```
#!/usr/bin/env python
#
# Distribución de Erlang
#

from tkinter import *
from tkinter import ttk, messagebox
import tkinter as tk
import numpy as np


class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.geometry("310x450")
        self.title("Distribución de Erlang")
        # Se inicializa la información
        self.parametro_k = tk.IntVar()  # Valor de k
        self.parametro_lambda = tk.DoubleVar()  # Valor de la media
        self.valor_minimo = tk.DoubleVar()  # Valor a calcular
        self.valor_maximo = tk.DoubleVar()  # Por si el cálculo es de intervalo
        self.simulacion = tk.StringVar()  # Tipo de simulación a realizar
        self.solucion = tk.DoubleVar()  # Se devuelve la solución
        # Crear widgets
        self.crear_widgets()

    def crear_widgets(self):
        datos = Frame(self, relief=SUNKEN)
        datos.pack(fill=tk.X)
        #
        # Captura de información
        #
        ttk.Label(datos, text="Indique el parámetro de forma (k)", justify=LEFT,
background="#C1E1C1").pack(
            anchor=tk.W, padx=10, pady=5, fill=tk.X
        )
        k = Entry(datos, textvariable=self.parametro_k)
        k.pack(anchor=tk.W, padx=10, pady=5, fill=tk.X)
        ttk.Label(datos, text="Indique el parámetro de escala (media o lambda)",
justify=LEFT,
                  background="#C1E1C1").pack(anchor=tk.W, padx=10, pady=5,
fill=tk.X
        )
        parametro_media = Entry(datos, textvariable=self.parametro_lambda)
        parametro_media.pack(anchor=tk.W, padx=10, pady=5, fill=tk.X)
        ttk.Label(datos, text="Tipo de cálculo a realizar", justify=LEFT,
background="#C1E1C1").pack(
            anchor=tk.W, padx=10, pady=5, fill=tk.X
        )
        seleccion = ttk.Combobox(datos, textvariable=self.simulacion,
                                 state='readonly', values=["<", "<=", ">", ">=",
"a<=x<=b"])
        seleccion.pack(anchor=tk.W, padx=10, pady=5, fill=tk.X)
        seleccion.current()
```

```
        ttk.Label(datos, text="Indique el valor del cálculo de probabilidad",
                  justify=LEFT, background="#C1E1C1").pack(
            anchor=tk.W, padx=10, pady=5, fill=tk.X
        )
        dato_inicial = Entry(datos, textvariable=self.valor_minimo)
        dato_inicial.pack(anchor=tk.W, padx=10, pady=5, fill=tk.X)
        ttk.Label(datos, text="Indique el valor final del intervalo (si
aplica)",
                  justify=LEFT, background="#C1E1C1").pack(
            anchor=tk.W, padx=10, pady=5, fill=tk.X
        )
        dato_final = Entry(datos, textvariable=self.valor_maximo)
        dato_final.pack(anchor=tk.W, padx=10, pady=5, fill=tk.X)
        #
        # Termina captura de información
        #
        # ############################
        # Creación de botones
        #
        style = ttk.Style()
        style.theme_use("alt")
        style.configure('TButton', background="blue", foreground="yellow")
        style.map('TButton', background=[('active', 'red')])
        ttk.Button(datos, text="Calcular", command=lambda:
self.calcular()).pack(side=tk.LEFT, padx=10, pady=5)
        ttk.Button(datos, text="Salir", command=lambda:
self.quit()).pack(side=tk.LEFT, padx=10, pady=5)
        #
        # Se muestra la solución
        #
        salida = Frame(self)
        salida.pack(fill=X)
        salida.configure(bg="gray")
        ttk.Label(salida, text="Solución",
                  justify=LEFT, background="gray").pack(
            anchor=tk.W, padx=10, pady=5, fill=tk.X
        )
        valor_probabilidad = Entry(salida, textvariable=self.solucion)
        valor_probabilidad.pack(anchor=tk.W, padx=10, pady=5, fill=tk.X)

    @staticmethod
    def lectura(combo):
        switch = {
            '<': 1,
            '<=': 2,
            '>': 3,
            '>=': 4,
            'a<=x<=b': 5
        }
        return switch.get(combo, 'e')

    def simular(self):
        valores = (1/self.parametro_k.get()) * np.random.gamma(
            self.parametro_k.get(),
```

```
            self.parametro_lambda.get(),
            [10, 5, 365]
        )
        valores = valores.flatten().tolist()
        tipo_calculo = self.lectura(self.simulacion.get())
        suma = 0
        if tipo_calculo == 1:
            for j in valores:
                suma = suma + 1 if j < self.valor_minimo.get() else suma + 0
        elif tipo_calculo == 2:
            for j in valores:
                suma = suma + 1 if j <= self.valor_minimo.get() else suma + 0
        elif tipo_calculo == 3:
            for j in valores:
                suma = suma + 1 if j > self.valor_minimo.get() else suma + 0
        elif tipo_calculo == 4:
            for j in valores:
                suma = suma + 1 if j >= self.valor_minimo.get() else suma + 0
        else:
            for j in valores:
                suma = suma + 1 if self.valor_minimo.get() <= j <=
self.valor_maximo.get() else suma + 0
        probabilidad = round((suma / len(valores)) * 100, 2)
        self.solucion.set(probabilidad)

    def calcular(self):
        bandera = 0
        if not self.parametro_k.get():
            messagebox.showerror("Error de media",
                                 "Se debe declarar el valor promedio")
        else:
            bandera += 1
        if not self.parametro_lambda.get():
            messagebox.showerror("Error de ingreso",
                                 "Se debe declarar a la desviación estándar")
        else:
            if self.parametro_lambda.get() <= 0:
                messagebox.showerror("Error de desviación",
                                     "La desviación es una distancia y no puede
ser negativa")
            else:
                bandera += 1
        if not self.simulacion.get():
            messagebox.showerror("Selección de probabilidad",
                                 "Debe indicar el tipo de cálculo a realizar")
        else:
            bandera += 1
            tipo_calculo = self.lectura(self.simulacion.get())
            if tipo_calculo == 5:
                if not self.valor_maximo.get():
                    messagebox.showerror("Declaración de intervalo",
                                         "Debe indicar el valor final del
intervalo")
                else:
```

```
                    if self.valor_maximo.get() <= self.valor_minimo.get():
                        messagebox.showerror("Declaración de intervalo",
                                             "El valor final no puede ser menor
al valor inicial")
                    else:
                        bandera += 2
            else:
                if not self.valor_minimo.get():
                    messagebox.showerror("Cálculo de probabilidad",
                                         "Requiere indicar el valor por
calcular")
                else:
                    bandera += 1
            if bandera >= 4:
                self.simular()


if __name__ == '__main__':
    app = App()
    app.mainloop()
```