Matplotlib

Unidad 6

Apunte de cátedra

Pensamiento computacional (90)

.UBA XXI

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, 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.

```
import matplotlib.pyplot as plt
```

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

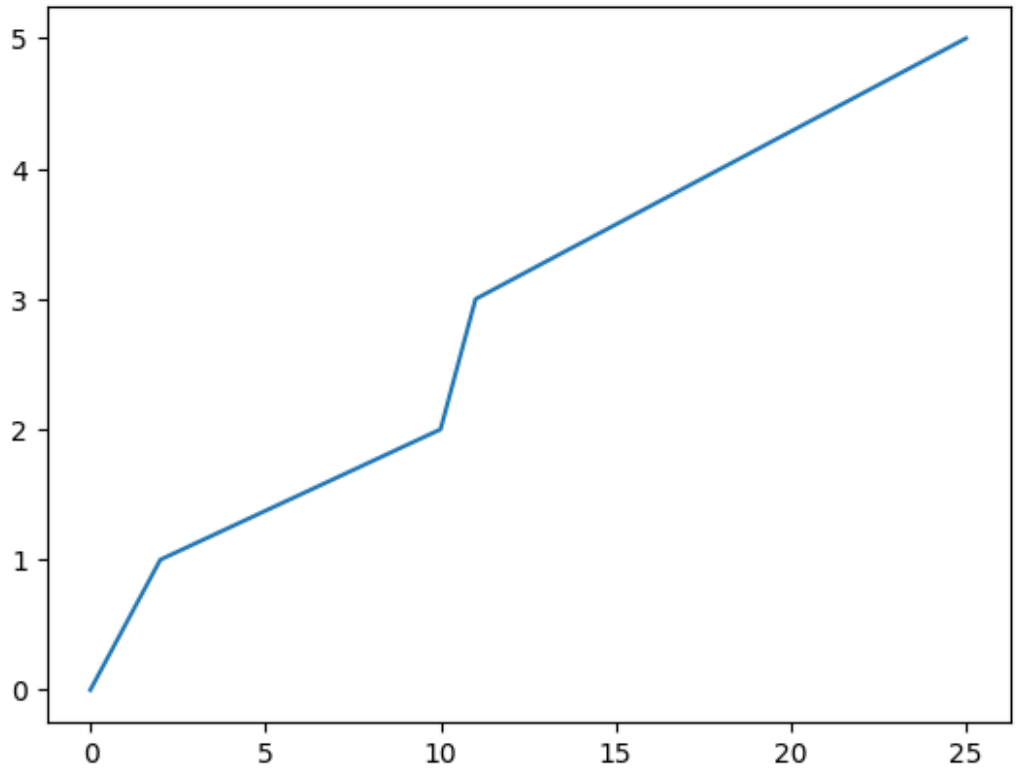
La función `**plot**` recibe 2 vectores que tienen que ser del mismo tamaño que representan una serie de puntos en el plano cartesiano. El primer arreglo son las coordenadas X y el segundo son las coordenadas Y. Finalmente lo que hace la función es unir primero esos elementos para crear puntos (coordenada a coordenada de los vectores, toma un valor de `x` y otro de `y` y arma un punt), y luego unir todos estos puntos con líneas para que en el dibujo podamos ver el gráfico de la función.

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

```
# Grafico elemental

x = [0,2,10,11,18,25]

y = [0,1,2,3,4,5]

fig = plt.figure()

plt.plot(x, y)
```

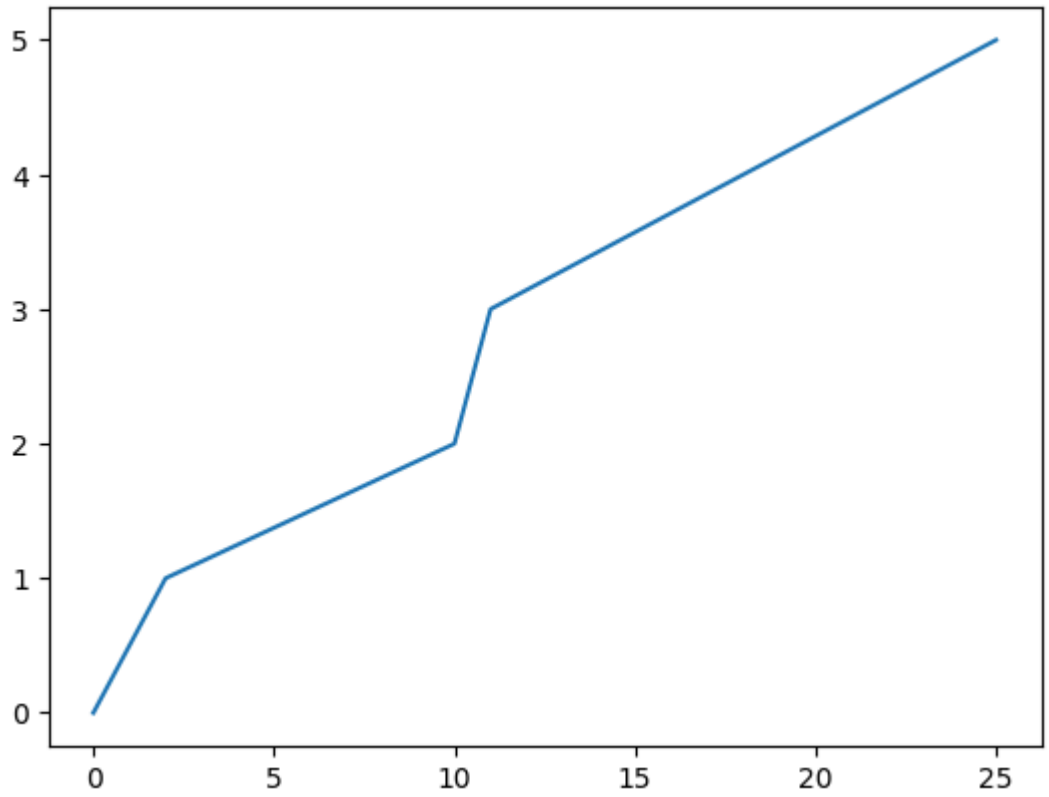
```
plt.show()
```

Resultado:



Para los valores de x e y definidos anteriormente, también se podría haber graficado utilizando el siguiente código:

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

ax.plot(x, y)

plt.show()
```



Pero, ¿cuál debería usar? Bueno, en realidad depende de lo que quieras hacer, por eso a continuación, vamos a dar un poco de detalle de lo que está ocurriendo en cada línea, para que así puedas elegir qué es lo mejor para vos:

Por un lado

```
plt.figure()
```

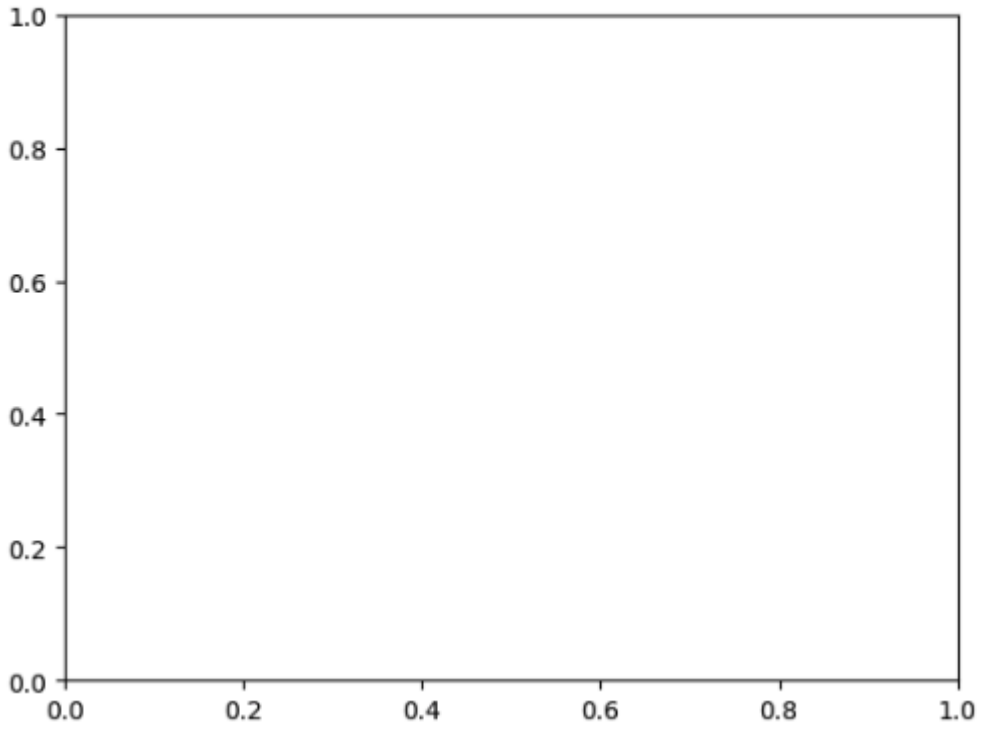
```
[ ] plt.figure()

    <Figure size 640x480 with 0 Axes>
    <Figure size 640x480 with 0 Axes>
```

Por el otro

