```python
# Importamos las librerias necesarias para la interfaz y los cálculos
import tkinter as tk  # Para crear la interfaz gráfica
from tkinter import ttk  # Para widgets con estilos (botones, tablas, etc.)
from tkinter import messagebox  # Para mostrar mensajes emergentes (errores, alertas)
import numpy as np  # Para manejar arreglos y operaciones matemáticas
import matplotlib.pyplot as plt  # Para graficar
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg  # Para insertar gráficas en Tkinter
import math  # Para funciones matemáticas como e, sin, cos, etc.
import re  # Para expresiones regulares

# Creamos una clase que contiene toda la lógica y la interfaz
class RegresionLinealApp:
    def __init__(self, root):
        # Inicializamos la ventana principal
        self.root = root
        self.root.title("Regresión Lineal RGB Gamer Style - Ultra Flexible")  # Título de la ventana
        self.root.geometry("900x700")  # Tamaño inicial de la ventana (más alto para más contenido)
        self.root.configure(bg='#0f0f0f')  # Fondo oscuro tipo gamer

        # Estilos personalizados tipo gamer (colores neón sobre fondo oscuro)
        style = ttk.Style()
        style.theme_use('clam')  # Usamos el tema 'clam' para personalizar los colores

        # Configuramos el estilo de los botones
        style.configure("TButton",
                        font=("Arial", 12),
                        foreground="lime",  # Texto en verde neón
                        background="#1f1f1f",  # Fondo oscuro
                        padding=10)

        # Configuramos el estilo de la tabla tipo Treeview
        style.configure("Treeview",
                        background="#1a1a1a",  # Fondo de las celdas
                        foreground="cyan",  # Texto en color cian
                        fieldbackground="#1a1a1a",
                        rowheight=25,
                        font=("Courier", 10))

        # Efecto visual cuando se pasa el mouse sobre el botón
        style.map("TButton", background=[('active', '#00ffcc')])

        # Llamamos a la función que arma toda la interfaz
        self.create_widgets()
```

```python

    def evaluar_expresion(self, expresion, x_val=None):
        """Evalúa expresiones matemáticas de manera segura"""
        try:
            # Reemplazamos ^ por ** para exponenciación en Python
            expresion = expresion.replace('^', '**')

            # Definimos constantes y funciones matemáticas permitidas
            contexto_seguro = {
                'e': math.e,
                'pi': math.pi,
                'sin': math.sin,
                'cos': math.cos,
                'tan': math.tan,
                'log': math.log,
                'ln': math.log,
                'sqrt': math.sqrt,
                'abs': abs,
                'exp': math.exp,
                '__builtins__': {}  # Removemos funciones peligrosas
            }

            # Si se proporciona un valor para x, lo agregamos al contexto
            if x_val is not None:
                contexto_seguro['x'] = x_val
                contexto_seguro['X'] = x_val

            # Evaluamos la expresión de manera segura
            resultado = eval(expresion, contexto_seguro)
            return float(resultado)

        except Exception as e:
            raise ValueError(f"Error al evaluar '{expresion}': {str(e)}")

    def parsear_punto(self, punto_str):
        """Parsea un punto que puede contener expresiones matemáticas"""
        # Removemos espacios y paréntesis externos
        punto_str = punto_str.strip()
        if punto_str.startswith('(') and punto_str.endswith(')'):
            punto_str = punto_str[1:-1]
```

```
84
85          # Verificamos si hay una coma para separar x,y
86          if ',' in punto_str:
87              # Formato estándar (x,y)
88              partes = punto_str.split(',', 1)  # Dividimos solo por la primera coma
89              x_expr = partes[0].strip()
90              y_expr = partes[1].strip()
91          else:
92              # Formato de expresión única - asumimos que es un valor de y, y x será un contador
93              x_expr = "0"  # Valor predeterminado, se actualizará después
94              y_expr = punto_str.strip()
95
96          # Si es un número simple, lo convertimos directamente
97          try:
98              x = float(x_expr)
99          except ValueError:
100             # Si no es un número, evaluamos como expresión
101             x = self.evaluar_expresion(x_expr)
102
103         try:
104             y = float(y_expr)
105         except ValueError:
106             # Si no es un número, evaluamos como expresión
