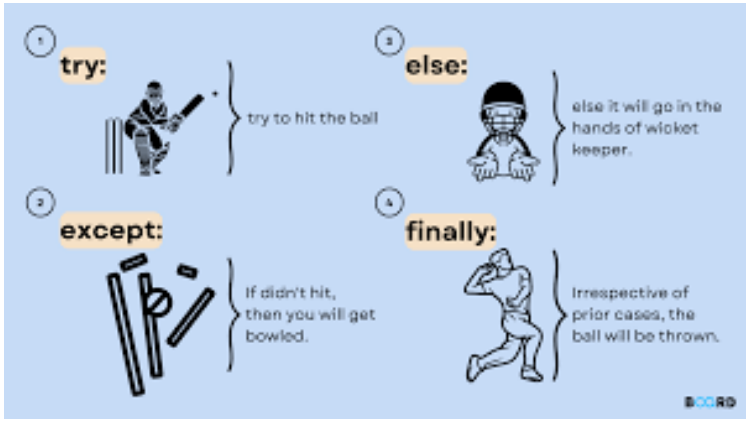
UNIVERSIDAD NACIONAL DE ASUNCIÓN
FACULTAD DE INGENIERÍA

Cursos Básicos

Manejo de Archivos y Visualización de Datos



matplotlib

seaborn

plotly

¿Qué veremos hoy?

- Manejo Básico de archivos(Repaso)
  - Lectura, Escritura, CSV, txt
- Manejo de Excepciones

  ¿Qué son las excepciones en Python?

  Tipos de excepciones comunes.

  Bloques try-except para manejar excepciones
- Introducción a Matplotlib

  Creación de gráficos básicos, Guardar gráficos en archivos.

  Creación de histogramas. Gráficos de líneas para series temporales.

  Gráficos de dispersión para relaciones entre variables.

  Visualización de imágenes con Matplotlib.

Manejo de arquivos

Lectura de Archivos

- **open():** Utilizamos la función open() para abrir un archivo en modo de lectura.
- **Métodos de Lectura:**
  - read(): Lee todo el contenido del archivo.
  - readline(): Lee una línea del archivo.
  - readlines(): Lee todas las líneas del archivo y las devuelve como una lista.

```
# Intentamos abrir el archivo en modo lectura
with open("datos.txt", "r") as archivo:
    # Leemos el contenido del archivo
    contenido = archivo.read()
    # Mostramos el contenido por pantalla
    print("Contenido del archivo:")
    print(contenido)
```

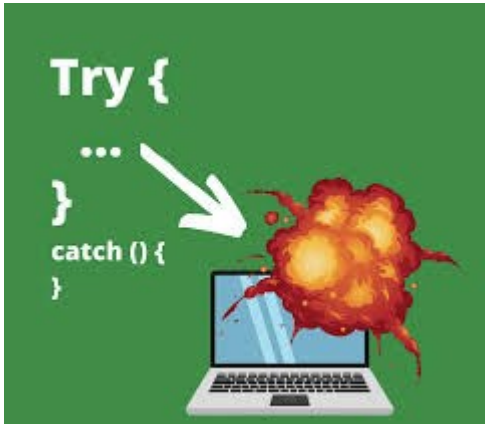
```
try:
    # Intentamos abrir el archivo en modo lectura
    with open("datos.txt", "r") as archivo:
        # Leemos el contenido del archivo
        contenido = archivo.read()
        # Mostramos el contenido por pantalla
        print("Contenido del archivo:")
        print(contenido)
except FileNotFoundError:
    # Si el archivo no existe, mostramos un mensaje de error
    print("El archivo no existe.")
```

Qué son las excepciones

Las excepciones son eventos que interrumpen el flujo normal de ejecución de un programa.

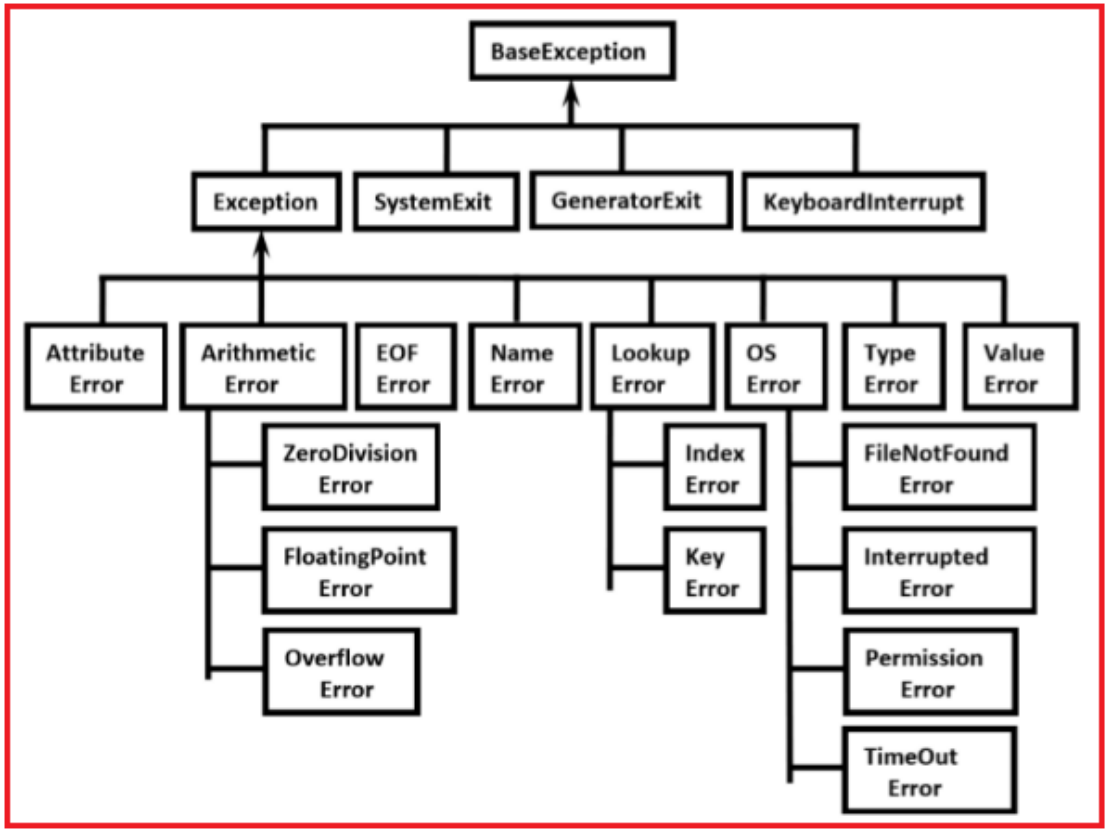
Representan situaciones anómalas o errores que pueden ocurrir durante la ejecución.

