### Trabajo de Investigación de Librerías en Python

### Librerías Investigadas: *tkinter*, *NumPy*, *SciPy*, *SymPy*

### Integrantes:

### Edison Maldonado

### Yadira Farez

### Richard Cuenca

### Cristopher Peralta

### Fecha: 20/06/2025

### Docente: Ing. Franklin Bermeo, MBA, PhD(e)

### ****¿Qué son las librerías?****

En programación, una **librería** es un conjunto de **códigos reutilizables** diseñados para resolver tareas específicas o comunes en el desarrollo de software, sin necesidad de que el programador escriba esas funciones desde cero. Estas librerías contienen **módulos, funciones, clases y estructuras de datos** que pueden ser incorporadas en programas más grandes para extender su funcionalidad.

Las librerías se crean con el objetivo de **ahorrar tiempo, mejorar la eficiencia y reducir errores**, ya que han sido probadas por comunidades o desarrolladores expertos. También permiten que el código sea más limpio, modular y mantenible. En muchos lenguajes, estas librerías están disponibles de forma gratuita y se pueden instalar fácilmente mediante gestores de paquetes como pip en Python, npm en JavaScript o Maven en Java.

En resumen, una librería:

* Evita tener que reinventar la rueda.
* Aporta soluciones listas para usar.
* Permite que los desarrolladores se enfoquen más en la lógica del problema específico que quieren resolver.

**Descripción técnica de la librería**

**. Descripción técnica de las librerías**

**a) tkinter** es la biblioteca estándar de Python para el desarrollo de interfaces gráficas (GUI). Permite construir ventanas, botones, campos de texto y otros elementos interactivos de forma sencilla y rápida. Es multiplataforma y viene incluida con la instalación estándar de Python.

Funciones principales:

* Creación de ventanas (Tk())
* Etiquetas (Label), campos de entrada (Entry), botones (Button)
* Eventos y funciones de respuesta (command)
* Disposición de widgets (pack, grid)

**b) NumPy** NumPy es la biblioteca fundamental para el cálculo numérico en Python. Permite trabajar con arrays multidimensionales y realizar operaciones vectorizadas, lo que la hace muy eficiente para grandes volúmenes de datos.

Funciones comunes:

* numpy.array: Crear arreglos
* numpy.dot: Producto escalar y matricial
* numpy.linalg.inv: Inversa de una matriz
* numpy.mean, numpy.std: Estadísticas básicas

Sitio oficial: https://numpy.org/doc

**c) SciPy** SciPy amplía las funcionalidades de NumPy e incluye módulos para optimización, integración, álgebra lineal avanzada, procesamiento de señales y resolución de ecuaciones diferenciales.

Módulos clave:

* scipy.linalg: Álgebra lineal
* scipy.integrate: Integración y EDOs
* scipy.optimize: Optimización
* scipy.fft: Transformadas de Fourier

Sitio oficial: https://docs.scipy.org/doc/scipy

**d) SymPy** SymPy es una librería de matemática simbólica en Python. Permite trabajar con expresiones algebraicas, derivadas, integrales, ecuaciones y matrices de forma exacta (no numérica).

Funciones principales:

* symbols: Definición de variables
* diff, integrate: Derivación e integración simbólica
* solve: Resolución de ecuaciones
* simplify: Simplificación de expresiones

**4. Instrucciones de instalación**

* tkinter: **Preinstalado** con Python. Verificar con python -m tkinter
* numpy, scipy, sympy: Instalar con pip:

py -m pip install ……

**5. Ejercicios**

**1.**

import tkinter as tk

def calcular\_resistencia():

    try:

        V = float(entry\_voltaje.get())

        I = float(entry\_corriente.get())

        R = V / I

        label\_resultado.config(text=f"Resistencia: {R:.2f} Ω")

    except:

        label\_resultado.config(text="Error en los datos")

ventana = tk.Tk()

ventana.title("Calculadora de Resistencia (Ley de Ohm)")

tk.Label(ventana, text="Voltaje (V):").pack()

entry\_voltaje = tk.Entry(ventana)

entry\_voltaje.pack()

tk.Label(ventana, text="Corriente (I):").pack()

entry\_corriente = tk.Entry(ventana)

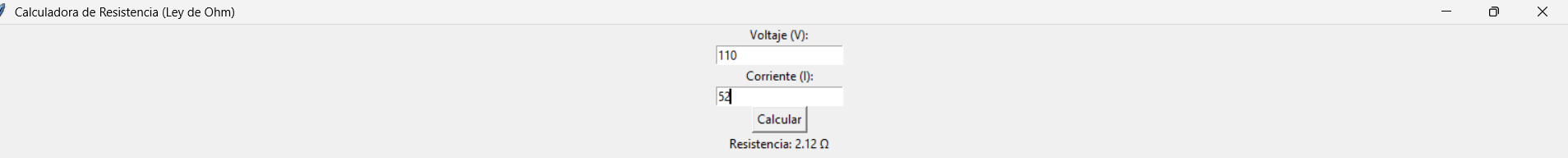
entry\_corriente.pack()

tk.Button(ventana, text="Calcular", command=calcular\_resistencia).pack()

label\_resultado = tk.Label(ventana, text="Resistencia: ")

label\_resultado.pack()

ventana.mainloop()

****

**2.**

import numpy as np

K = np.array([[12, -6], [-6, 4]])

F = np.array([100, 50])

U = np.linalg.solve(K, F)

print("Desplazamientos calculados:", U)

****

**3.**

from scipy.integrate import solve\_ivp

import numpy as np

import matplotlib.pyplot as plt

def modelo(t, x):

    return -2 \* x

sol = solve\_ivp(modelo, [0, 10], [1], t\_eval=np.linspace(0, 10, 100))

plt.plot(sol.t, sol.y[0])