```
plt.subplots()
```

```
[ ] plt.subplots()
```

```
(<Figure size 640x480 with 1 Axes>, <Axes: >)
```



Si bien no se puede ver un gráfico en ninguno de los outputs, analicemos lo que nos imprime:

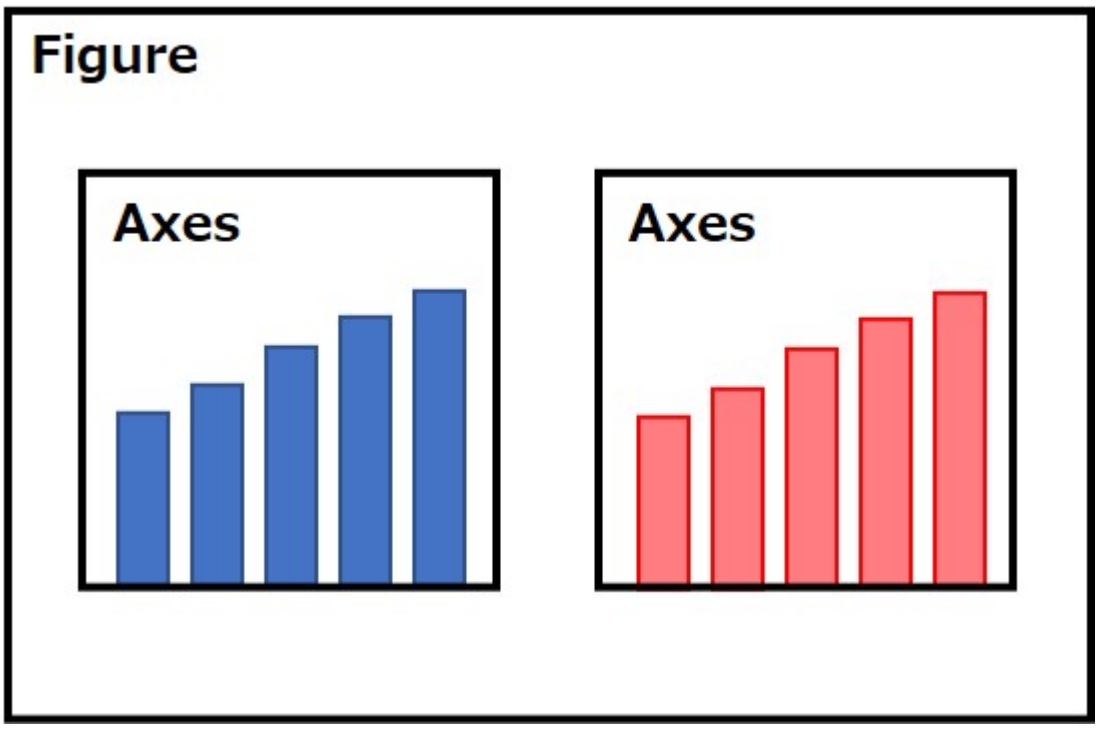
1. `plt.figure()` crea una **figura pero sin axes** (por eso dice <Figure size 640x480 with 0 Axes> , se tiene una figura y 0 aces).
2. `plt.subplots()` permite **crear ambos** (por eso dice <Figure size 640x480 with 1 Axes>, <Axes: >, hay figura y axes).

Note que se ha seguido una convención al nombrarse la **figura** como `fig` y los **axes** como `ax`.

Ok, perfecto, pero... **¿qué es una figura y un axes?**

Una figura es el marco que delimita la zona donde se trazan los gráficos, mientras que los axes, son lo que llamamos comunmente gráficos, es decir, son las áreas donde los puntos se pueden especificar en términos de coordenadas.

Por lo tanto, una figura puede contener muchos axes, pero un axes determinado sólo puede estar contenido en una única figura.

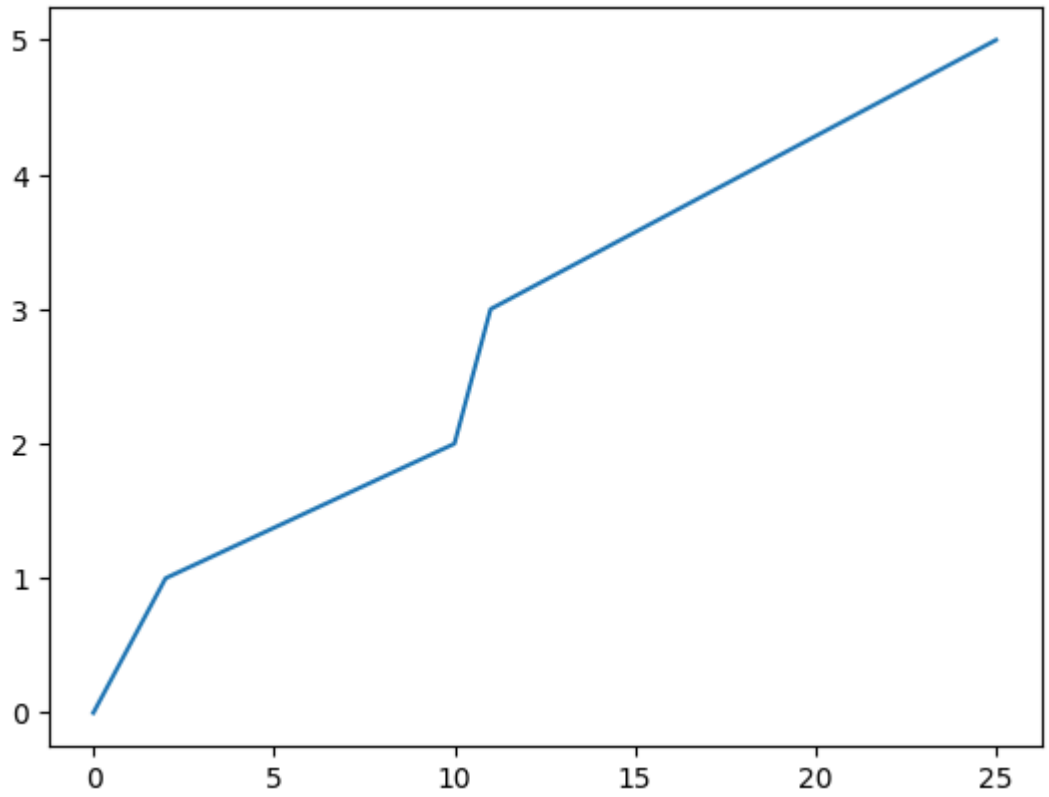


¡Ojo! No confundir axes con axis.

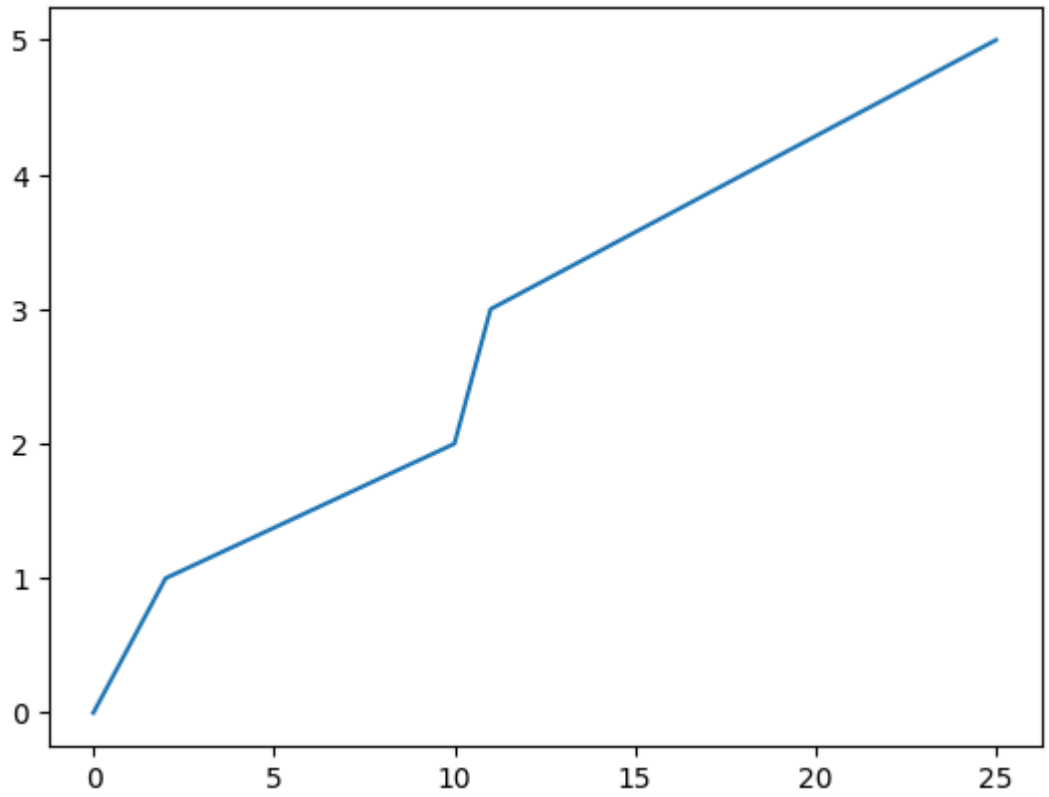
Los Axis son los ejes cartesianos que se encargan de establecer los límites, la escala y las dimensiones del gráfico: un axes puede tener 2 Axis, si es un gráfico plano, o 3, si es un gráfico en 3D.

Entonces, con lo aprendido hasta el momento, volvamos a revisar las líneas de código para los gráficos anteriores:

```
# Grafico elemental explicado:

fig = plt.figure() # Se crea una figura vacía sin Axes


plt.plot(x, y) # Agregar ploteo (gráfico de linea)

plt.show() # Mostrar
```




```
# Grafico elemental explicado:

fig, ax = plt.subplots() # Se crea una figura con un único Axes.


ax.plot(x, y) # Agregar los ploteos individuales (gráfico de linea)