Permiten manejar de manera controlada y elegante los errores que pueden surgir en un programa.



- **SyntaxError**: Errores de sintaxis en el código Python.
- **IndentationError**: Problemas de indentación incorrecta.
- **NameError**: Uso de variables o funciones no definidas.
- **TypeError**: Operaciones con tipos de datos incompatibles.
- **ValueError**: Argumentos con valores incorrectos para una función.
- **FileNotFoundError**: Archivos no encontrados al intentar abrirlos.
- **IOError**: Errores de entrada/salida durante la lectura o escritura de archivos.

Tipos de Excepciones Comunes



Bloques try-except para Manejar Excepciones

Se utilizan para manejar excepciones de manera controlada.

El bloque try permite ejecutar código que puede generar excepciones.

El bloque except captura y maneja las excepciones que se producen.

Permite continuar la ejecución del programa incluso si se produce un error.

Uso de las Cláusulas finally y else

finally: Se ejecuta siempre, independientemente de si se produce una excepción o no.

Se utiliza para realizar limpieza de recursos.

else: Opcional, se ejecuta si no se produce ninguna excepción en el bloque try.

Mejores Prácticas para el Manejo de Excepciones

Capturar excepciones de manera específica en lugar de general.

No capturar excepciones demasiado amplias como Exception.

Mantener los bloques try-except lo más cortos posible.

Utilizar la cláusula finally para la limpieza de recursos.

Registrar o manejar las excepciones de manera adecuada para el usuario final.

¿Qué es Matplotlib?

Matplotlib fue desarrollado principalmente por John D. Hunter.

Comenzó el proyecto en 2002 como un esfuerzo para crear una biblioteca de visualización de datos en Python que fuera similar a las capacidades de trazado en MATLAB.

Latest stable release
3.8.3: docs |

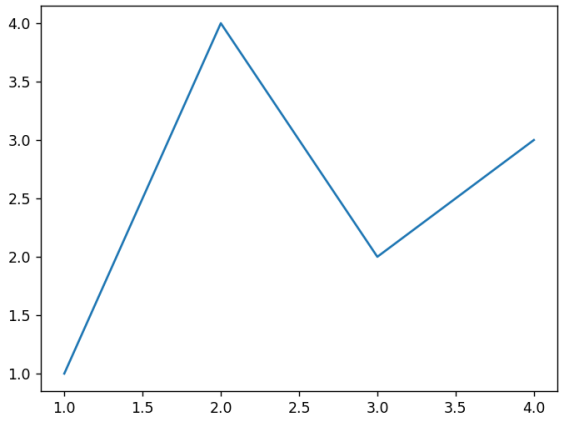
https://matplotlib.org/stable/users/index.html

Matplotlib es una biblioteca de visualización de datos en Python.