107             # Para y, podemos usar el valor de x si la expresión lo requiere
108             y = self.evaluar_expresion(y_expr, x)
109
110         return (x, y)
111
112     def create_widgets(self):
113         """Crea todos los widgets visibles: entrada de puntos, botón, tabla y etiqueta de resultado"""
114         # Etiqueta que indica al usuario qué hacer
115         self.label = tk.Label(self.root, text="Ingresa los puntos (x, y) o expresiones matemáticas:",
116                               fg="white", bg="#0f0f0f", font=("Arial", 14))
117         self.label.pack(pady=10)
118
119         # Etiqueta de ayuda
120         self.help_label = tk.Label(self.root,
121                                    text="Ejemplos: (2,4), (e^2, 5), e^x2, sin(pi/2), (2, 7)",
122                                    fg="yellow", bg="#0f0f0f", font=("Arial", 10))
123         self.help_label.pack(pady=5)
```

```
124
125         # Cuadro de texto para ingresar los puntos manualmente
126         self.text_entry = tk.Text(self.root, height=5, width=60,
127                                   bg="#1a1a1a", fg="white", insertbackground='white')
128         self.text_entry.insert(tk.END, "(2,4), e^2, (0, e^2-4), sin(pi/2)")  # Puntos de ejemplo con expresiones
129         self.text_entry.pack(pady=10)
130
131         # Botón que calcula la regresión cuando se presiona
132         self.calc_button = ttk.Button(self.root, text="Calcular Recta de Regresión", command=self.calcular_regresion)
133         self.calc_button.pack(pady=10)
134
135         # Tabla donde se mostrarán los datos calculados: x, y, xy y x²
136         self.tree = ttk.Treeview(self.root, columns=("x", "y", "xy", "x^2"), show="headings")
137         for col in ("x", "y", "xy", "x^2"):
138             self.tree.heading(col, text=col)
139             self.tree.column(col, width=120)
140         self.tree.pack(pady=20)
141
142         # Etiqueta donde se mostrará la ecuación final de la regresión
143         self.resultado_label = tk.Label(self.root, text="", fg="white", bg="#0f0f0f", font=("Arial", 14))
144         self.resultado_label.pack()
145
146         # Etiqueta para mostrar los puntos evaluados
147         self.puntos_label = tk.Label(self.root, text="", fg="cyan", bg="#0f0f0f", font=("Arial", 10),
148                                      wraplength=800)  # Permitimos que el texto se envuelva
149         self.puntos_label.pack(pady=5)
150
151     def extraer_puntos_del_texto(self, texto):
152         """Extrae puntos del texto ingresado en varios formatos posibles"""
153         puntos = []
154
155         # Primero buscamos patrones de puntos con paréntesis (x,y)
156         patron_puntos = r'$$[^)]+$$'
157         puntos_con_parentesis = re.findall(patron_puntos, texto)
158
159         # Procesamos los puntos con paréntesis
160         for punto_str in puntos_con_parentesis:
161             puntos.append(punto_str)
162             # Removemos el punto procesado del texto
163             texto = texto.replace(punto_str, '', 1)
164
165         # Ahora procesamos el texto restante para buscar expresiones sueltas
166         # Dividimos por comas y espacios
167         expresiones_sueltas = [exp.strip() for exp in re.split(r'[,\s]+', texto) if exp.strip()]
168
```

```
        # Agregamos las expresiones sueltas como puntos
        puntos.extend(expresiones_sueltas)

        return puntos

    def calcular_regresion(self):
        """Función principal que calcula la regresión lineal"""
        try:
            # Obtenemos el texto ingresado
            texto = self.text_entry.get("1.0", tk.END).strip()

            if not texto:
                messagebox.showerror("Error", "Por favor ingresa al menos dos puntos o expresiones")
                return

            # Extraemos los puntos del texto en varios formatos posibles
            puntos_str = self.extraer_puntos_del_texto(texto)

            if not puntos_str:
                raise ValueError("No se encontraron puntos o expresiones válidas")

            puntos = []
            puntos_evaluados_str = []

            # Procesamos cada punto o expresión
            contador_x = 0  # Para asignar valores de x a expresiones sueltas
            for punto_str in puntos_str:
                try:
                    # Si es una expresión suelta (sin formato de punto), le asignamos un valor de x
                    if not (punto_str.startswith('(') and ')' in punto_str):
                        punto_str = f"{contador_x},{punto_str}"
                        contador_x += 1

                    x, y = self.parsear_punto(punto_str)