plt.show() # Mostrar
```

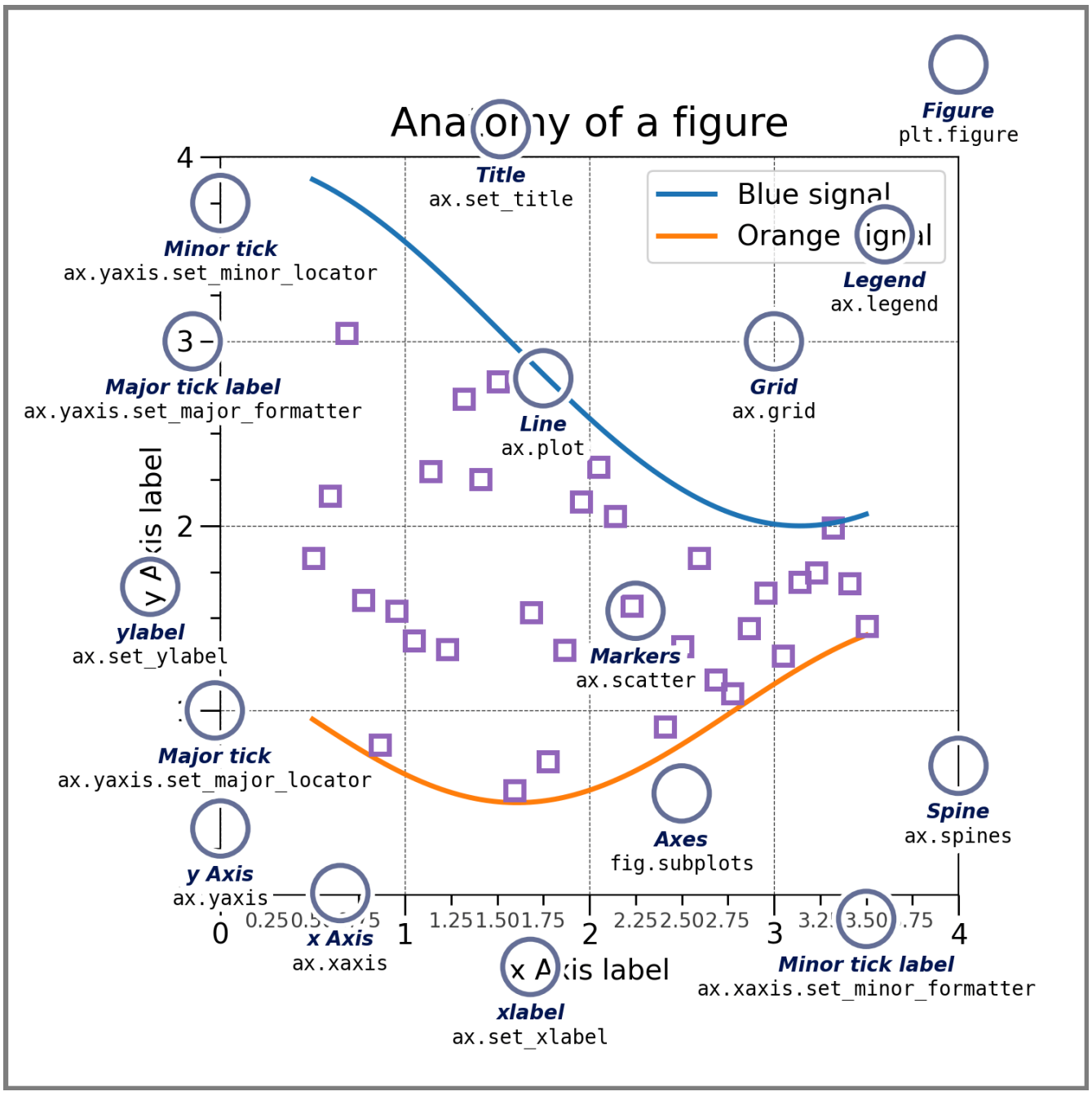


Entonces, **`plt.plot()`** es un método más amigable para principiantes ya que es más conciso y resulta muy útil cuando simplemente se desea crear un gráfico para verificar resultados rápidamente.

Ahora, si se desean gráficos más complejos o con un ajuste fino como los que veremos al final de este apunte, necesitaremos un enfoque más flexible, como **`plt.subplots()`**.

Partes de una Figura

Esta imagen, fue obtenida de la referencia de matplotlib (https://matplotlib.org/stable/tutorials/introductory/quick_start.html) y resume de manera fácil y visual los detalles que podemos agregarle a las figuras creadas. A continuación, vamos a mostrar cómo lo hacemos, utilizando el gráfico anterior:



Cambiar el Aspecto de los Gráficos

Para diferenciar distintas curvas o simplemente, para modificar los gráficos según nuestros gustos personales, se pueden definir los distintos parámetros dentro de `ax.plot()`, estableciendo el tipo de línea y puntos, el grosor, el color, etc:

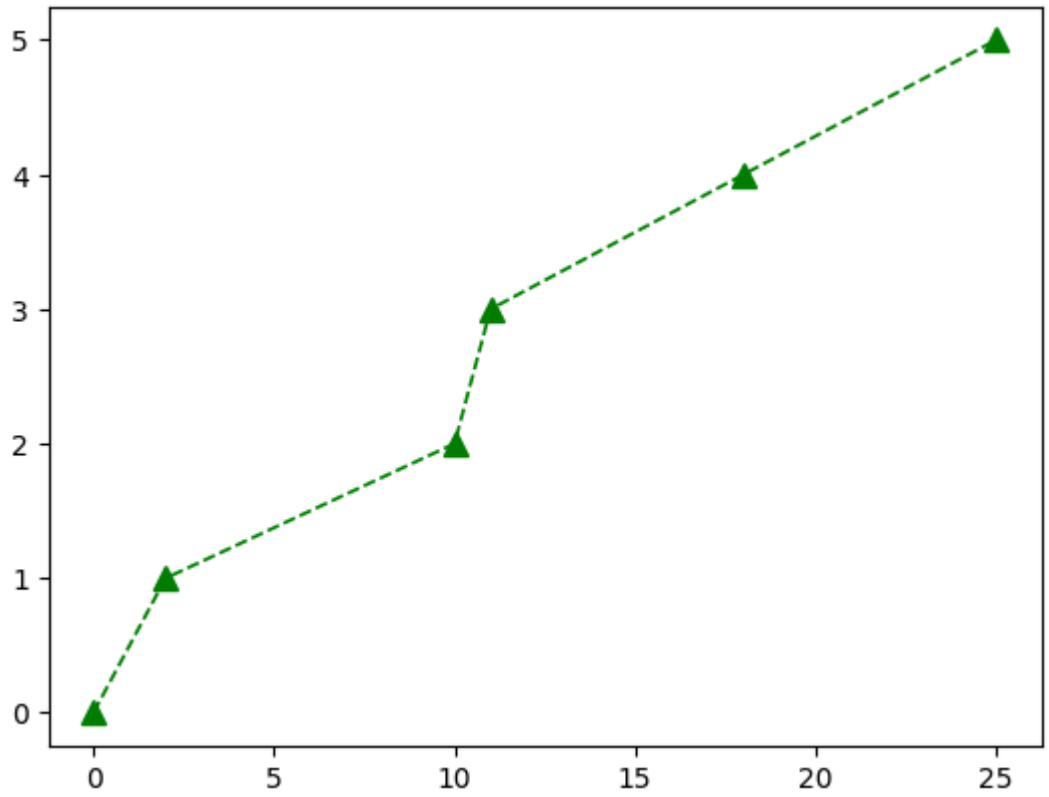
- **color =** nombre del color, por ejemplo: `'blue'`, `'green'`, `'red'`, etc.
- **marker =** forma de los puntos marcadores, por ejemplo: `'^'`, `'o'`, `'v'`, etc.

- **linestyle =** estilo de línea, por ejemplo: `'solid'`, `'dashed'`, `'dotted'` o sus equivalentes:`'-'`, `'--'`, `':'`, entre otros.
- **markersize, linewidth =** con un número, establecemos el tamaño del marcador y el espesor de la línea.

Notar que si no le asignamos un valor, se establecen los predefinidos.

Para ver las múltiples opciones disponibles, les dejamos el siguiente link de consulta: https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

```
x = [0,2,10,11,18,25] # Tiempo (min)

y = [0,1,2,3,4,5] # Distancia (m)


fig, ax = plt.subplots()


ax.plot(x, y, color='green', marker='^', linestyle='--', markersize=8,
linewidth=1.2)

plt.show()
```

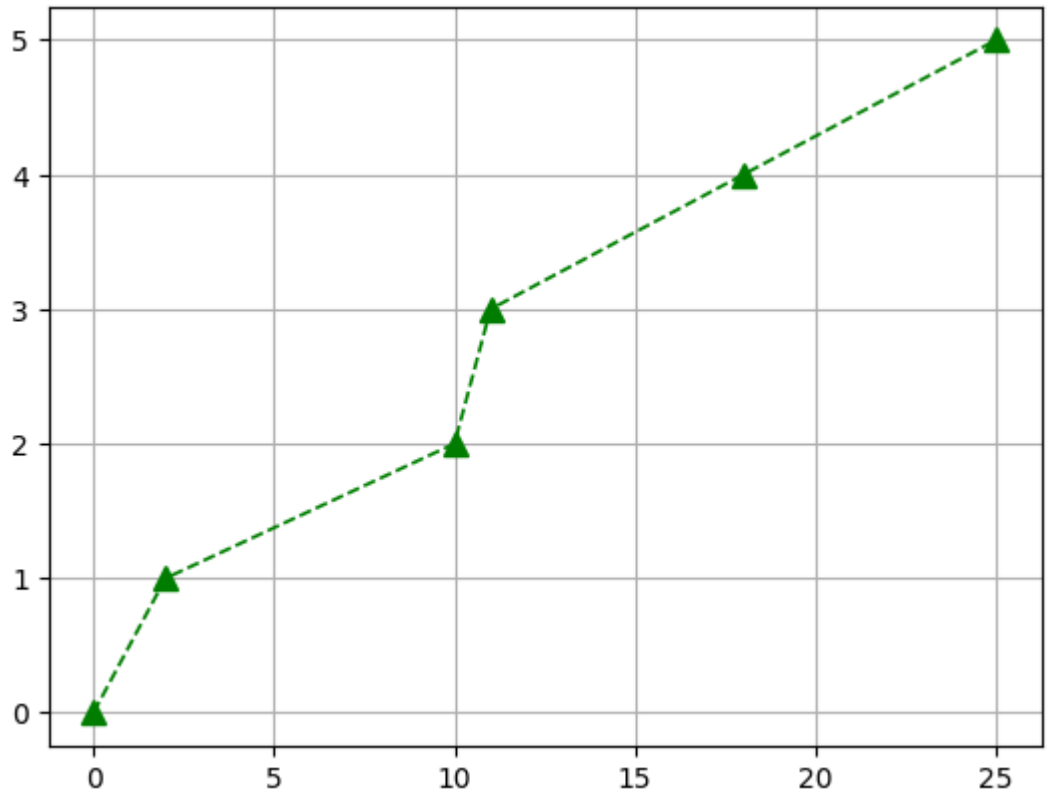


Grilla o Cuadrícula

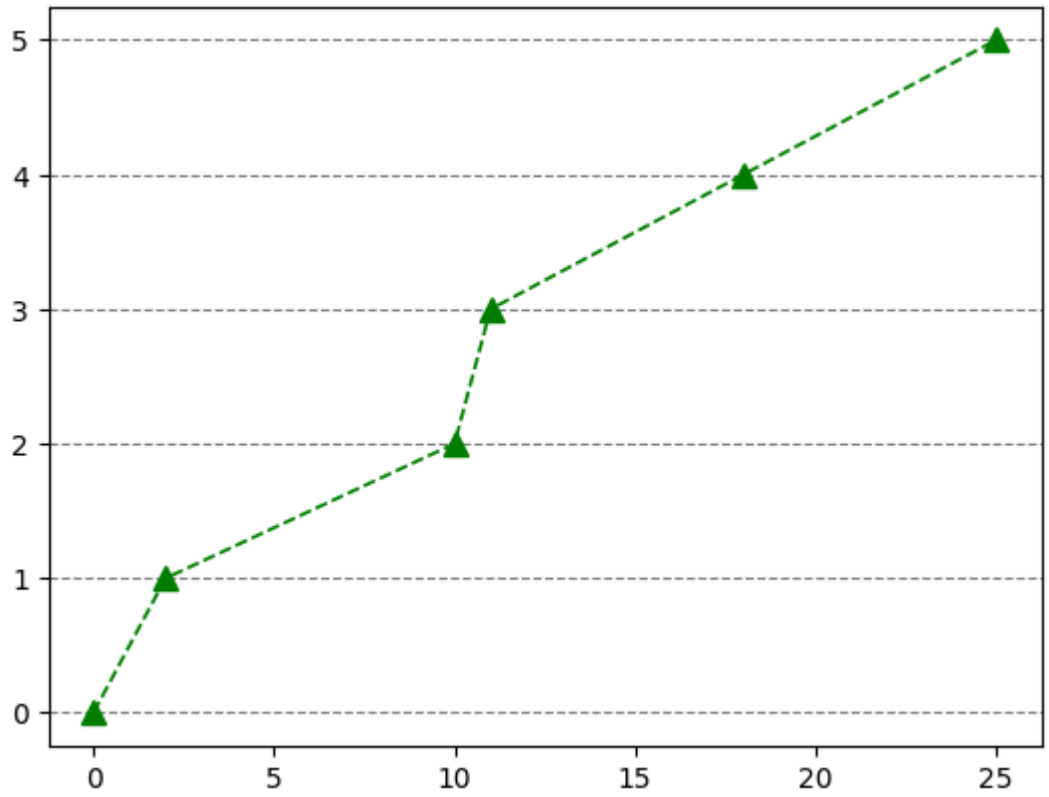
Para leer facilmente cada punto, podemos agregar una cuadrícula usando `ax.grid()`.

Si a esta deseamos modificarla, como por ejemplo, el color, el estilo de línea, o si sólo deseamos que se vea en uno de los ejes, podemos indicarlo utilizando parámetros muy similares a los vistos anteriormente pero en la funcion `ax.grid()`.