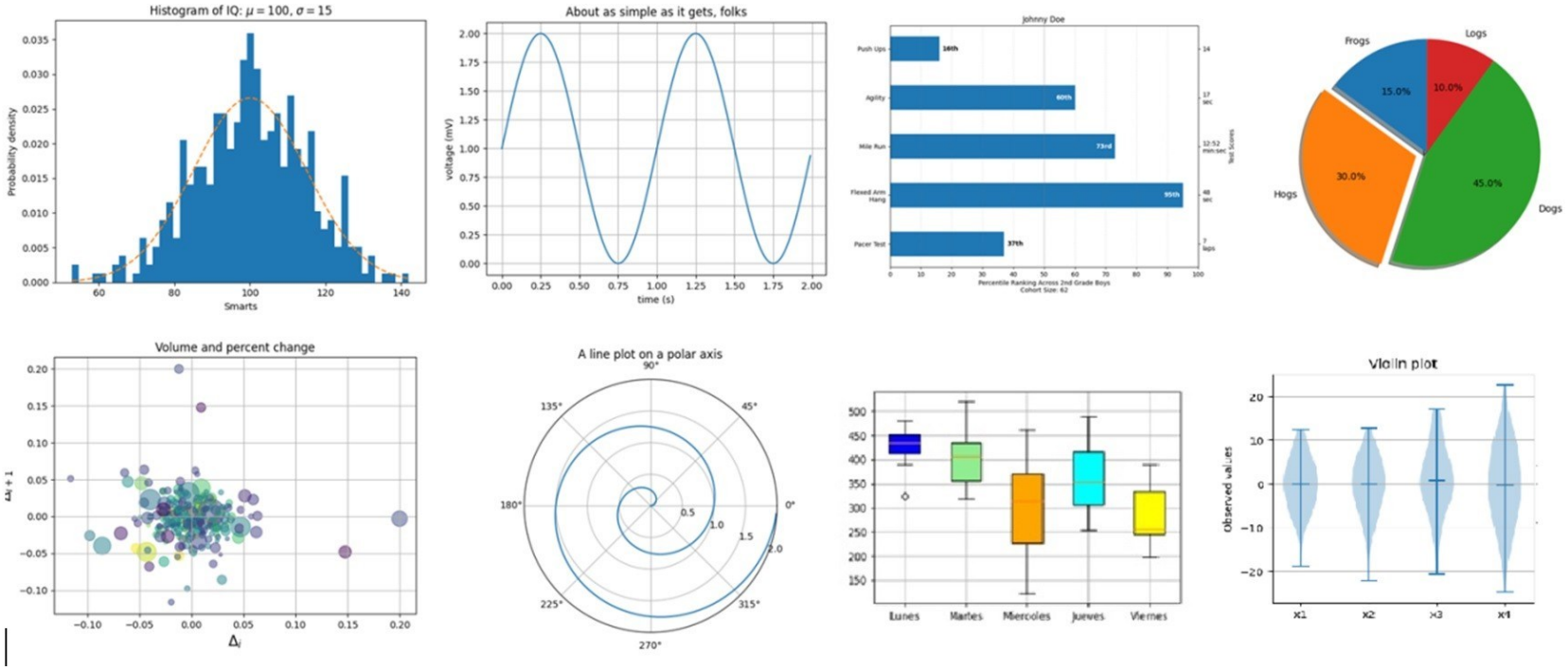
Permite crear gráficos de alta calidad, como gráficos de líneas, histogramas, gráficos de dispersión y más.

Es altamente personalizable y compatible con varios formatos de archivo para guardar gráficos.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])
plt.show()
```



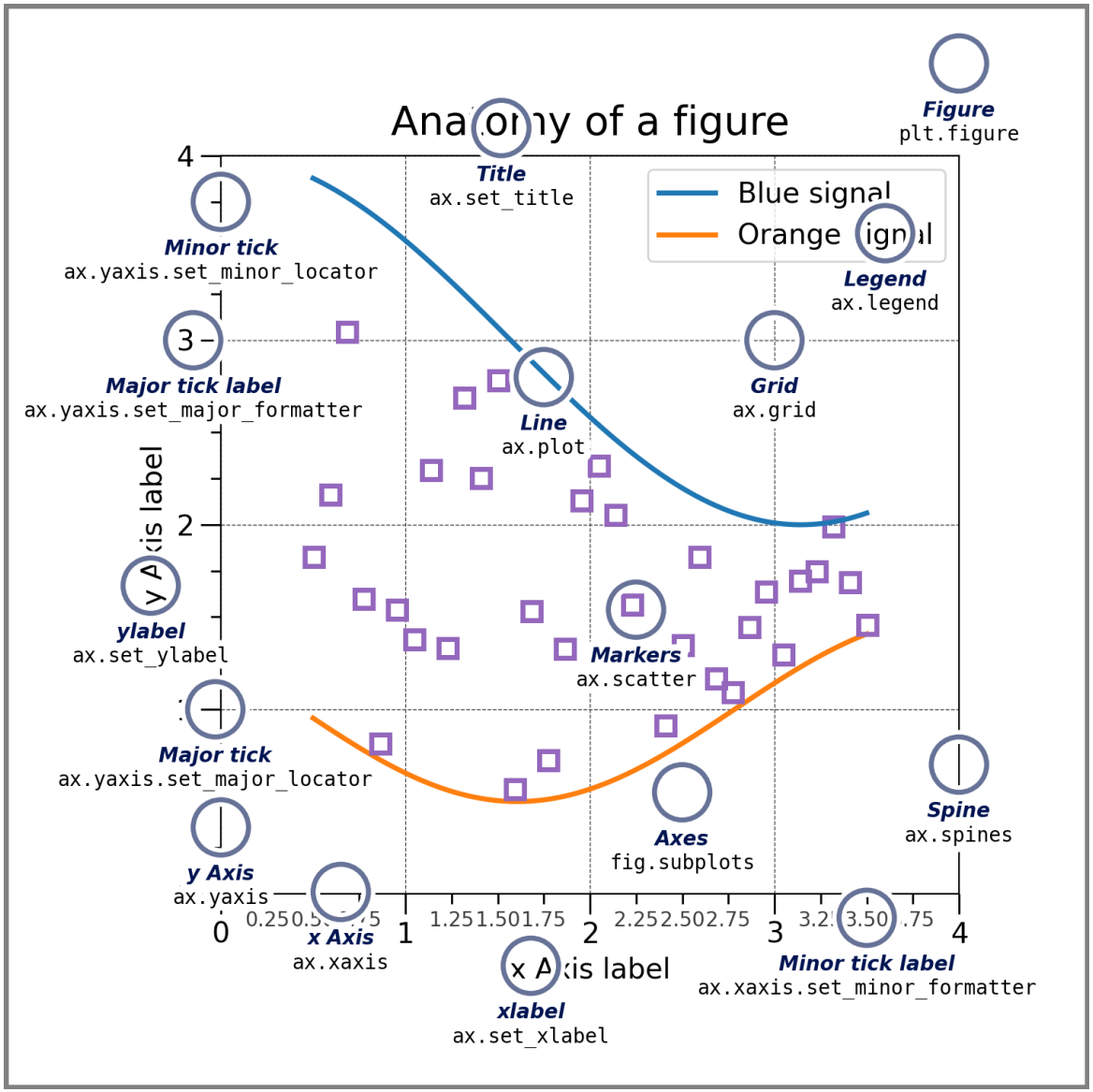
Creación de Gráficos Básicos



Latest stable release
3.8.3: docs |

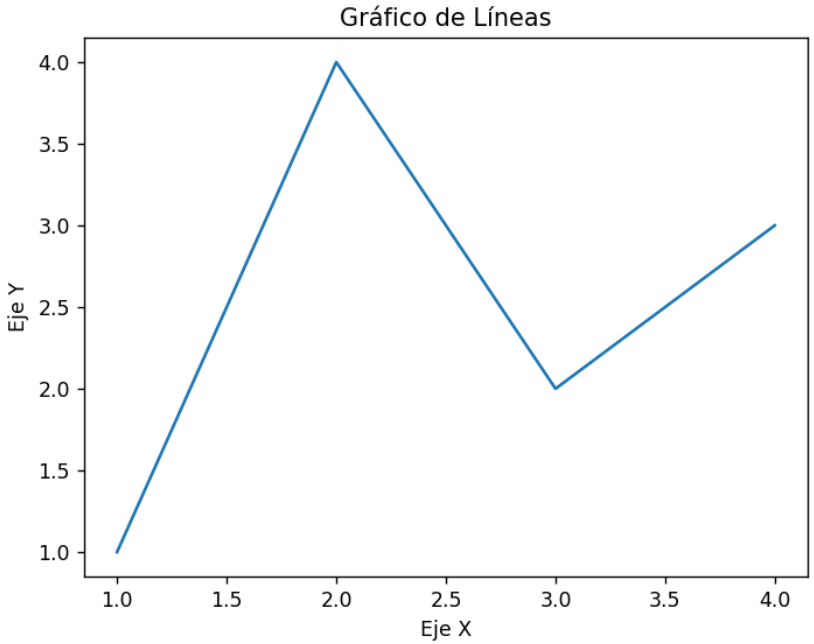
https://matplotlib.org/stable/users/index.html

Personalizable



Títulos y ejes