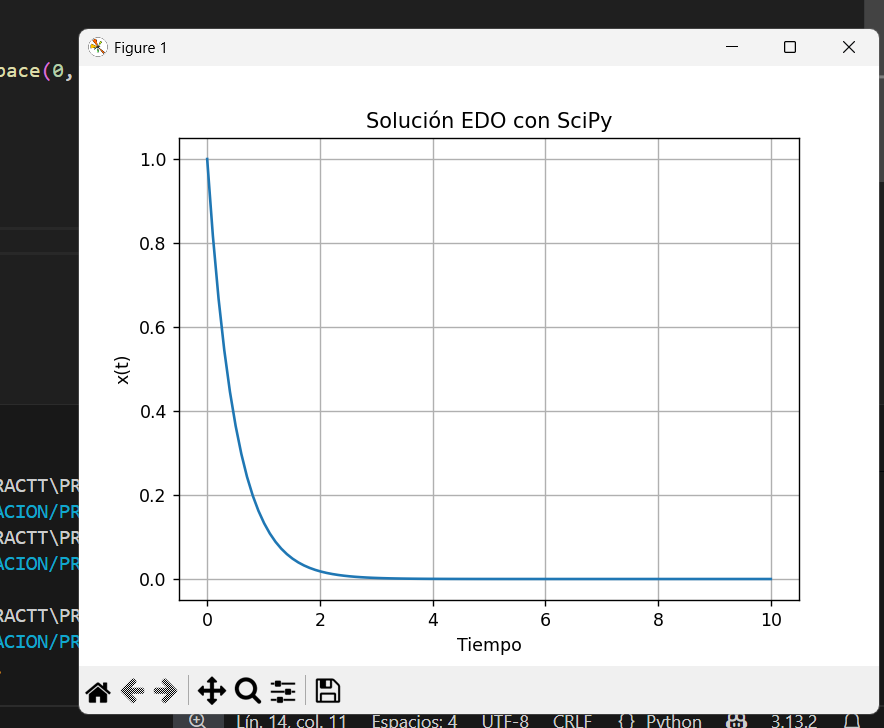
plt.title("Solución EDO con SciPy")

plt.xlabel("Tiempo")

plt.ylabel("x(t)")

plt.grid()

plt.show()

****

**4.**

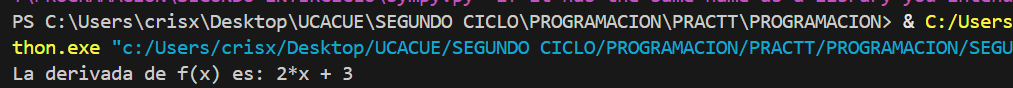
import sympy as sp

x = sp.symbols('x')

f = x\*\*2 + 3\*x + 2

dfdx = sp.diff(f, x)

print("La derivada de f(x) es:", dfdx)

****

**5.**

import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 400)

y = x\*\*2 - 4\*x + 5

plt.plot(x, y, label='f(x) = x^2 - 4x + 5')

plt.axhline(0, color='gray', linestyle='--')

plt.axvline(0, color='gray', linestyle='--')

plt.title('Gráfico de función cuadrática')

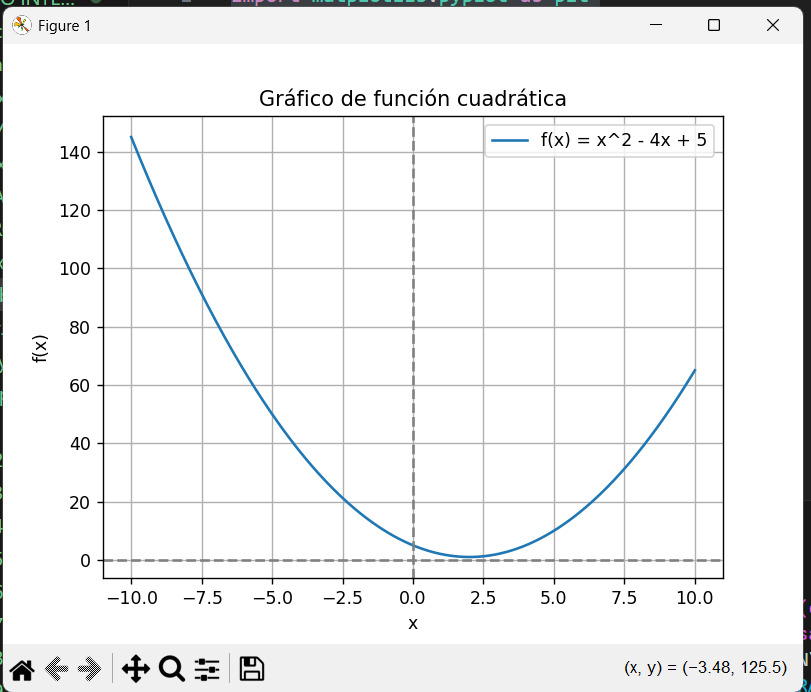
plt.xlabel('x')

plt.ylabel('f(x)')

plt.legend()

plt.grid(True)

plt.show()

****

**8. Conclusiones del equipo**

* Las librerías analizadas cubren aspectos fundamentales del desarrollo científico y técnico.
* Su integración permite crear simulaciones, resolver ecuaciones y presentar resultados con interfaz gráfica.
* Aprender a utilizarlas es vital para la automatización de tareas en ingeniería.

**9. Referencias utilizadas**

1. https://docs.python.org/3/library/tkinter.html
2. https://numpy.org/doc
3. https://docs.scipy.org/doc/scipy
4. https://docs.sympy.org
5. https://realpython.com