```
x = [0,2,10,11,18,25] # Tiempo (min)

y = [0,1,2,3,4,5] # Distancia (m)

fig, ax = plt.subplots()

ax.plot(x, y, color='green', marker='^', linestyle='--', markersize=8,
linewidth=1.2)

# Grilla preestablecida

ax.grid()

plt.show()
```




```
#Gráfica con la grilla preestablecida
x = [0,2,10,11,18,25] # Tiempo (min)
y = [0,1,2,3,4,5] # Distancia (m)

fig, ax = plt.subplots()

ax.plot(x, y, color='green', marker='^', linestyle='--', markersize=8,
linewidth=1.2)

#Grilla modificada
ax.grid(axis = 'y', color = 'gray', linestyle = 'dashed')
plt.show()
```


5
4
3
2
1
0
0
5
10
15
20
25

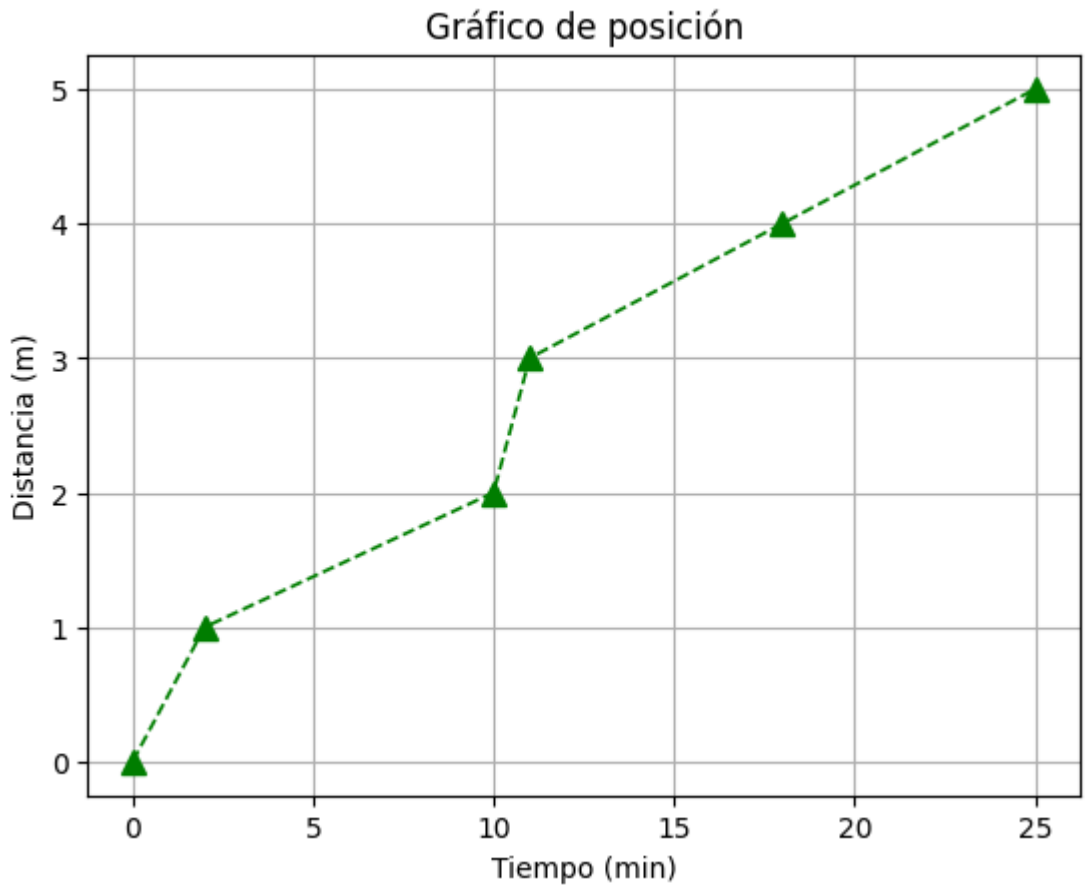
Títulos

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

Eso se hace con las funciones `ax.set_xlabel()`, `ax.set_ylabel()` y `ax.set_title()`. Cada una recibe un string que se usará como etiqueta del eje X, etiqueta del eje Y o título, respectivamente.

Siendo los valores de **x** se trataba del tiempo medido en minutos y los de **y** de una distancia en metros, entonces:

```
x = [0,2,10,11,18,25] # Tiempo (min)
y = [0,1,2,3,4,5] # Distancia (m)

fig, ax = plt.subplots()

ax.plot(x, y, color='green', marker='^', linestyle='--', markersize=8,
linewidth=1.2)

# Mostrar el título del gráfico
ax.set_title("Gráfico de posición")

# Mostrar el título de los ejes
ax.set_xlabel('Tiempo (min)')
ax.set_ylabel('Distancia (m)')

# Grilla preestablecida
ax.grid()
plt.show()
```

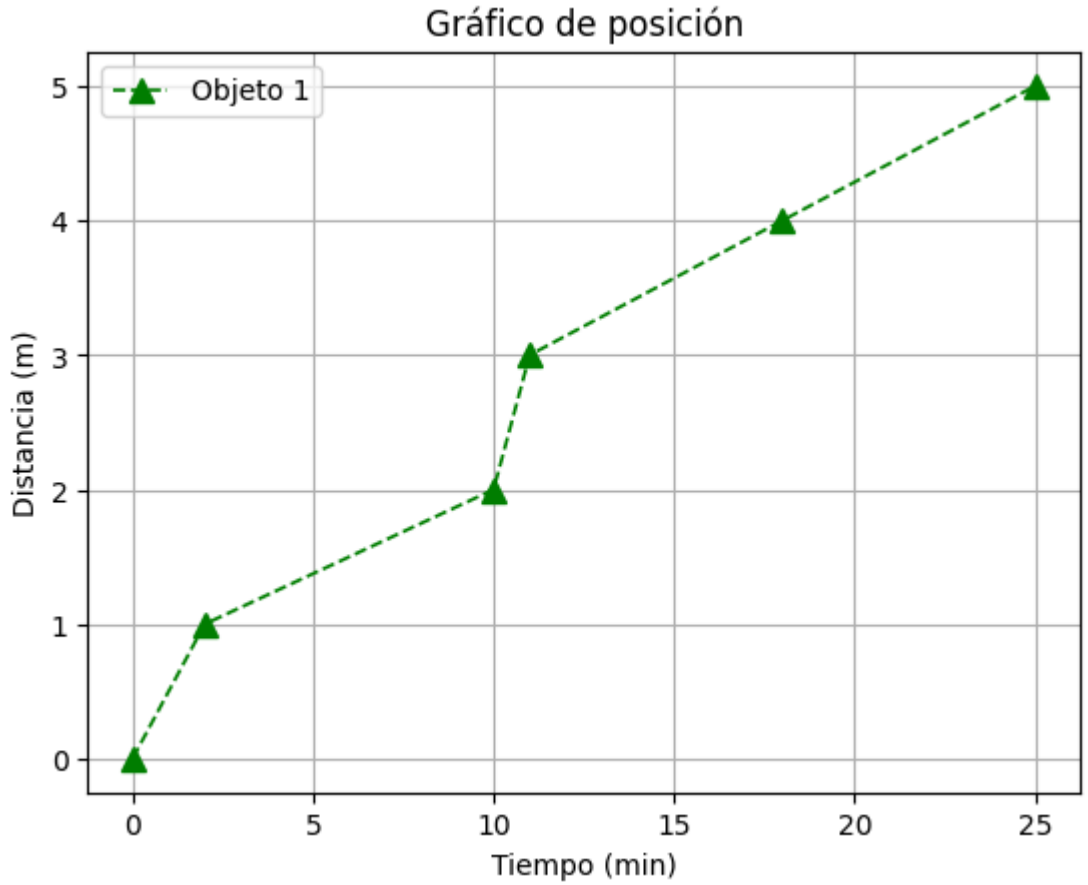


Referencias

El gráfico con el que estamos trabajando sólo cuenta con una línea: en caso de tener más de una, el uso de referencias es escencial para lograr el entendimiento del mismo. Para ellos, dentro de `ax.plot()` se debe definir la referencia como `label`. Luego se coloca `ax.legend()`

```
x = [0,2,10,11,18,25] # Tiempo (min)

y = [0,1,2,3,4,5] # Distancia (m)

fig, ax = plt.subplots()


ax.plot(x, y, label='Objeto 1', color='green', marker='^', linestyle='--',
markersize=8, linewidth=1.2)

# Mostrar el título del gráfico

ax.set_title("Gráfico de posición")

# Mostrar el título de los ejes
```

```
ax.set_xlabel('Tiempo (min)')

ax.set_ylabel('Distancia (m)')

# Agregar la refencia

ax.legend()

# Grilla preestablecida

ax.grid()

plt.show()
```



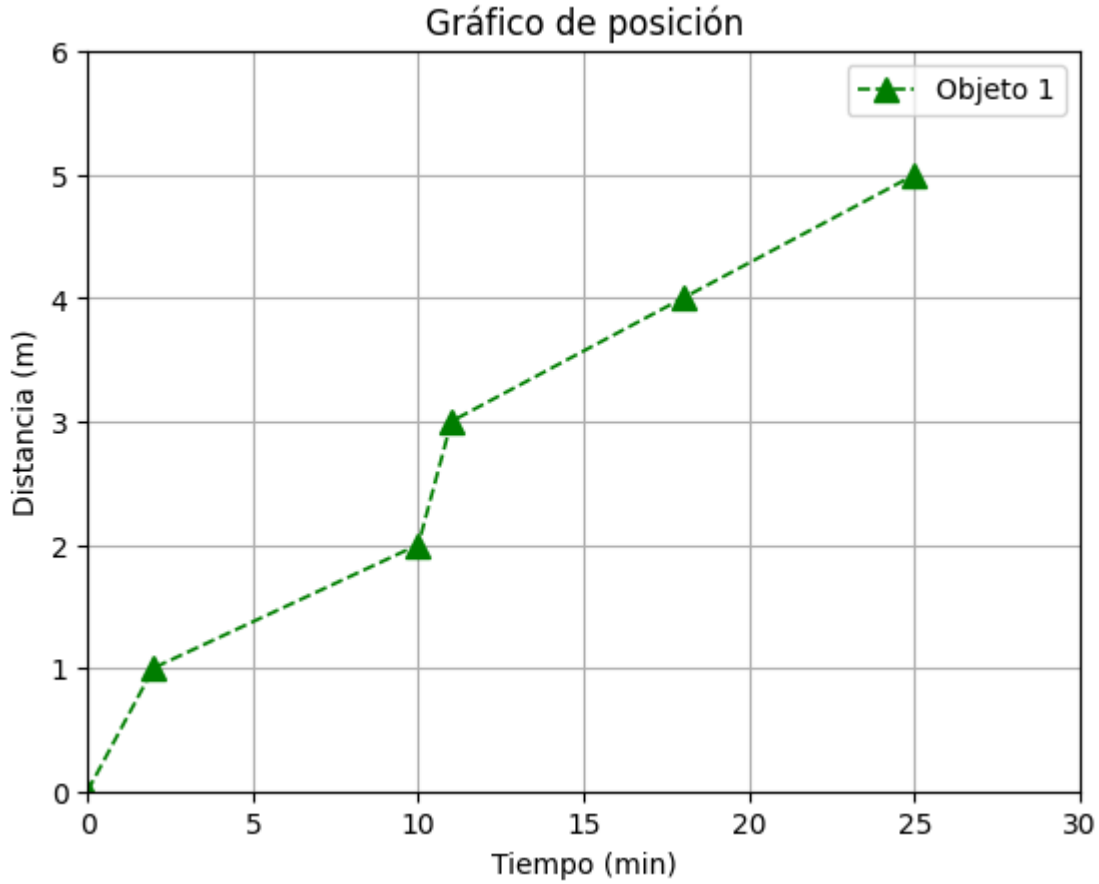
Características de los Ejes

Como podemos identificar en los gráficos anteriores, python decidió las características de los ejes:

- El eje x: se extiende del 0 a 25, de 5 en 5.
- El eje y: se extiende del 0 a 5, de 1 en 1.

Podemos establecer los limites del eje x e y usando `ax.set_xlim()` y `ax.set_ylim()` respectivamente.