```python
import matplotlib.pyplot as plt
# Crear figura y ejes
fig, ax = plt.subplots()
# Graficar datos
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])
# Agregar etiquetas y título
ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_title('Gráfico de Líneas')
# Mostrar el gráfico
plt.show()
```



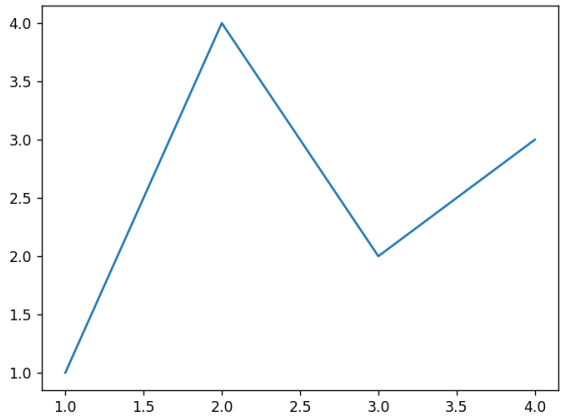
Permite guardar las imágenes

Matplotlib permite guardar gráficos en varios formatos de archivo, como PNG, PDF, SVG, etc.

Utilizando la función savefig, se puede guardar el gráfico generado en el directorio especificado.

Esto es útil para compartir y utilizar los gráficos en informes, presentaciones y publicaciones.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])
plt.show()
plt.savefig("grafico.png")
```



Parámetros:
'nombre_archivo.formato': Especifica el nombre y el formato del archivo en el que se guardará el gráfico.
dpi: (opcional) Resolución en puntos por pulgada (dots per inch) del archivo de imagen. Por defecto es 100.
bbox_inches: (opcional) Ajuste del tamaño de la imagen para que quepan todas las partes de la figura. 'tight' ajusta el tamaño de la figura al contenido, eliminando los márgenes en blanco alrededor de la figura.
Resolución (dpi):
Controla la calidad de la imagen guardada. Un valor más alto de dpi produce una imagen con mayor resolución, pero también un archivo más grande.
Uso adecuado:
Se recomienda llamar a savefig después de plt.show(), de lo contrario, puede que no guarde la figura correctamente.

Histogramas

```
import numpy as np
import matplotlib.pyplot as plt
# Generar 1000 números aleatorios distribuidos uniformemente
entre 0 y 100
numeros_aleatorios = np.random.uniform(0, 100, 1000)
# Crear el histograma
plt.hist(numeros_aleatorios, bins=10, color='skyblue',
edgecolor='black')
```

Datos de entrada:

x: Los datos que se utilizarán para construir el histograma.

Parámetros de histograma:

bins: Controla el número de contenedores (bins) utilizados en el histograma. Puede ser un entero, una secuencia o una cadena.

range: Especifica el rango de valores que se incluirán en el histograma. Por ejemplo, range=(0, 10) limitará el histograma a valores entre 0 y 10.

density: Si es True, el histograma mostrará una densidad de probabilidad normalizada en lugar de recuentos de muestras.

