# Clase 1: Introducción a Python para Ciencia de Datos

FM849 - Programación Científica para Proyectos de Inteligencia Artificial (IA)

5 de enero de 2026

# ¿Por qué Python?

- Python es un lenguaje de programación bastante flexible.
- Permite procesar grandes cantidades de datos de manera rápida.
- Permite modificar y visualizar datos de manera eficiente.
- Tiene la mayor parte de librerías de código abierto de IA y Machine Learning.

# Básicos de Python: todo es un objeto

- Todos los elementos de Python son objetos.
  - Por ejemplo, cada número, texto (*string*), tabla, etc.
- Pueden pensar un objeto como una caja con atributos y funcionalidades.
- Cada objeto tiene asociado un tipo de dato. Por ejemplo:
  - Todos los textos son del tipo *string*.
  - Todos los números enteros son del tipo *int* ($\{1, -3, 4, -66, \dots\}$).

## Ejemplo

El objeto *list* permite almacenar objetos de todo tipo.

```
>>> lista = [1, 2, 3, "cuatro"]
>>> lista.reversed()
>>> lista[2]
```

# Tipos de datos nativos o escalares

Python tiene un pequeño conjunto de datos nativos que también son llamados *escalares*. Estos son datos base que se utilizan en cualquier aplicación.

| Escalar | Descripción | Representación |
|---|---|---|
| `None` | Valor nulo | `None` |
| `str` | Datos de tipo *string* o texto | `"¡Hola, mundo!"` |
| `float` | Número de punto flotante de doble precisión | `3.1415` |
| `bool` | Un valor que puede ser *True* o *False* | `False` |
| `int` | Entero de precisión arbitraria | `33` |

Vamos a ver más ejemplos en código: Material complementario.

# Extras

- Los *imports* sirven para invocar funcionalidades de un paquete.

```
>>> import numpy as np
>>> np.array((2, 2))
```

- Los comentarios sirven para añadir información adicional al código. Pueden mejorar la legibilidad si son usados correctamente. Hay estándares que recomendamos seguir, como PEP 257 [Goodger and van Rossum, 2001].

```
>>> import matplotlib.pyplot as plt
>>> y = [3, 2, 1]
>>> plt.plot(y) # Recta que pasa por (0, 3), (1, 2) y (2, 1).
>>> plt.show() # Muestra el gráfico en pantalla.
```

# Condicionales e indentación

- Python tiene condicionales lógicos para controlar el flujo de ejecución del código.
- Para entender los condicionales, necesitamos introducir el concepto de indentación.
- Una indentación consiste en una cantidad de espacios generados con la tecla Tab.
- Al final de la línea anterior a una indentación, se deben colocar dos puntos : como marca indicadora.

## Simulación de una condición

```
>>> num = 3
>>> if num > 0:
...     print("¡Positivo!") # Con indentación.
"¡Positivo!"
```

# Condicionales

- La palabra reservada `if` simula la condición "si ocurre $X$ entonces $Y$":

```
num = 3
if num > 0:
    print("¡Positivo!")
```

- La palabra reservada `elif` es una condición alternativa que puede ocurrir después de un `if`:

```
num = -2
if num > 0:
    print("¡Positivo!")
elif num < 0:
    print("¡Negativo!")
```

# Condicionales

- La palabra reservada `else` ejecuta un bloque de código cuando todas las condiciones previas son falsas:

```
num = 0
if num > 0:
    print("¡Positivo!")
elif num < 0:
    print("¡Negativo!")
else:
    print("¡Cero!")
```

Vamos a ver más ejemplos en código: Material complementario.

# Referencias

David Goodger and Guido van Rossum. PEP 257 – docstring conventions, 2001. URL https://peps.python.org/pep-0257/.

Wes McKinney. *Python for Data Analysis*. O'Reilly Media, 3 edition, 2022.