```
x = [0,2,10,11,18,25] # Tiempo (min)
y = [0,1,2,3,4,5] # Distancia (m)

fig, ax = plt.subplots()

ax.plot(x, y, label='Objeto 1', color='green', marker='^', linestyle='--',
markersize=8, linewidth=1.2)

# Mostrar el título del gráfico
ax.set_title("Gráfico de posición")

# Mostrar el título de los ejes
ax.set_xlabel('Tiempo (min)')
ax.set_ylabel('Distancia (m)')

# Establecer los límites de los ejes
ax.set_xlim(0, 30)
ax.set_ylim(0, 6)

# Agregar la refencia
ax.legend()

# Grilla preestablecida
ax.grid()
plt.show()
```



Tipos de Gráficos

A continuación, vamos a enumerar los tipos de gráficos más comunes, las funciones que son necesarias para crearlos y cuándo debemos utilizar a cada uno de ellos.

Recuerden que todo lo aprendido para modificar el aspecto del gráfico, como agregar títulos, cuadrículas, límites a los ejes, etc., se puede hacer también en estas figuras. Además, de manera análoga a como establecíamos los valores de los parámetros de ax.plot(), podemos hacerlo en las nuevas funciones que vamos a aprender a continuación.

Gráfico de Línea

El gráfico de línea permite visualizar cambios en los valores lo largo de un rango continuo (tendencias), como puede ser el tiempo o la distancia. Para crearlo, se utiliza la función `ax.plot()`, como vimos anteriormente:

```
x = [0,2,10,11,18,25]
```

```
y = [0,1,2,3,4,5]


fig, ax = plt.subplots()


ax.plot(x, y)

plt.show()
```

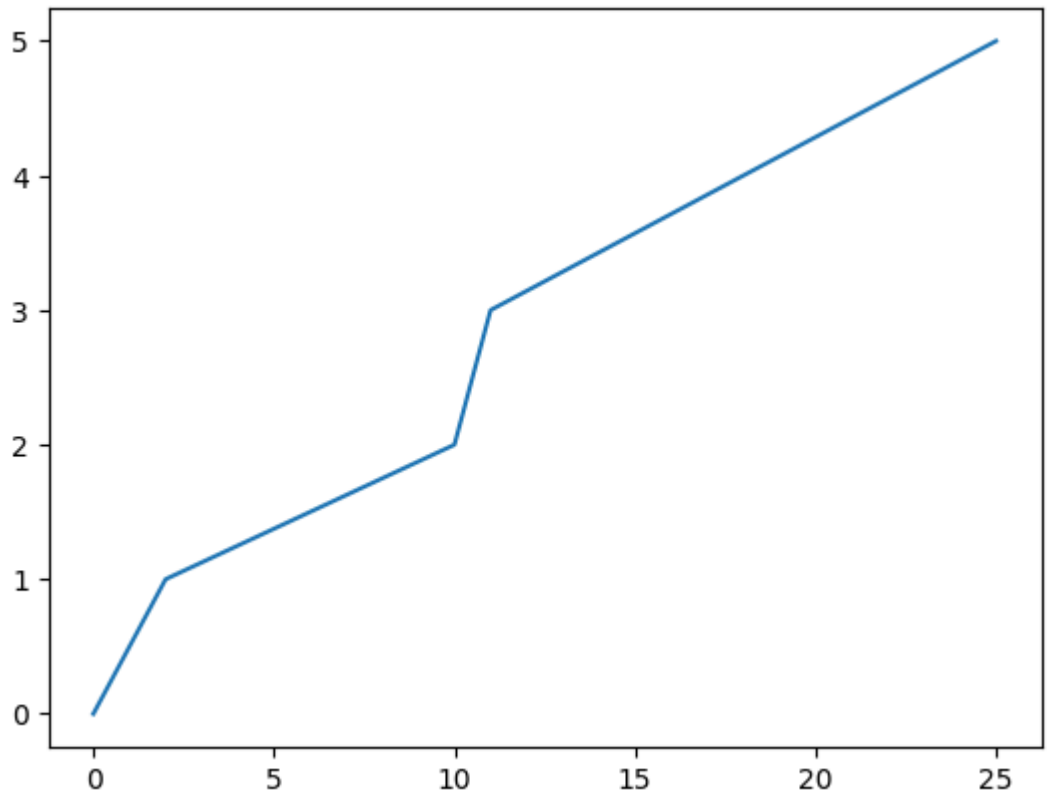


Gráfico de Dispersión o Puntos

El gráfico de dispersión o puntos permite visualizar la relación entre las variables. Para crearlo, se utiliza la función `ax.scatter()`:

```
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]


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

```
ax.scatter(x, y)

plt.show()
```

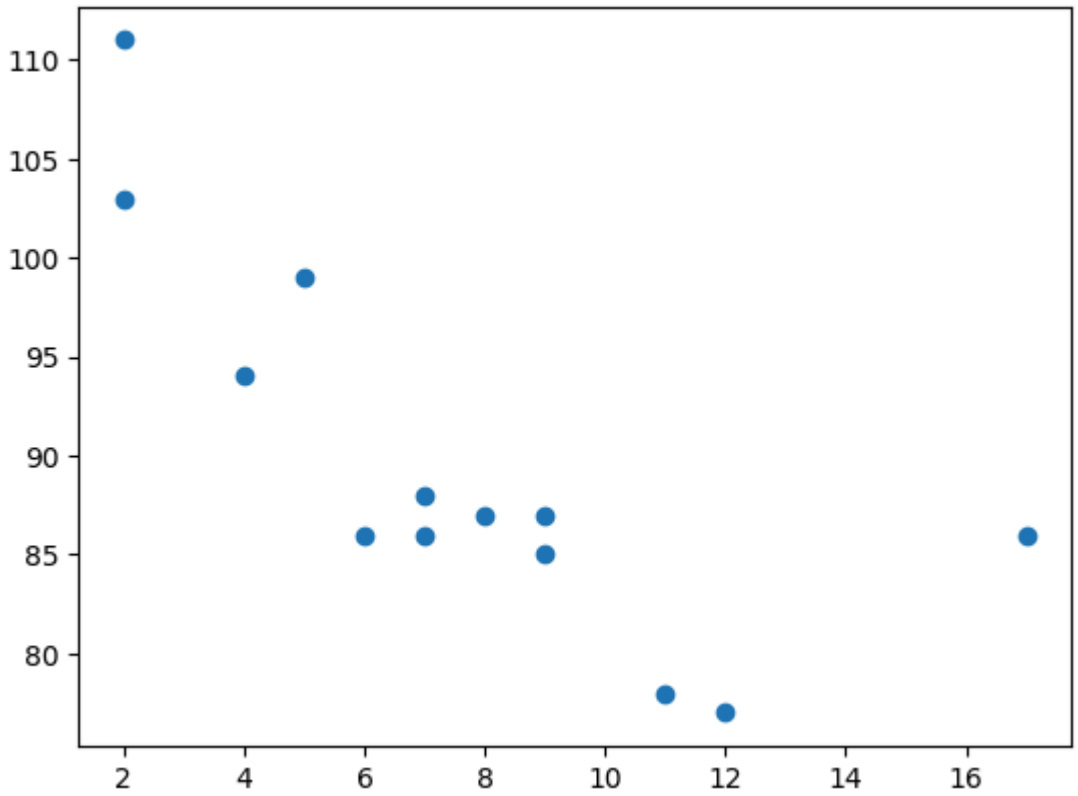
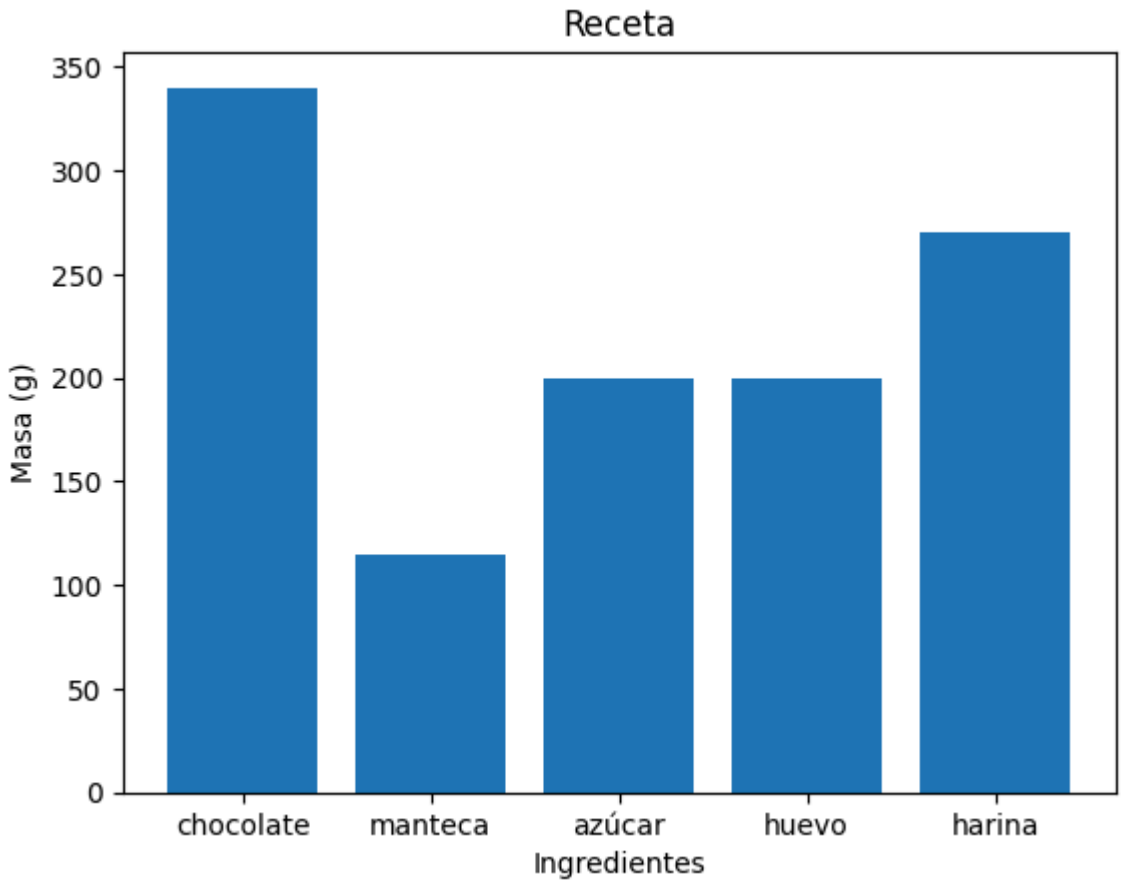


Gráfico de Barras

El gráfico de barras permite visualizar proporciones, comparando dos o más valores entre sí. Para crearlo, se utiliza la función `ax.bar()`, la cual 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']


fig, ax = plt.subplots()


ax.bar(ingredientes, peso)
```

```
ax.set_xlabel('Ingredientes')
ax.set_ylabel('Masa (g)')


ax.set_title("Receta")


plt.show()
```



Note que con la función anterior, las barras adquieren una dirección vertical: si quisieramos verlas de manera horizontal, debemos usar la funcion `ax.barh()` y cambiar los títulos de los ejes según corresponda:

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

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


ax.barh(ingredientes, peso)


ax.set_ylabel('Ingredientes')
ax.set_xlabel('Masa (g)')


ax.set_title("Receta")


plt.show()
```

