Programación Básica-Intermedia en Python
Clase 3

Fabián Palacios Pereira, MSc. Ing.[1]

[1] *fpalacios@fiuna.edu.py* Laboratorio de Sistemas Distribuidos,
Dpto. de Ing. Electrónica y Mecatrónica (DIEM),
Centro de Innovación Tecnológica (CITEC),
Facultad de Ingeniería de la UNA (FIUNA), Isla Bogado, Luque, Paraguay.

23 de Mayo del 2025

Resumen

En esta clase se introduce la biblioteca Matplotlib, una herramienta fundamental para la visualización de datos en Python. Se explorarán los diferentes tipos de gráficos en 2D como líneas, barras y dispersión, y se aprenderá cómo personalizar estos gráficos mediante el uso de colores, etiquetas, títulos y estilos. Posteriormente, se ampliará el conocimiento hacia gráficos en 3D, abordando cómo representar superficies, contornos y funciones matemáticas tridimensionales. Esta clase busca brindar a los estudiantes las habilidades necesarias para interpretar, representar y comunicar datos de forma gráfica y efectiva, utilizando Python como herramienta principal.

1. Entornos virtuales en Python

Un entorno virtual es una carpeta aislada donde puedes instalar versiones específicas de Python y sus paquetes sin afectar el sistema global ni otros proyectos.

¿Por qué usarlo?

- Evita conflictos entre versiones de librerías.
- Permite trabajar en múltiples proyectos con requerimientos diferentes.
- Es fácil de crear, activar y eliminar si se desea.
- Es considerada una **buena práctica** al trabajar con proyectos con Python.

Herramientas comunes
Python **incluye un módulo** llamado venv para crear entornos virtuales. También existe virtualenv, que es externo, pero venv es suficiente para casi todo.

1.1. Creando un entorno virtual paso a paso en Windows

1.1.1. Abrir la terminal (o CMD/PowerShell en Windows)

Crear una carpeta en algun directorio y ponerle un nombre a libre elección, luego con la combinación de teclas Win+R abrir el terminal escribiendo "cmd" y presionando Enter, luego ir a la ubicacion de la carpeta con el comando:

```
cd C:\Users...[ruta a tu carpeta]
```

1.1.2. Crear el entorno virtual

Con el siguiente comando de ejemplo, se crea el entorno virtual:

```
python -m venv nombre_del_entorno
```

Esto crea una carpeta con el nombre [nombre_del_entorno] con todo lo necesario para trabajar de forma aislada.

1.1.3. Activar el entorno virtual

La activación en Windows por CMD se hace con el siguiente comando:

```
nombre_del_entorno\Scripts\activate.bat
```

Despues de activar se verá algo como esto:

```
(mi_entorno) C:\ruta\proyecto>
```

Mientras esté activado, todos los paquetes que instales se guardarán **solo en ese entorno**.

1.1.4. Instalar paquetes en el entorno

Una vez activado, puedes instalar cualquier librería, por ejemplo:

```
pip install numpy
```

1.1.5. Salir del entorno virtual

El entorno virtual solo es necesario que esté activado cada vez que se quiera instalar una librería o gestionarlas. Luego de utilizarlo se puede desactivar con el simple comando de:

```
deactivate
```

Al cerrar el terminal también automáticamente se desactiva el entorno virtual.

1.1.6. Uso de entornos virtuales en Visual Studio Code

Para poder escribir código utilizando las librerías instaladas en el entorno virtual, se debe crear un archivo Pÿthon como se hace comúnmente, abrir el archivo, luego en la esquina inferior derecha hacer click sobre la versión de Python y se abrirá una pequeña barra de selección. En ésta barra lo que se realiza es seleccionar el intérprete de Python que se creó junto con el entorno virtual, por lo cual se debe presionar en “Enter interpreter path” y luego en “Find”, allí ir a la carpeta del entorno virtual, abrir la carpeta “Scripts” y seleccionar el archivo “Python”.

A partir de aquí Visual Studio Code utilizará Python desde el entorno virtual y con todas las librerías que se deseen de manera aislada.

2. Introducción a Matplotlib

Matplotlib es una biblioteca de Python especializada en la generación de gráficos estáticos, animados e interactivos. Es ampliamente utilizada en ingeniería, ciencia de datos y análisis científico, ya que permite visualizar datos de forma clara y profesional. Su módulo más conocido y usado es pyplot, el cual proporciona una interfaz sencilla para crear gráficos, similar a, por ejemplo, MATLAB.