Parámetros de visualización:

color: Especifica el color de las barras del histograma.

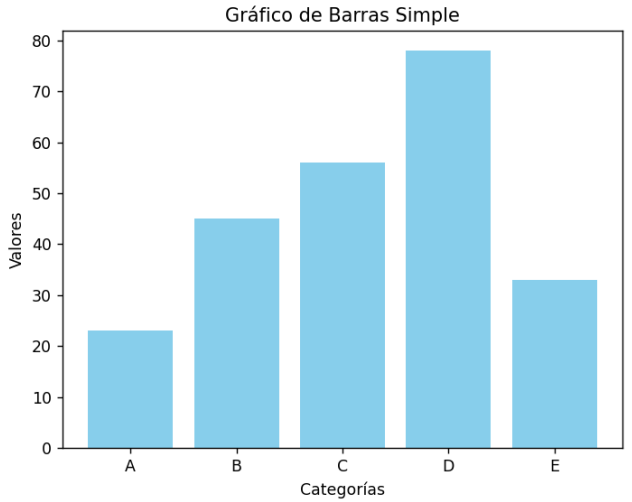
edgecolor: Especifica el color del borde de las barras del histograma.

alpha: Controla la transparencia de las barras del histograma.

histtype: Tipo de histograma. Puede ser 'bar', 'barstacked', 'step', 'stepfilled'.

Graficos de barras

```
import matplotlib.pyplot as plt
categorias = ['A', 'B', 'C', 'D', 'E']
valores = [23, 45, 56, 78, 33]
plt.bar(categorias, valores,color='skyblue')
plt.xlabel('Categorías')
plt.ylabel('Valores')
plt.title('Gráfico de Barras Simple')
plt.show()
```

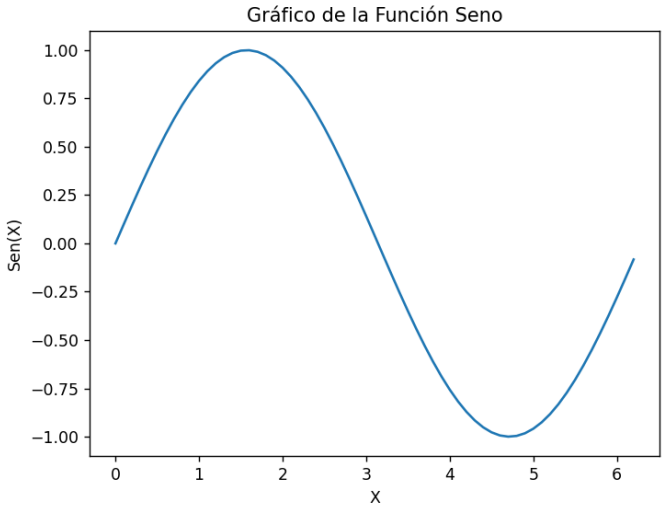


Parámetros más importantes de la función bar:

1.x: La posición en el eje x de las barras.
2.height: La altura de las barras.
3.width: El ancho de las barras. Por defecto es 0.8.
4.bottom: La coordenada en el eje y donde comienzan las barras. Por defecto es 0.
5.align: La alineación de las barras con respecto a la posición en el eje x. Puede ser 'center' (por defecto), 'edge', o 'align'.
6.color: El color de las barras.
7.edgecolor: El color del borde de las barras.
8.linewidth: El ancho del borde de las barras.
9.tick_label: Las etiquetas para las marcas del eje x.
10.orientation: La orientación de las barras. Puede ser 'vertical' (por defecto) o 'horizontal'.

