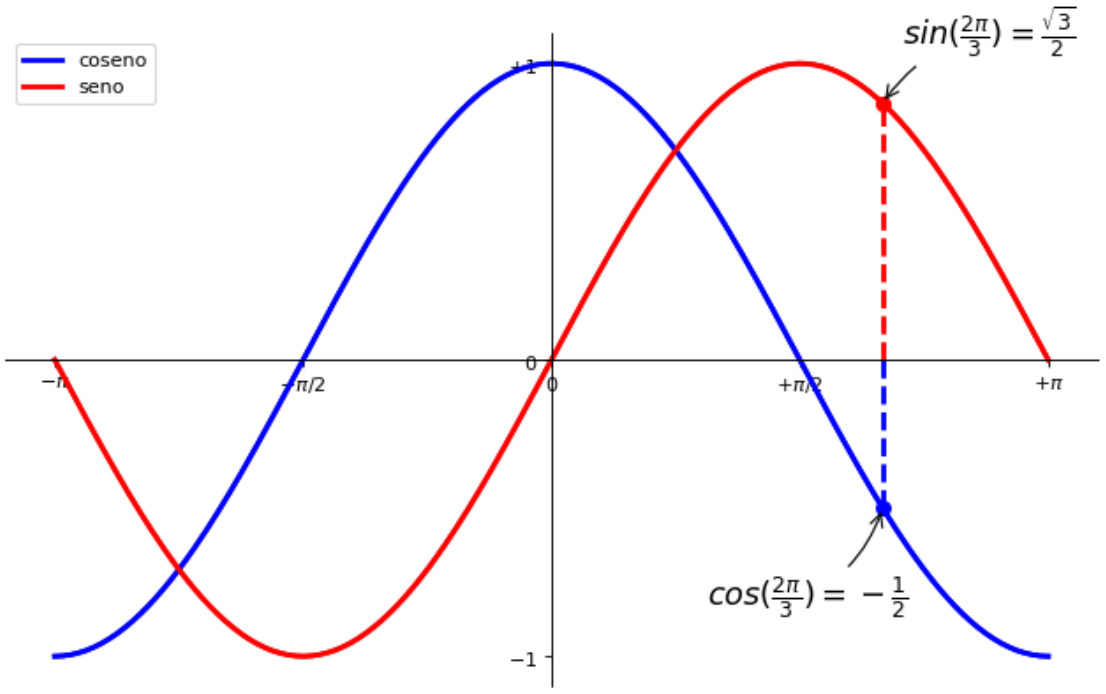
Vamos a marcar algunos puntos interesantes usando el comando `annotate`. Elegimos el valor $2\pi/3$ y queremos marcar tanto el seno como el coseno. Vamos a dibujar una marca en la curva y una línea recta punteada. Además, vamos a usar `annotate` para mostrar texto y una flecha para destacar el valor de las funciones.

```
plt.figure(figsize=(10, 6), dpi=80)

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
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],
           [r'$-1$', r'$0$', r'$+1$'])
ax = plt.gca()  # gca es 'get current axis' ó 'tomar eje actual'
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="coseno")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-", label="seno")
plt.legend(loc='upper left')

t = 2 * np.pi / 3
plt.plot([t, t], [0, np.cos(t)], color='blue', linewidth=2.5, linestyle="--")
plt.scatter([t, ], [np.cos(t), ], 50, color='blue')
```

```
plt.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$',
             xy=(t, np.cos(t)), xycoords='data',
             xytext=(-90, -50), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t, t],[0, np.sin(t)], color='red', linewidth=2.5, linestyle="--")
plt.scatter([t, ],[np.sin(t), ], 50, color='red')

plt.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
             xy=(t, np.sin(t)), xycoords='data',
             xytext=(+10, +30), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```

Resultado:

```
Text(10, 30, '$sin(\\frac{2\\pi}{3})=\\frac{\\sqrt{3}}{2}$')
```



Unificando todo

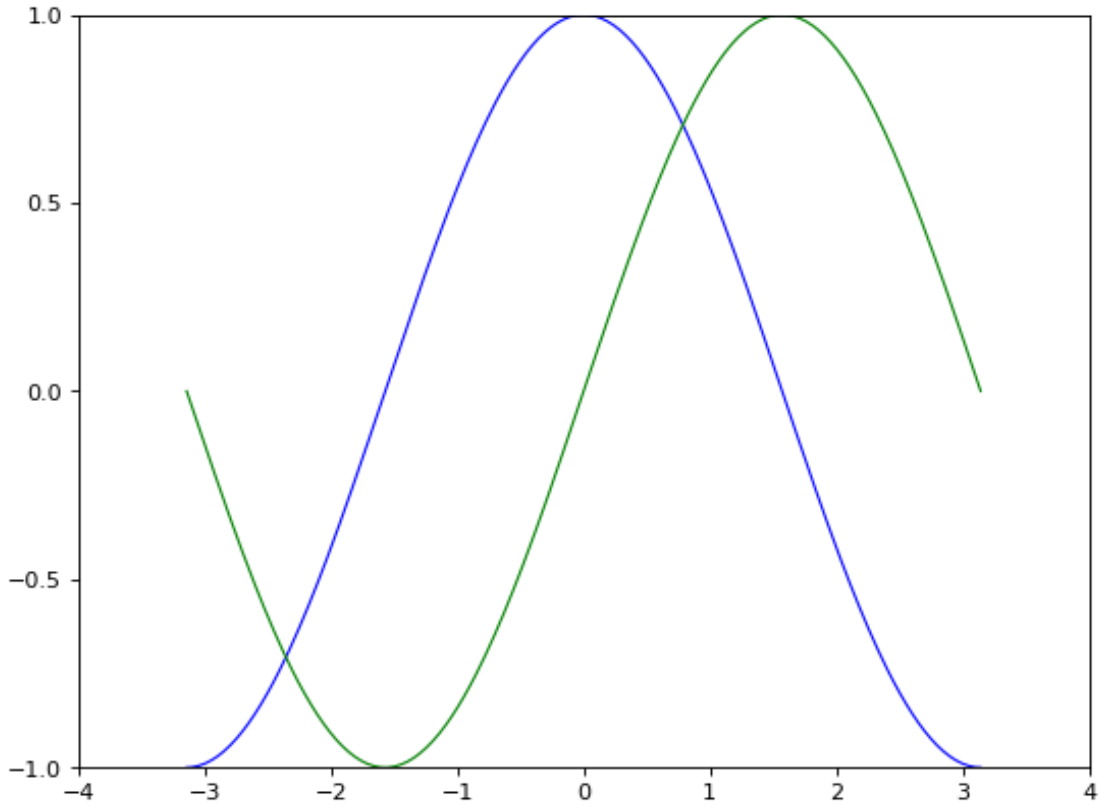
En el siguiente script, hemos explicitado y comentado todas las propiedades de una figura que influyen en la apariencia de un gráfico.

Cada propiedad se configuró a un valor típico y cercano al valor por omisión. Podés modificarlos y jugar con ellos para ver sus efectos sobre el gráfico. Sobre propiedades y estilos de las líneas hablaremos luego.

```
# Crea una figura nueva, de 8x6 pulgadas, con 80 puntos por pulgada
plt.figure(figsize=(8, 6), dpi=80)
```

```
# Crea un nuevo subplot, en una grilla de 1x1
plt.subplot(1, 1, 1)

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

# Plotea el coseno con una línea azul contínua de ancho 1 (en pixeles)
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")

# Plotea el seno con una línea verde contínua de ancho 1 (en pixeles)
plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")

# Rango del eje x
plt.xlim(-4.0, 4.0)

# Ponemos marcas (ticks) en el eje x
plt.xticks(np.linspace(-4, 4, 9))

# Rango del eje y
plt.ylim(-1.0, 1.0)

# Ponemos marcas (ticks) en el eje y
plt.yticks(np.linspace(-1, 1, 5))

# Podemos grabar el gráfico (con 72 dpi)
# plt.savefig("ejercicio_2.png)", dpi=72)
plt.show()
```

Resultado:



Los gráficos que genera matplotlib son muy flexibles, te dejamos un machete resumiendo las variaciones más usuales.

A continuación presentamos detalles técnicos de esta biblioteca tan útil. No hace falta que te los aprendas (igual te los vas a olvidar), ni que pruebes todas las combinaciones. Podés volver a esta página o a la documentación cuando lo necesites. Iguál mirá los ejercicios al final de esta sección, te pediremos que entregues el segundo.