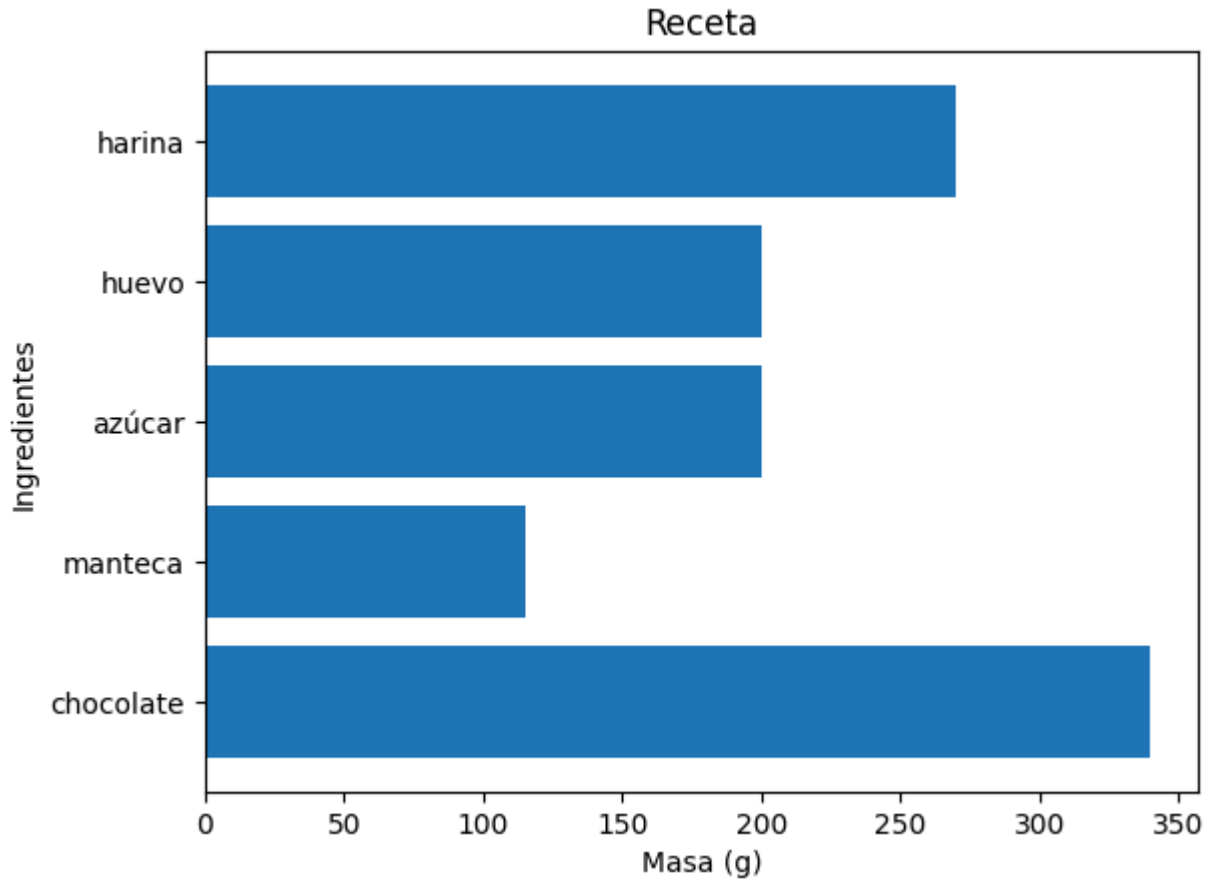
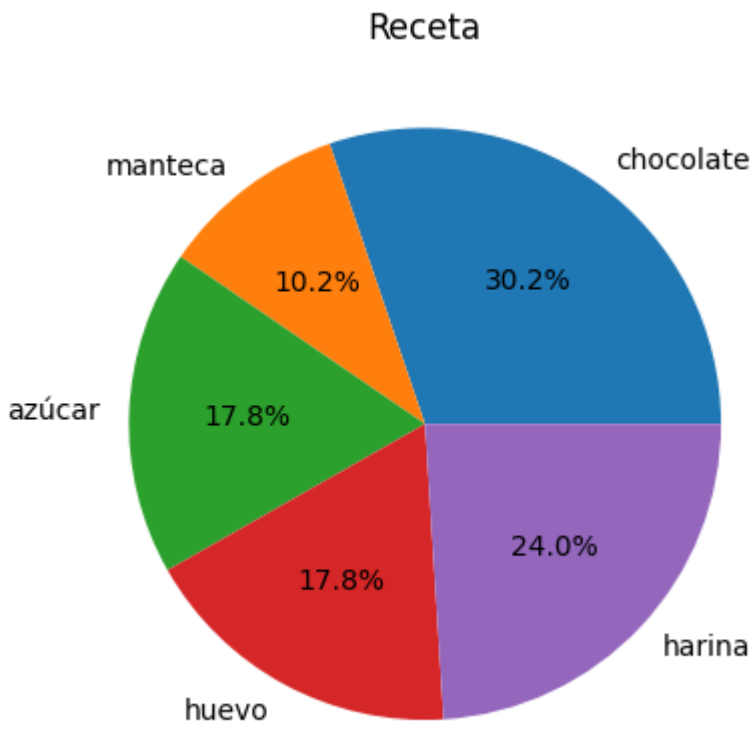


Gráfico de Torta

El gráfico de torta, como el de barras, permite visualizar y comparar proporciones pero de manera circular y como partes de un todo. Para crearlo, se utiliza la función `ax.pie()`, la cual podría solamente recibir un arreglo de números pero es útil también saber qué simboliza cada parte. Para eso se usa el parámetro `labels` que le asigna una etiqueta a cada porción y el parámetro `autopct` que indica cómo se mostrará el porcentaje: **`%1.1f%%`** le indica que el porcentaje tendrá un decimal, mientras que **`%1.2f%%`** tendrá 2 decimales.

```
peso = [340, 115, 200, 200, 270]

ingredientes = ['chocolate', 'manteca', 'azúcar', 'huevo', 'harina']



fig, ax = plt.subplots()



ax.pie(peso, labels= ingredientes, autopct='%1.1f%%')



ax.set_title("Receta")



plt.show()
```

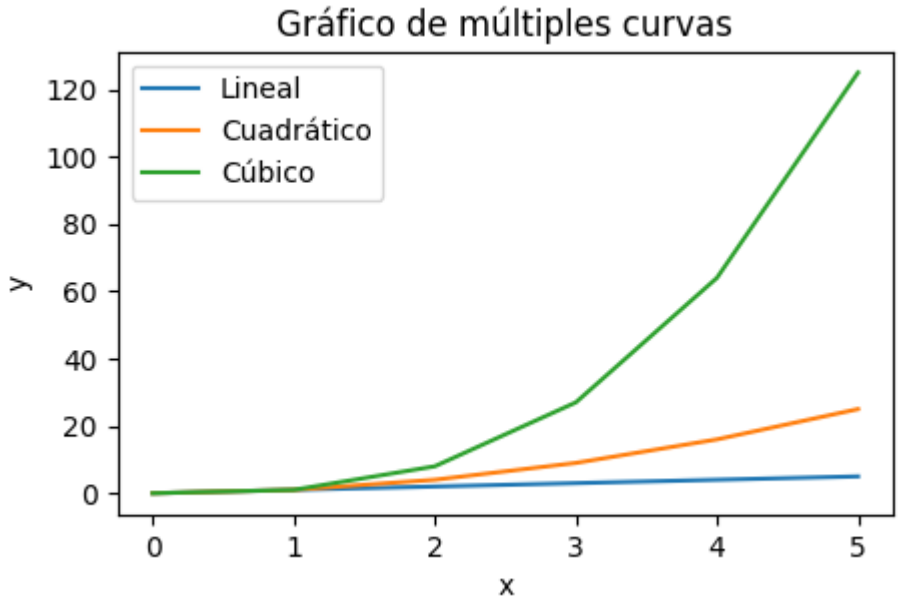


Gráficos Múltiples

En los casos anteriores, creamos siempre un sólo gráfico con una curva, en una figura. Pero...
¿Cómo podríamos graficar varias curvas en un mismo gráfico?

```
# Valores que se desean graficar

x = [0, 1, 2, 3, 4, 5]

y_linear = [0, 1, 2, 3, 4, 5]

y_quadratic = [0, 1, 4, 9, 16, 25]

y_cubic = [0, 1, 8, 27, 64, 125]


fig, ax = plt.subplots(figsize=(5, 3))


ax.plot(x, y_linear, label='Lineal')

ax.plot(x, y_quadratic, label='Cuadrático')

ax.plot(x, y_cubic, label='Cúbico')
```

```
ax.set_title("Gráfico de múltiples curvas")

ax.set_xlabel('x')

ax.set_ylabel('y')

ax.legend()


plt.show()
```

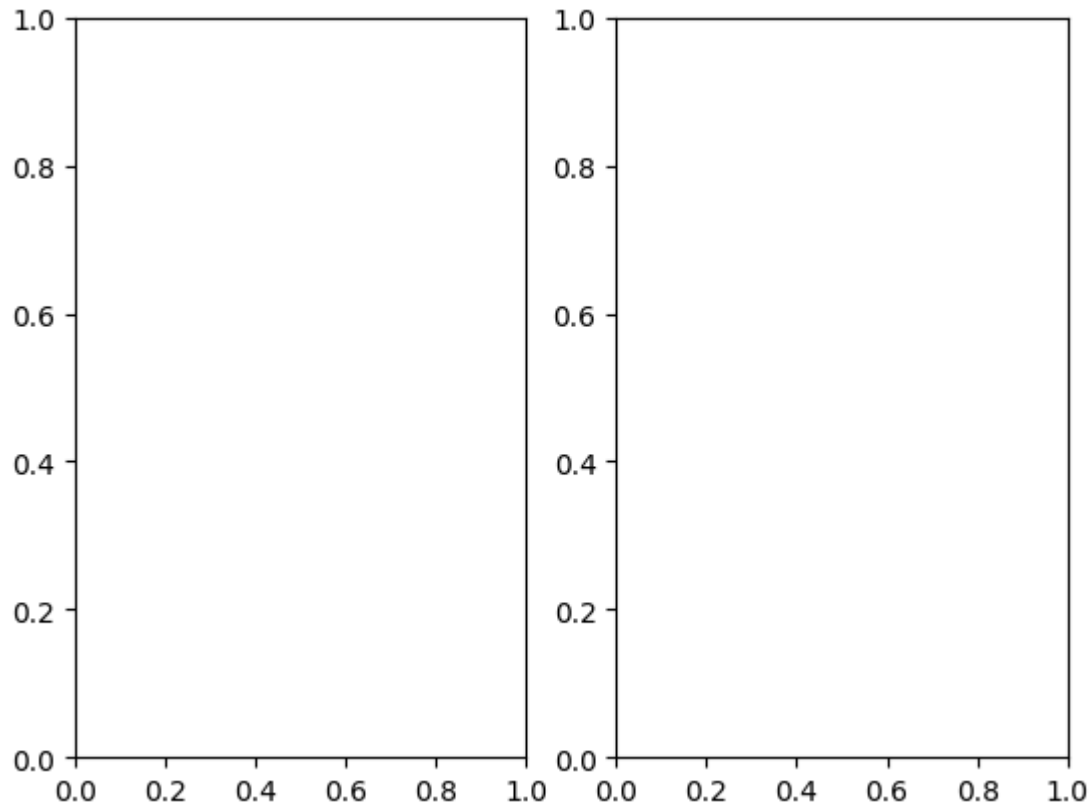


Note que estamos agregando nuevos datos al mismo axes, por lo que siempre usamos `ax.plot()` pero con distintos valores de y. Además, se estableció un tamaño de la figura con `figsize=(width, height)`.