Graficos de lineas

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 2*np.pi, 0.1)
y = np.sin(x)
plt.plot(x, y)
plt.xlabel('X')
plt.ylabel('Sen(X)')
plt.title('Gráfico de la Función Seno')
plt.show()
```



Parámetros de formato:

color: Especifica el color de la línea. Puede ser una cadena de color o un código de color hexadecimal.

linestyle: Especifica el estilo de la línea. Puede ser '-' para sólida, '--' para discontinua, ':' para punteada, entre otros.

linewidth: Especifica el ancho de la línea en puntos.

marker: Especifica el tipo de marcador para los puntos de datos. Puede ser 'o' para círculos, 's' para cuadrados, 'x' para cruces, entre otros.

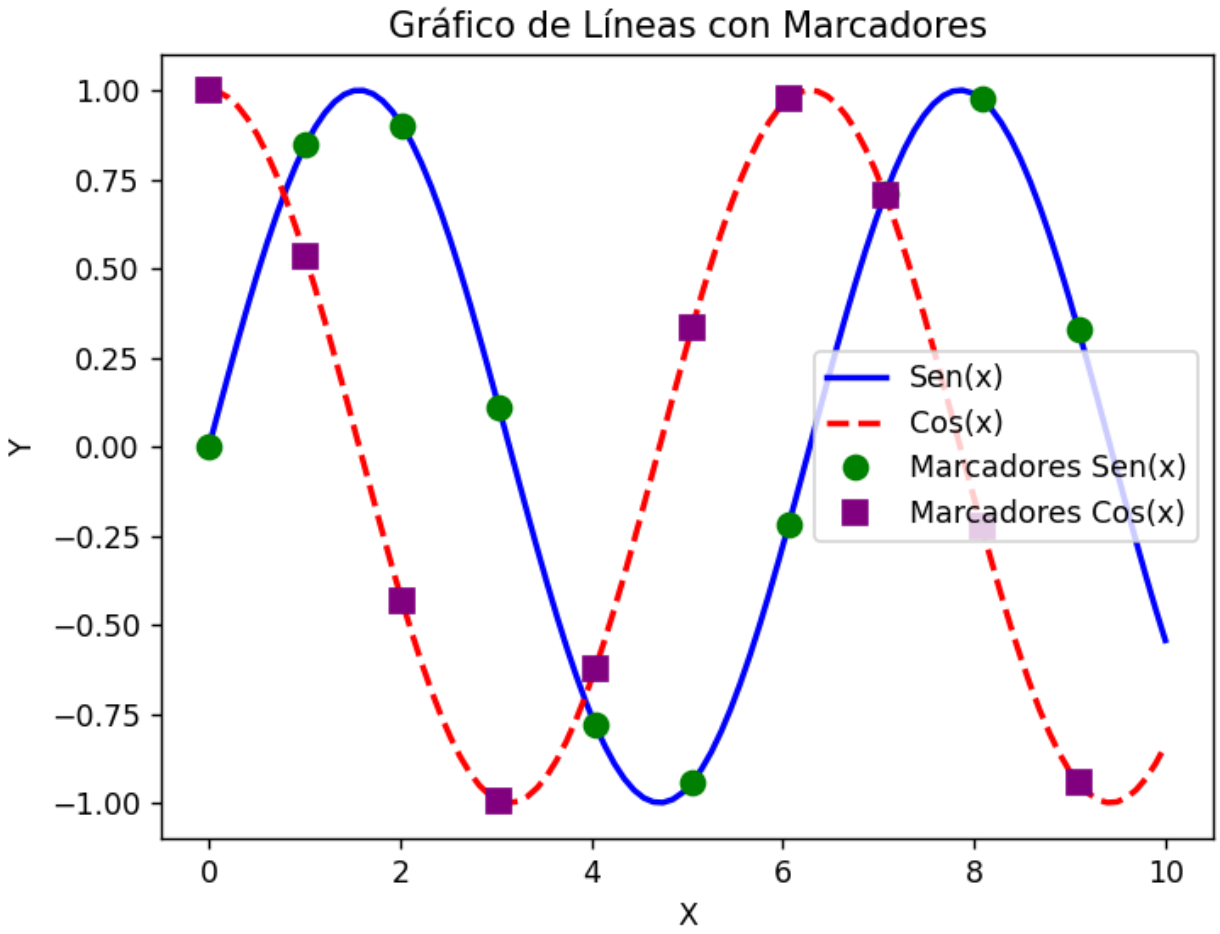
Otros parámetros importantes:

label: Etiqueta para la línea, que se utilizará en la leyenda si se ha creado.

alpha: Controla la transparencia de la línea.

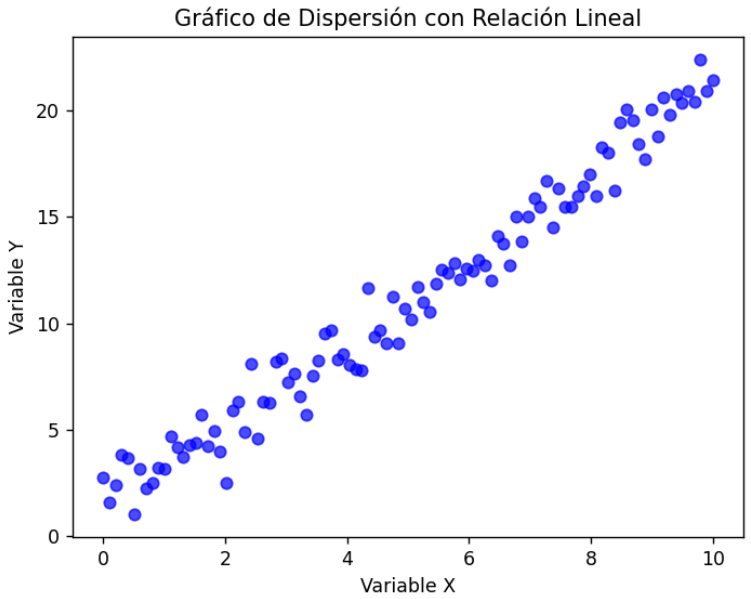
zorder: Controla el orden de superposición de la línea respecto a otros elementos gráficos.

Graficos de lineas



Graficos de dispersión