¿Por qué usar Matplotlib?

- Es versátil: permite crear desde gráficos simples de líneas hasta representaciones tridimensionales complejas.
- Es personalizable: se puede modificar prácticamente todo del gráfico (colores, estilos, etiquetas, leyendas, tamaños, etc.).
- Se integra muy bien con NumPy: se pueden graficar directamente arreglos y funciones creadas con NumPy.
- Es de código abierto y tiene una gran comunidad de soporte.

Instalación de la librería

Para instalar la librería Matplotlib, es posible realizarlo mediante el gestor de paquetes pip, directamente desde el Terminal CMD, o en un entorno virtual (recomendado), con la siguiente línea de comando:

```
pip install matplotlib
```

Importación

Para usar Matplotlib en tus scripts o notebooks, debés importar el módulo pyplot:

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

Por convención, siempre se importa con el alias “plt”.

2.1. Gráficos de líneas

Un gráfico de líneas es una representación visual que muestra la relación entre dos variables, generalmente a lo largo del tiempo o con respecto a algún parámetro independiente.

- En el eje x se coloca la variable independiente.
- En el eje y se representa la variable dependiente.

Los puntos se conectan mediante segmentos de línea, lo que permite observar tendencias, patrones o variaciones en los datos.

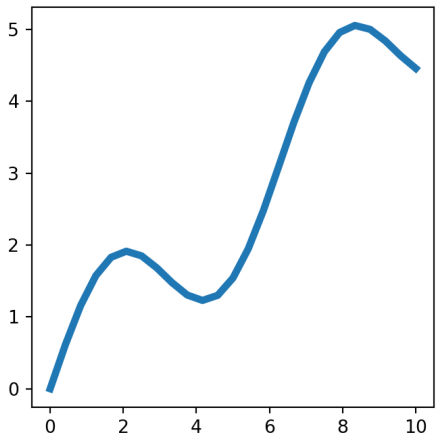

Figura 1: Gráfico de líneas simple.

¿Por qué se usan en ingeniería?

- Permiten visualizar señales en el tiempo (como corriente o voltaje).
- Son útiles para analizar comportamientos dinámicos de sistemas.
- Facilitan la comparación entre varias curvas (por ejemplo, entrada y salida de un sistema).
- Se pueden usar para representar funciones matemáticas.

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

# Datos: una funcion matematica simple
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Crear grafico
plt.plot(x, y)
plt.show()
```

Listing 1: Ejemplo de gráfico de línea

Ejemplo de gráfico con más personalización:

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

# Datos: una funcion matematica simple
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Crear grafico
plt.plot(x, y, label="y = sin(x)", color="blue",
    linestyle="-", linewidth=2)
plt.title("Grafico de la funcion seno")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()
plt.show()
```

Listing 2: Gráfico personalizado

Ejercicio 1

Dibuja la respuesta a un escalón unitario de un sistema de primer orden de la forma:

$$y(t) = 1 - e^{-t} \tag{1}$$

en el intervalo $t \in [0, 10]$ utilizando 200 puntos.

Pistas:

- Usa NumPy para crear el eje del tiempo.
- Usa np.exp() para la exponencial.
- Grafica la función con etiquetas.

Ejercicio 2
Supón que un condensador se descarga en un circuito RC. La tensión en el condensador viene dada por:

$$V(t) = V_0 \cdot e^{\frac{-t}{RC}} \tag{2}$$

Dibuja tres curvas en el mismo gráfico para tres valores distintos de R, Por ejemplo, $R = 1, R = 2, R = 3$, manteniendo $V_0 = 5$ y $C = 1$

2.2. Gráficos de barras

Los gráficos de barras son una forma de representar valores discretos o categóricos mediante rectángulos (barras), cuya longitud o altura es proporcional al valor que representan.

¿Cuándo usar gráficos de barra?