Grilla de Gráficos

También podríamos querer ver varios axes en una misma figura. Para ello, tenemos que definir, como si se tratase de una tabla, cuántas columnas `ncols` y cuantas `nrows` de gráficos deseamos. Por ejemplo, supongamos que quiero ver dos gráficos en una misma fila:

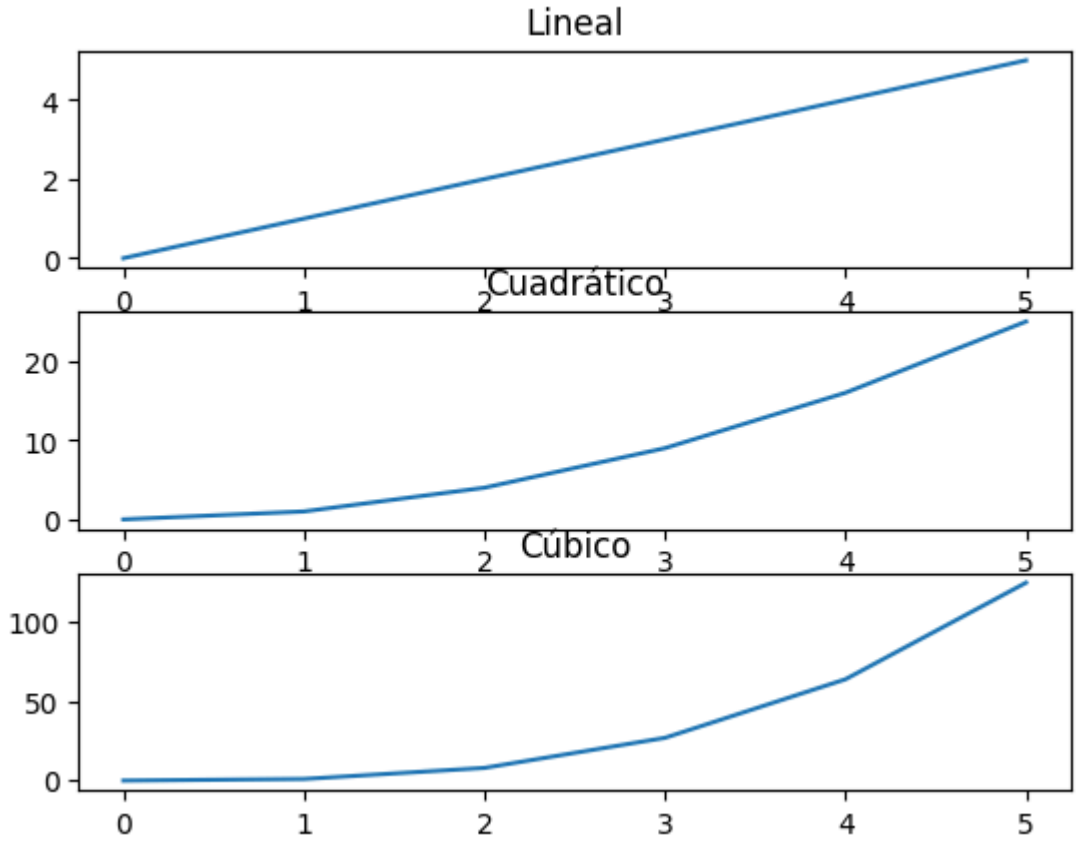
```
fig, ax = plt.subplots(nrows=1, ncols=2) # o simplemente plt.subplots(1,2)
```



De manera análoga, podemos representar las 3 curvas de la figura titulada "Gráfico de múltiples curvas" pero viendo 3 filas en una única columna de gráficos:

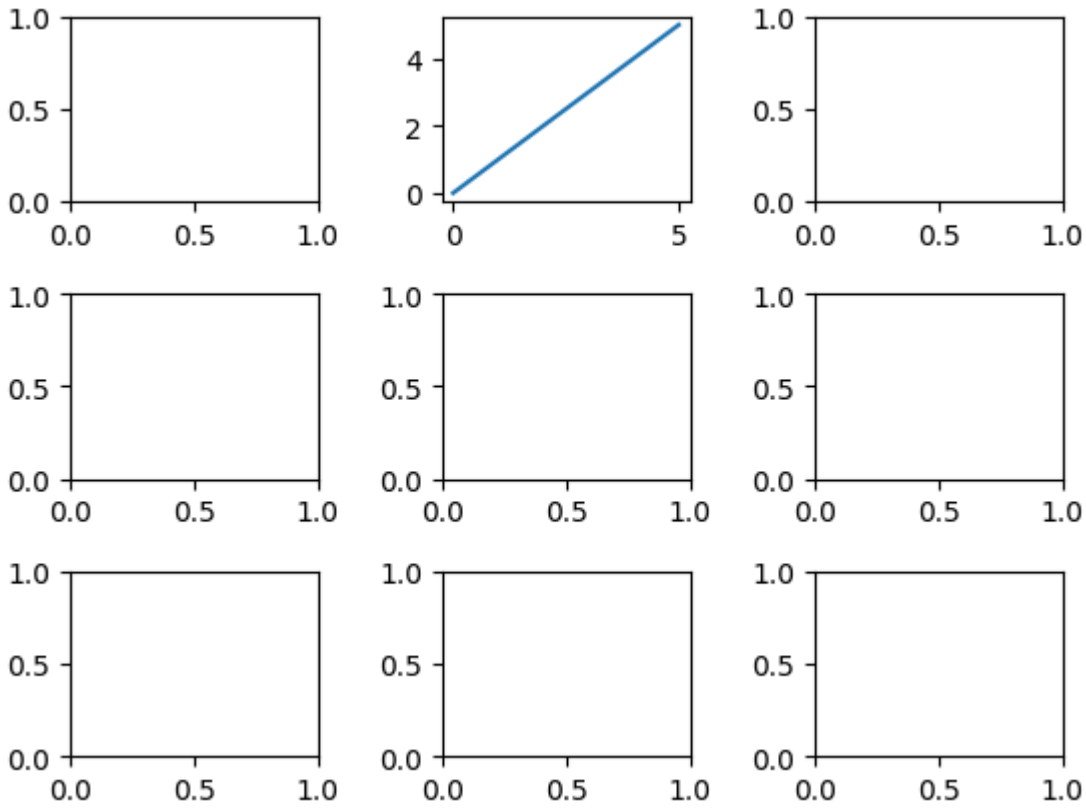
```
# Valores que se desean graficar

x = [0, 1, 2, 3, 4, 5]

x_linear = [0, 1, 2, 3, 4, 5]

x_quadratic = [0, 1, 4, 9, 16, 25]

x_cubic = [0, 1, 8, 27, 64, 125]


fig, ax = plt.subplots(nrows=3, ncols=1)
```

```
ax[0].plot(x, x_linear)

ax[0].set_title('Lineal')



ax[1].plot(x, x_quadratic)

ax[1].set_title('Cuadrático')



ax[2].plot(x, x_cubic)

ax[2].set_title('Cúbico')



plt.show()
```



Entonces, ahora en vez de graficar usando `ax.plot()` unicamente, tengo que **indicar la posición del axes con los números dentro del corchete**, ¿pero si tengo varias columnas y filas?

```
fig, ax = plt.subplots(nrows=3, ncols=3)

fig.subplots_adjust(wspace=0.5, hspace=0.5) # Con esto indicamos el espacio libre entre los subplots



ax[0, 1].plot(x, x_linear)

plt.show()
```



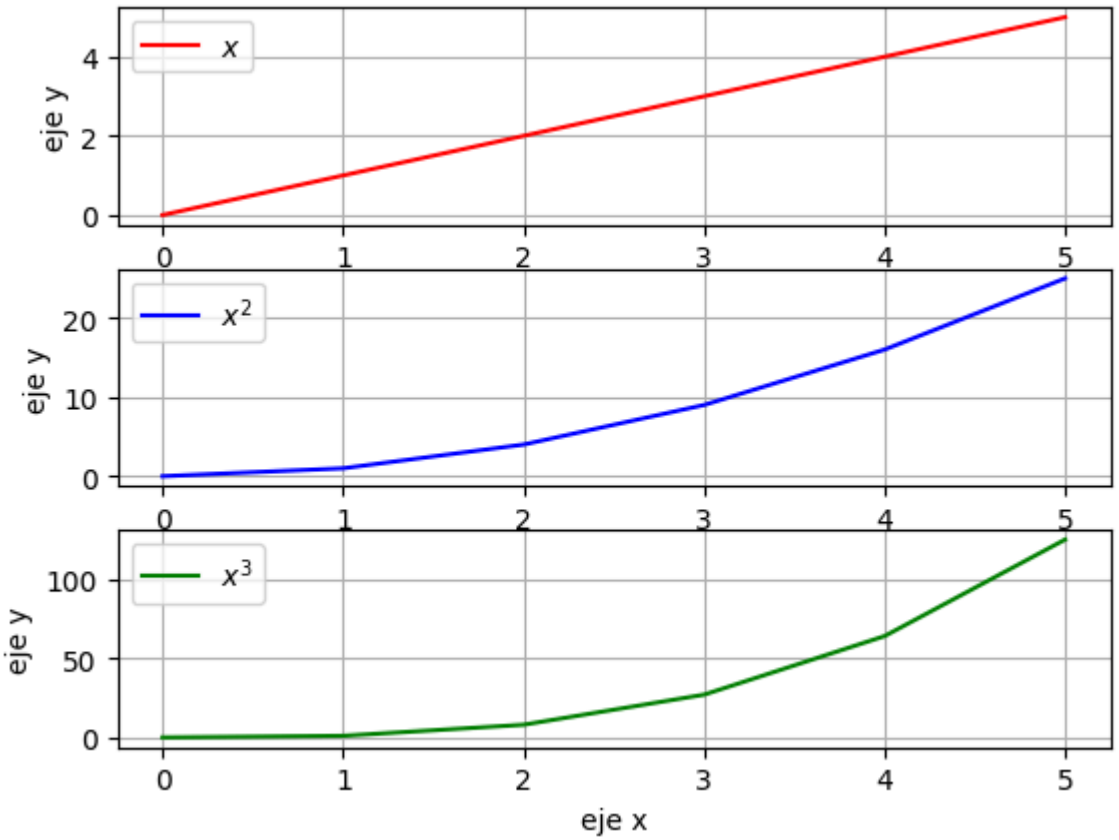
Note que dentro del corchete, primero se indica la fila y luego la columna: `ax[fila, columna]`.