```
import numpy as np
import matplotlib.pyplot as plt
# Generar datos para una relación lineal y agregar
ruido aleatorio
np.random.seed(0)
x = np.linspace(0, 10, 100)
y = 2 * x + 1 + np.random.normal(0, 1, 100)  #
# Crear el gráfico de dispersión
plt.scatter(x, y, color='blue', alpha=0.7)
# Agregar etiquetas y título
plt.xlabel('Variable X')
plt.ylabel('Variable Y')
plt.title('Gráfico de Dispersión con Relación
Lineal')
```

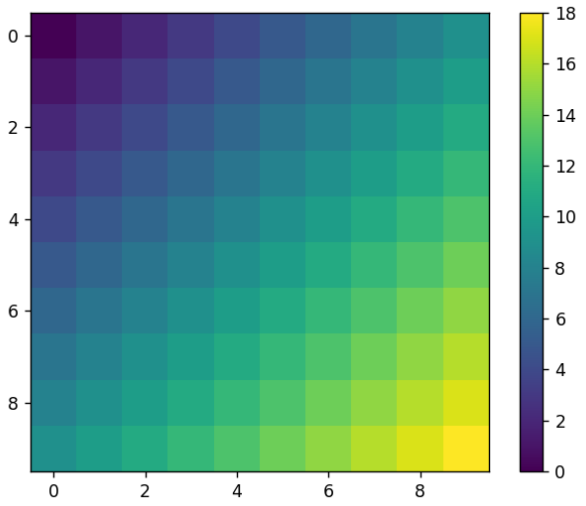


Parámetros

1.x: La posición en el eje x de los puntos.
2.y: La posición en el eje y de los puntos.
3.s: El tamaño de los puntos.
4.c: El color o secuencia de colores de los puntos.
5.marker: El tipo de marcador utilizado para representar los puntos.
6.cmap: Un mapa de colores que se utilizará si c es un array de valores numéricos.
7.alpha: La transparencia de los puntos.
8.linewidths: El ancho de los bordes de los puntos.
9.edgecolors: El color de los bordes de los puntos.
10.label: Etiqueta asociada con los puntos que se utilizará en la leyenda si se ha creado.
11.zorder: El orden de superposición de los puntos respecto a otros elementos gráficos.

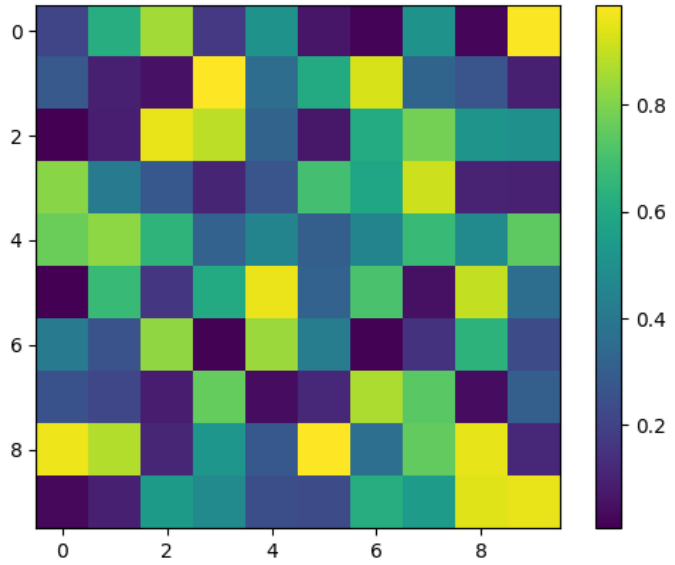
Imágenes