- Para comparar cantidades entre diferentes categorías.
- Para visualizar frecuencias, porcentajes o conteos.
- Ideal para datos agrupados por categorías: países, productos, tipos de sensores, etc.

```
import matplotlib.pyplot as plt

puntos = ['P1', 'P2', 'P3', 'P4', 'P5']
voltajes = [3.3, 4.1, 2.9, 3.7, 4.0]

import matplotlib.pyplot as plt

# Datos
puntos = ['P1', 'P2', 'P3', 'P4', 'P5']
voltajes = [3.3, 4.1, 2.9, 3.7, 4.0]

```

```
# Crear grafico de barras
plt.bar(puntos, voltajes, color='skyblue')

# Etiquetas
plt.xlabel("Puntos del circuito")
plt.ylabel("Voltaje (V)")
plt.title("Mediciones de voltaje en el circuito")

# Mostrar el grafico
plt.show()
```

Listing 3: Gráfico de barras básico

Es posible personalizar los gráficos, a partir de un código básico:

```
import matplotlib.pyplot as plt

puntos = ['P1', 'P2', 'P3', 'P4', 'P5']
voltajes = [3.3, 4.1, 2.9, 3.7, 4.0]

plt.bar(puntos, voltajes)
plt.show()
```

- Cambiar colores de las barras: Se puede usar un color uniforme o una lista con colores diferentes para cada barra

```
    plt.bar(puntos, voltajes, color='coral')  # Color uniforme

    # O colores individuales
    colores = ['red', 'green', 'blue', 'orange', 'purple']
    plt.bar(puntos, voltajes, color=colores)
    plt.show()
```

- Ancho de las barras: Por defecto es 0.8, pero puedes hacerlo más grueso o delgado.

```
    plt.bar(puntos, voltajes, width=0.5, color='skyblue')
    plt.show()
```

- Agregar etiquetas encima de cada barra: Para mostrar los valores exactos sobre cada barra, útil para análisis precisos.

```
    bars = plt.bar(puntos, voltajes, color='
       lightgreen')

    for barra in bars:
        altura = barra.get_height()
        plt.text(barra.get_x() + barra.get_width
           ()/2, altura + 0.05, f'{altura:.2f}',
           ha='center')

    plt.show()
```

- Cambiar estilo de borde de barrasPuedes agregar borde a las barras y cambiar el color y grosor.

```
    plt.bar(puntos, voltajes, color='lightblue',
       edgecolor='black', linewidth=1.5)
    plt.show()
```

- Agregar título y etiquetas con más formato.

```
    plt.bar(puntos, voltajes, color='
       mediumslateblue')
    plt.xlabel('Puntos del circuito', fontsize
       =12, color='navy')
    plt.ylabel('Voltaje (V)', fontsize=12, color=
       'navy')
    plt.title('Mediciones de voltaje en el
       circuito', fontsize=14, fontweight='bold')
    plt.show()
```

Ejercicio 3

Se tienen cinco sensores de temperatura instalados en una planta industrial. Se registra la temperatura media diaria (en °C) para cada sensor. Representa los valores mediante un gráfico de barras.

```
sensores = ['Sensor A', 'Sensor B', 'Sensor C', '
   Sensor D', 'Sensor E']
temperaturas = [23.5, 25.0, 22.8, 24.3, 23.9]
```

Ejercicio 4

Una estación meteorológica mide la cantidad de lluvia (en mm) por día durante una semana. Muestra la información en un gráfico de barras y resalta con un color diferente el día que tuvo mayor precipitación.

```
dias = ['Lun', 'Mar', 'Mie', 'Jue', 'Vie', 'Sab', 'Dom']
lluvia = [5, 0, 12, 8, 0, 15, 3]
```

2.3. Gráficos de dispersión

¿Qué es un gráfico de dispersión?

- Un gráfico de dispersión es una representación visual de puntos que muestra la relación entre dos variables numéricas.
- Cada punto en el gráfico representa un par (x, y).
- Es ideal para analizar patrones, correlaciones, agrupamientos y tendencias entre dos conjuntos de datos.
- Muy usado en ingeniería para representar datos experimentales, mediciones, análisis estadístico, etc.

Ejemplo:

```
import matplotlib.pyplot as plt

# Datos de ejemplo
temperatura = [20, 22, 25, 23, 21, 19, 24, 26]
presion = [101.2, 101.3, 101.8, 101.5, 101.1, 101.0, 101.7, 101.9]

# Crear grafico de dispersion
plt.scatter(temperatura, presion)
plt.show()
```

Gráfico más personalizado:

```python
import matplotlib.pyplot as plt

# Datos
temperatura = [20, 22, 25, 23, 21, 19, 24, 26]
presion = [101.2, 101.3, 101.8, 101.5, 101.1, 101.0,
    101.7, 101.9]

# Crear grafico scatter con personalizacion basica
plt.scatter(temperatura, presion, s=80, c='orange',
    alpha=0.7, edgecolors='black')

plt.xlabel('Temperatura (Celsius)')
plt.ylabel('Presion (kPa)')
plt.title('Grafico de dispersion personalizado basico'
    )

plt.show()
```

Ejercicio 5

Genera 100 puntos de datos que representen la temperatura (°C) y presión (kPa) medidas en un proceso industrial simulado. La temperatura varía entre 15 y 30 °C, la presión está relacionada con la temperatura por la fórmula:

$$presion = 100 + 0{,}5(temperatura) + ruido \tag{3}$$

donde el ruido es un valor aleatorio pequeño para simular mediciones reales (usa random()).