                    puntos.append((x, y))
                    puntos_evaluados_str.append(f"({x:.3f}, {y:.3f})")
                except Exception as e:
                    messagebox.showerror("Error", f"Error en '{punto_str}': {str(e)}")
                    return

            if len(puntos) < 2:
                messagebox.showerror("Error", "Se necesitan al menos 2 puntos para calcular la regresión")
                return
```

```
212
213             # Mostramos los puntos evaluados
214             self.puntos_label.config(text=f"Puntos evaluados: {', '.join(puntos_evaluados_str)}")
215
216             # Separamos las coordenadas x e y
217             x_vals = np.array([p[0] for p in puntos])
218             y_vals = np.array([p[1] for p in puntos])
219
220             # Limpiamos la tabla anterior
221             for item in self.tree.get_children():
222                 self.tree.delete(item)
223
224             # Calculamos los valores necesarios para la regresión
225             xy_vals = x_vals * y_vals
226             x2_vals = x_vals ** 2
227
228             # Llenamos la tabla con los datos
229             for i in range(len(x_vals)):
230                 self.tree.insert("", "end", values=(
231                     f"{x_vals[i]:.4f}",
232                     f"{y_vals[i]:.4f}",
233                     f"{xy_vals[i]:.4f}",
234                     f"{x2_vals[i]:.4f}"
235                 ))
236
237             # Calculamos las sumas necesarias
238             n = len(x_vals)
239             sum_x = np.sum(x_vals)
240             sum_y = np.sum(y_vals)
241             sum_xy = np.sum(xy_vals)
242             sum_x2 = np.sum(x2_vals)
243
244             # Fórmulas de regresión lineal
245             # m = (n*Σxy - Σx*Σy) / (n*Σx² - (Σx)²)
246             # b = (Σy - m*Σx) / n
247             denominador = n * sum_x2 - sum_x ** 2
248             if abs(denominador) < 1e-10:
249                 messagebox.showerror("Error", "No se puede calcular la regresión (denominador muy pequeño)")
250                 return
251
252             m = (n * sum_xy - sum_x * sum_y) / denominador
253             b = (sum_y - m * sum_x) / n
254
```

```
            # Mostramos la ecuación final redondeada
            self.resultado_label.config(text=f"Ecuación: Y = {m:.4f}X + {b:.4f}")

            # Llamamos a la función para mostrar la gráfica
            self.mostrar_grafica(x_vals, y_vals, m, b)

        except Exception as e:
            # Si hay algún error, mostramos una alerta
            messagebox.showerror("Error", f"Error: {str(e)}")

    def mostrar_grafica(self, x_vals, y_vals, m, b):
        """Dibuja una gráfica de dispersión con la recta de regresión sobrepuesta"""
        # Limpiamos gráficas anteriores
        for widget in self.root.winfo_children():
            if isinstance(widget, tk.Widget) and hasattr(widget, 'get_tk_widget'):
                widget.destroy()

        fig, ax = plt.subplots(figsize=(6, 4))  # Creamos la figura

        # Dibujamos los puntos
        ax.scatter(x_vals, y_vals, color='cyan', s=50, zorder=5)  # Puntos en cian

        # Creamos una línea suave para la regresión
        x_min, x_max = min(x_vals), max(x_vals)
        x_range = x_max - x_min
        x_line = np.linspace(x_min - 0.1*x_range, x_max + 0.1*x_range, 100)
        y_line = m * x_line + b

        # Dibujamos la recta de regresión
        ax.plot(x_line, y_line, color='lime', linestyle='--', linewidth=2)  # Línea en verde

        # Agregamos etiquetas a los puntos
        for i, (x, y) in enumerate(zip(x_vals, y_vals)):
            ax.annotate(f'P{i+1}', (x, y), xytext=(5, 5), textcoords='offset points',
                        color='white', fontsize=8)
```

```
        # Personalizamos el fondo y los ejes para que luzca "gamer"
        ax.set_facecolor('#1a1a1a')
        fig.patch.set_facecolor('#0f0f0f')
        ax.set_title('Regresión Lineal Ultra Flexible', color='white', fontsize=12)
        ax.set_xlabel('X', color='white')
        ax.set_ylabel('Y', color='white')
        ax.tick_params(colors='white')
        ax.spines['bottom'].set_color('white')
        ax.spines['top'].set_color('white')
        ax.spines['left'].set_color('white')
        ax.spines['right'].set_color('white')
        ax.grid(True, alpha=0.3, color='gray')

        # Insertamos la gráfica en la interfaz gráfica
        canvas = FigureCanvasTkAgg(fig, master=self.root)
        canvas.draw()
        canvas.get_tk_widget().pack(pady=10)

# Código principal: lanzamos la aplicación
if __name__ == '__main__':
    root = tk.Tk()
    app = RegresionLinealApp(root)
    root.mainloop()
```