```
import numpy as np
import matplotlib.pyplot as plt
# Generar una matriz aleatoria de 10x10
#matriz = np.random.rand(10, 10)
matriz = np.array([[i + j for i in range(10)] for
j in range(10)])
# Visualizar la matriz utilizando imshow
plt.imshow(matriz, cmap='viridis',
interpolation='nearest')
# Agregar barra de color
plt.colorbar()
# Mostrar la imagen
plt.show()
```



Imágene

*Con valores generados aleatoriamente

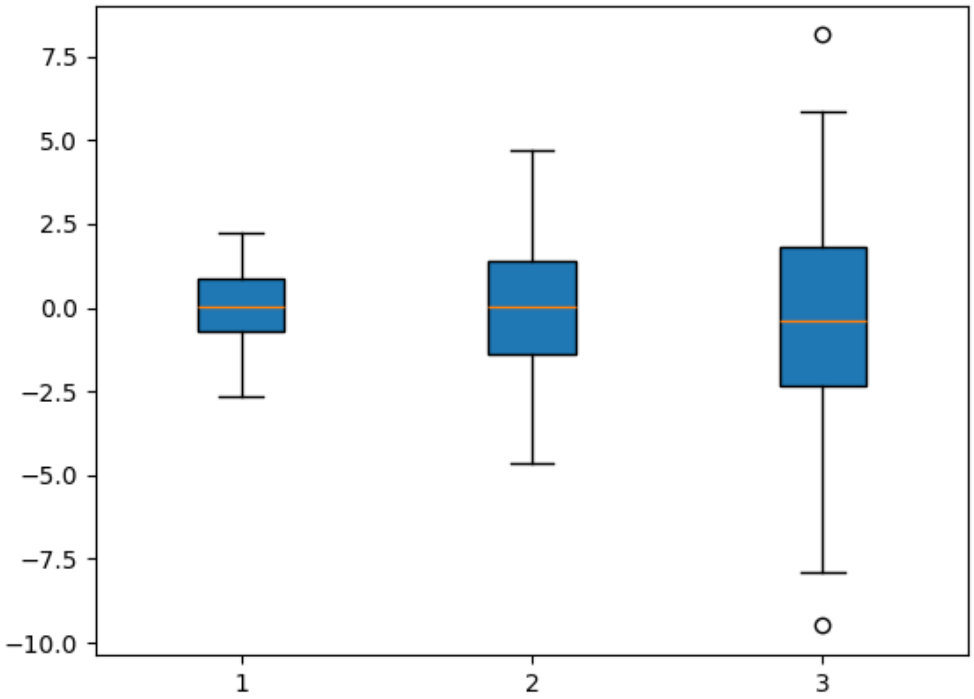


1.X: La matriz de datos que se va a visualizar.
2.cmap: El mapa de colores que se utilizará para la visualización.
3.aspect: La relación de aspecto de los ejes. Puede ser 'auto', 'equal', o un número que represente la relación de aspecto deseada.
4.interpolation: El método de interpolación utilizado para mapear los valores de la matriz a los píxeles de la imagen. Puede ser 'nearest', 'bilinear', 'bicubic', entre otros.
5.alpha: La transparencia de la imagen. Un valor de 0 significa completamente transparente y un valor de 1 significa completamente opaco.
6.origin: La ubicación del origen de la imagen. Puede ser 'upper' (por defecto) o 'lower'.
7.extent: El rango de valores para los ejes x e y de la imagen.
8.vmin y **vmax**: Los valores mínimo y máximo que se utilizarán para normalizar los valores de la matriz a valores de color.
9.norm: La normalización que se aplicará a los valores de la matriz antes de mapearlos a valores de color.
10.filternorm y **filterrad**: Parámetros para controlar la suavidad de la interpolación.
11.origin: La ubicación del origen de la imagen. Puede ser 'upper' o 'lower'.
12.extent: El rango de valores para los ejes x e y de la imagen.
13.filternorm y **filterrad**: Parámetros para controlar la suavidad de la interpolación.

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html

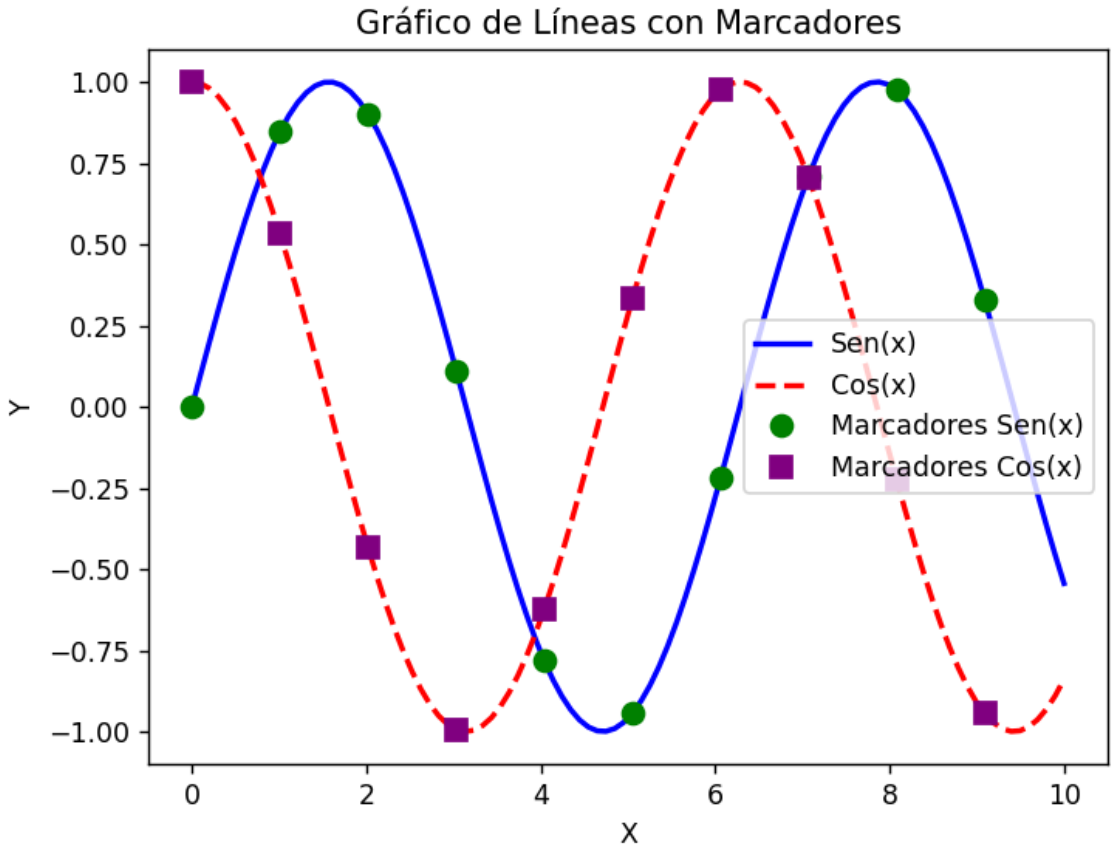
Diagrama de caja o *box plot*

```
#Generamos listas con distribución normal (con distintas
desviaciones estándar)
data = [np.random.normal(0, std, 100) for std in range(1, 4)]

# box plot rectangular
plt.boxplot(data,vert=True,patch_artist=True)

# Mostrar la imagen
plt.show()
```



https://es.wikipedia.org/wiki/Diagrama_de_caja
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html

Opciones en la generación de gráficos



Ejemplo

Escribe un programa en Python que permita ordenar un vector de números utilizando el algoritmo de ordenamiento de burbuja y visualice los pasos del proceso de ordenamiento mediante gráficos de barras.

Especificaciones:

1.El programa debe generar un vector aleatorio de números enteros.
2.Utiliza el algoritmo de ordenamiento de burbuja para ordenar el vector.
3.Durante cada paso del proceso de ordenamiento, visualiza el estado actual del vector mediante un gráfico de barras.
4.La altura de cada barra en el gráfico debe representar el valor correspondiente en el vector.
5.El gráfico debe mostrar el paso actual del proceso de ordenamiento y la iteración actual del algoritmo.
6.Después de cada paso, espera un breve período para permitir una visualización clara de los cambios en el vector.
7.Al final del proceso de ordenamiento, muestra el vector ordenado y finaliza el programa.