Grafica estos datos en un gráfico de dispersión donde:

- El tamaño de cada punto sea proporcional a la temperatura.
- El color de los puntos represente la presión, usando un mapa de colores (cmap) adecuado.
- Añade etiquetas y título a la gráfica.

Ejercicio 6

Simula un conjunto de 300 puntos de datos con tres variables relacionadas:

- x generado como 300 valores aleatorios uniformes entre 0 y 10.
- y calculado como $y = 2x + sin(x) + ruido$, donde ruido es un valor aleatorio normal con media 0 y desviación estándar 0.5.
- z calculado como $z = 3x + cos(x) + ruido$ (similar al anterior).

Grafica en un scatter 2D los puntos proyectados en el plano x vs y, donde:

- El tamaño del punto está dado por los valores absolutos de z (más grandes para valores mayores).
- El color indica el valor de z usando un mapa de colores (cmap).
- Añade una barra de colores para interpretar el color.
- Personaliza etiquetas y título.

2.4. Gráficos en 3D con Matplotlib

Matplotlib permite crear gráficos en 3 dimensiones, lo que es muy útil para visualizar funciones de dos variables o datos con tres dimensiones espaciales. Para ello, usamos el módulo "mpl_toolkits.mplot3d" que extiende Matplotlib para soportar gráficos 3D. La importación se hará de la siguiente manera:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
```

2.4.1. Gráficos de superficie (Surface plots)

Se usan para visualizar funciones $z = f(x, y)$ mostrando la superficie que genera la función.

- Representa la superficie completa que forma la función $Z = f(X, Y)$.
- Visualizas la forma tridimensional de la función como una "piel" o "manta" que cubre el espacio XY.

- Útil para ver la forma, picos y valles de la función.

Ejemplo:

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Para graficos 3D
import numpy as np

# Crear una cuadricula de puntos (x, y)
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)

# Definir la funcion z = f(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

# Crear figura y eje 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Graficar superficie
surface = ax.plot_surface(X, Y, Z, cmap='viridis')

# Agregar barra de colores
fig.colorbar(surface, shrink=0.5, aspect=5)

ax.set_title('Grafico de superficie 3D')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

2.4.2. Gráficos de contornos en 3D (Contour plots)

- Representa sólo las curvas de nivel o líneas de contorno de la función en el espacio 3D.
- Muestra las líneas donde la función tiene valores constantes (como un mapa topográfico, pero en 3D).

- Útil para analizar los niveles o valores específicos que toma la función.

Ejemplo:

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

# Crear dominio
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)

# Crear malla
X, Y = np.meshgrid(x, y)

# Definir funcion
Z = np.sin(X) * np.cos(Y)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Grafico de contornos 3D
contours = ax.contour3D(X, Y, Z, 50, cmap='viridis')

ax.set_title('Grafico de contornos 3D sencillo')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

Ejercicio 7

Representar la función:

$$z = cos(x) \cdot sin(y) \tag{4}$$

en el dominio $[-2\pi, 2\pi]$ para x e y, usando un gráfico de superficie con una barra indicadora.

Ejercicio 8

Graficar la función:

$$z = \frac{sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}} \tag{5}$$

en el dominio $[-10, 10]$ para x e y con 100 puntos por dimensión, usando gráficos de superficie y de contorno superpuestos en la misma figura.

2.5. Extra: Gráficos dinámicos

Un gráfico dinámico es un gráfico que se actualiza constantemente mientras se ejecuta un programa, permitiendo visualizar la evolución de datos o procesos iterativos. A diferencia de un gráfico estático que se dibuja una vez, el dinámico puede mostrar paso a paso cómo cambian los valores.

Esto es muy útil para:

- Visualizar algoritmos iterativos (por ejemplo, descenso de gradiente).
- Monitorizar datos que llegan en tiempo real.
- Entender mejor procesos que evolucionan con el tiempo.