Funciones de Gráficas

En términos generales, si nos encontramos en la situacion de copiar y pegar las mismas líneas de código para realizar gráficos similares, tendríamos que pensar en crear una función que simplifique esta tarea.

```
x = [0, 1, 2, 3, 4, 5]
```

```
x_linear = [0, 1, 2, 3, 4, 5]

x_quadratic = [0, 1, 4, 9, 16, 25]

x_cubic = [0, 1, 8, 27, 64, 125]



fig, ax=plt.subplots(3)

ax[0].plot(x,x_linear,label="$x$",color="r")

ax[0].set_xlabel("eje x")

ax[0].set_ylabel("eje y")

ax[0].legend()

ax[0].grid()




ax[1].plot(x,x_quadratic,label="$x^2$",color="b")

ax[1].set_xlabel("eje x")

ax[1].set_ylabel("eje y")

ax[1].legend()

ax[1].grid()




ax[2].plot(x,x_cubic,label="$x^3$",color="g")

ax[2].set_xlabel("eje x")

ax[2].set_ylabel("eje y")

ax[2].legend()

ax[2].grid()

plt.show()
```



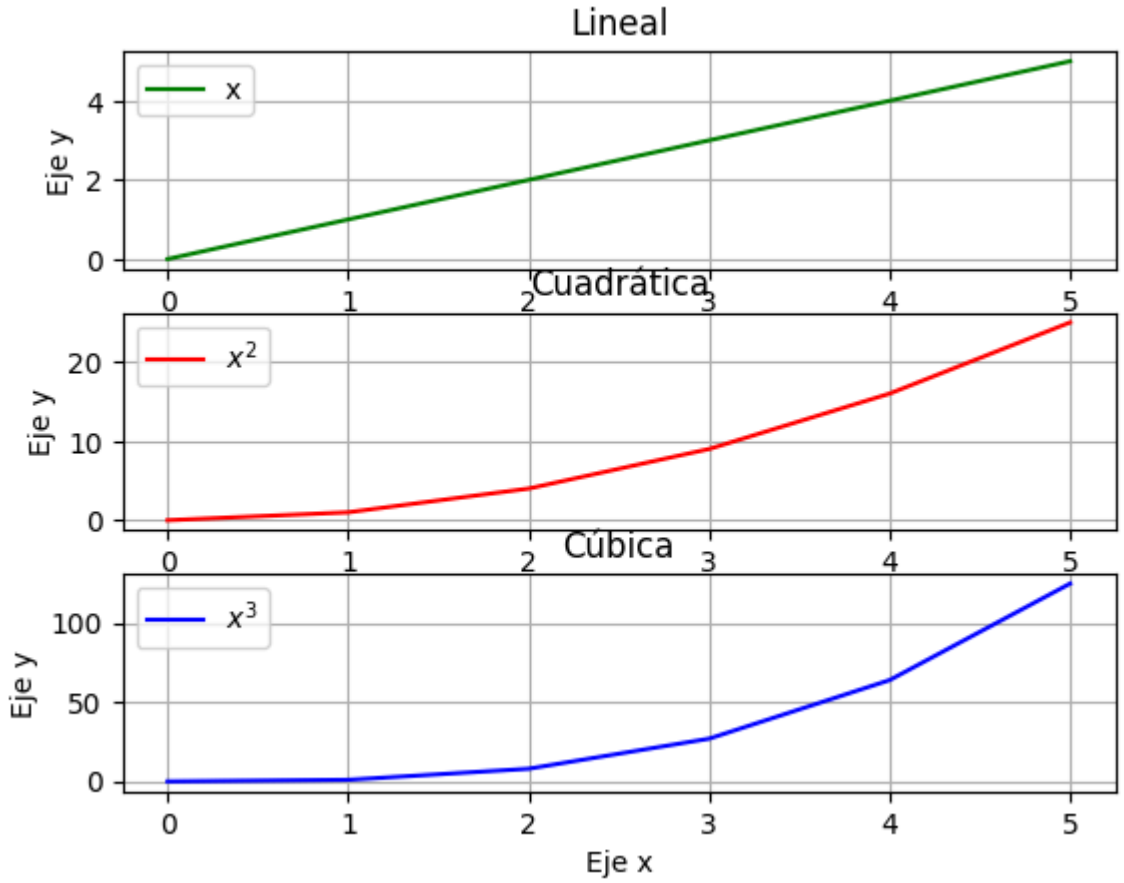
Para evitar lo anterior, definimos una función a la que le debemos entregar los valores a graficar:

```
def create_easy_graph(x, y, label, ax, xlabel, ylabel, title, color):

"""Crea un gráfico a partir de vectores con valores de los ejes x e y.

Recibe además:

- El texto para el label

- El subplot a donde graficar

- Un label para el eje x

- Un label para el eje y

- Un título para el gráfico

- Un color

El color y el eje pueden ser None. En ese caso toman valores por default"""


if color == None:
```

```
color = "blue"


# Si sólo haremos un gráfico, no necesito indicarle la posición
if ax == None:
fig, ax = plt.subplots()


# Definimos el gráfico
ax.plot(x, y, label=label, color=color)
ax.set_xlabel(xlabel)
ax.set_ylabel(ylabel)
ax.set_title(title)


return ax
```

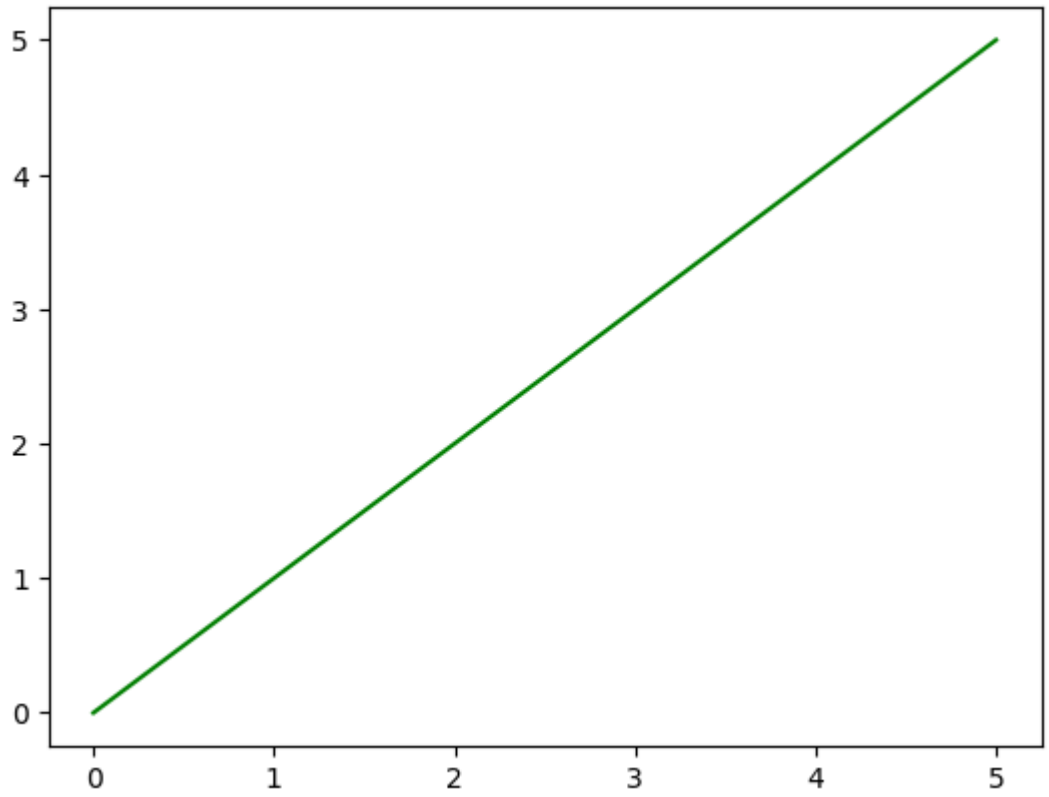
Y por otro lado:

```
fig , ax = plt.subplots(3)


# En vez de copiar y pegar el código, llamo a la función easy_graph():
create_easy_graph(x, x_linear, "x", ax[0], "Eje x", "Eje y", "Lineal",
color="green")
create_easy_graph(x, x_quadratic, "$x^2$", ax[1], "Eje x", "Eje y",
"Cuadrática", color="red")
create_easy_graph(x, x_cubic, "$x^3$", ax[2], "Eje x", "Eje y", "Cúbica", None)


# Hacemos un for para agregar la cuadrícula y las referencias en cada axes:
for axes in fig.axes[:]:
axes.grid()
axes.legend()
```



Como comentamos dentro de la función, también podemos usar `easy_graph()` para un único gráfico:

```
create_easy_graph(x, x_linear, "x", None, "", "", "", "green")
```



Gráficos Utilizando DataFrames

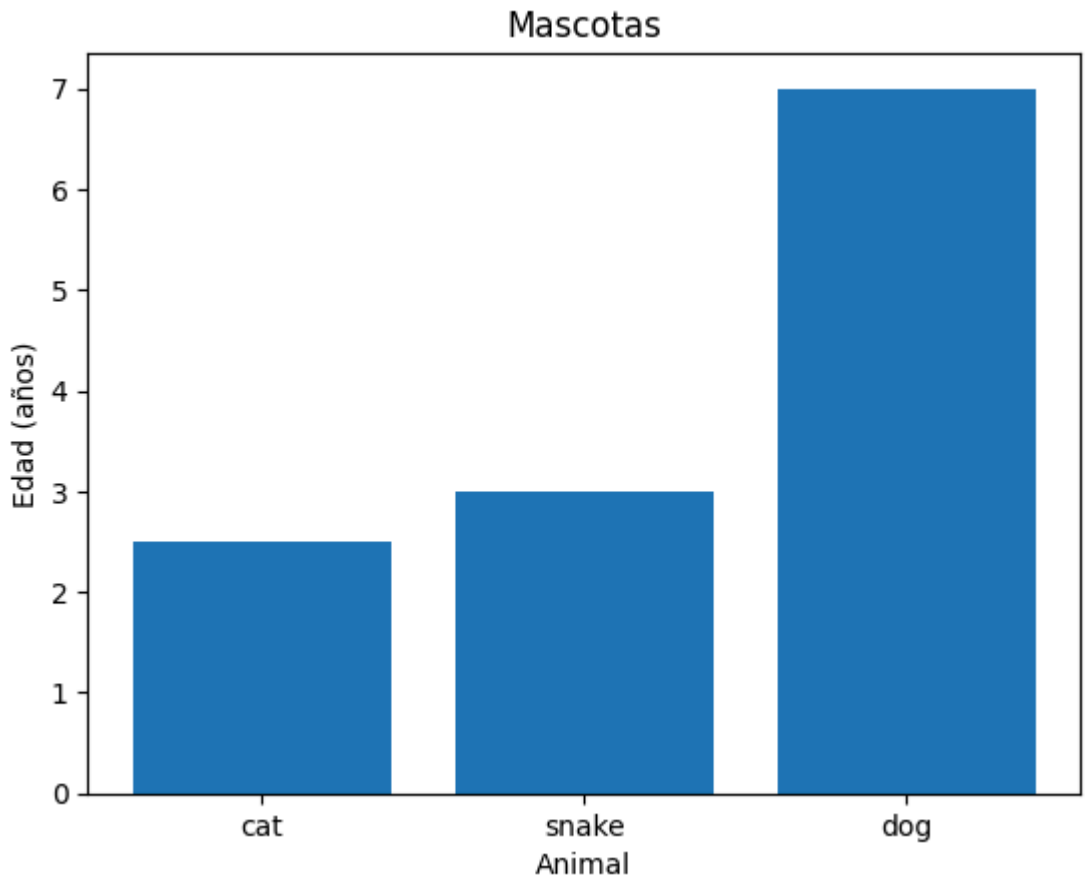
No es necesario que todos nuestros datos provengan de vectores de valores. También podríamos utilizar los datos contenidos en un DataFrame como los que vimos en la sesión pasada.

```
import pandas as pd
```

```
data = {'animal': ['cat','snake', 'dog'],
'age': [2.5, 3, 7],
'visits': [1, 3, 2],
'priority': ['yes', 'yes', 'no']}

df = pd.DataFrame(data)
df
```

	animal	age	visits	priority
0	cat	2.5	1	yes
1	snake	3.0	3	yes
2	dog	7.0	2	no

```
# Determino las columnas del DataFrame que queremos graficar
x_values = df['animal']
y_values = df['age']

fig, ax = plt.subplots()

ax.bar(x_values, y_values)

ax.set_xlabel('Animal')
ax.set_ylabel('Edad (años)')

ax.set_title("Mascotas")

plt.show()
```



De esta forma, podríamos trabajar con dataframes de la sesión anterior y usarlos para graficar.