



Calculadora gráfica en Python

Sofia Molano Martínez

Sergio Alexander Melendez Rodriguez

Diego Andres Torres Losada

5 de septiembre del 2024

Índice

Índice.....	1
1. Objetivo general.....	2
2. Objetivos específicos.....	2
3. Requerimientos Técnicos Software y Hardware.....	2
4. Herramientas utilizadas para el desarrollo.....	3
4.1 Librerías utilizadas.....	3
5. Instalación.....	4
6. Diseño de la arquitectura.....	5
7. Conexión con Firebase.....	6
8. Análisis de código.....	6
8.1. CRUD.....	6
8.2. Inicio de sesión y registro.....	8
8.3. Mostrar la calculadora.....	11
8.4. Realizar operaciones.....	13
8.5. Graficar en 2d y 3d.....	15
8.5.1 VIEW_ventana_grafica.....	15
8.5.2 v_grafica_funcionamiento.....	18
9. Diagramas de casos de uso.....	20
10. Diagramas de flujo de funciones.....	22
10.1. Función ingresar().....	22
10.2. Función de graficar.....	23
11. Excepciones.....	24
12. Información a tener en cuenta.....	25
13. Conclusiones.....	25



1. Objetivo general

La creación de una calculadora en el sistema windows 10 programada en python con la capacidad de cumplir con las funciones de calcular operaciones básicas, operaciones trigonométricas, recibir funciones y graficarlas (en 2d o 3d), y utilizar la base de datos y autenticación de Firebase para la persistencia de datos y CRUD del historial por cada usuario.

2. Objetivos específicos

- Investigar sobre temas relacionados a la programación de la calculadora como lo son arquitecturas de programación, librerías de matemáticas, gráficas, y ventana del programa, funciones, métodos etc.
- Diseñar la calculadora para que cumpla con la capacidad de realizar las operaciones básicas, trigonométricas, funciones y gráficas 2d y 3d, así como realizar el CRUD del historial con una base de datos Firebase.
- Programar la calculadora mediante la aplicación de los conocimientos adquiridos de los temas relacionados y el diseño creado según los requerimientos.

3. Requerimientos Técnicos Software y Hardware

Software:

Entornos de desarrollo como Visual Studio Code

Lenguaje de programación Python

Gestor de bases de datos conectada (Firebase)

Hardware:

Una computadora completa (parlantes no necesarios) esto incluye: Mouse, teclado, monitor, cpu. Conexión a internet WiFi o Datos Móviles.

4. Herramientas utilizadas para el desarrollo

Navegador Web: Para la verificación de la base de datos (Firebase).



Gestor de bases de datos: FireBase para la administración de los datos almacenados.

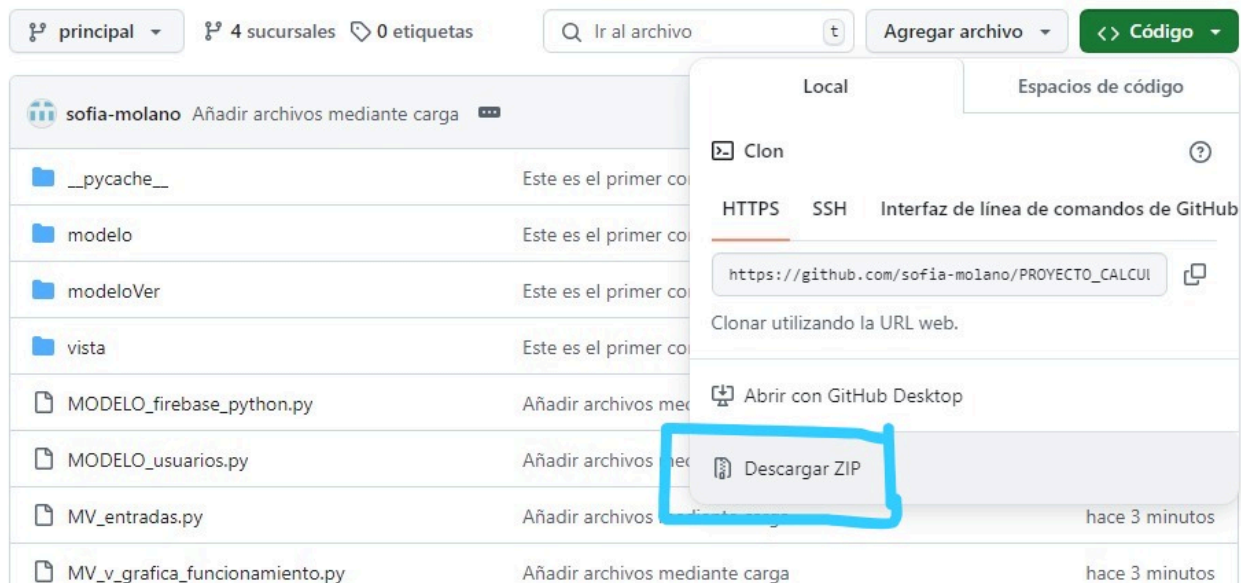
Studio V.S.C 1.92.2 para la modificación y/o actualización del código fuente incluyendo diseño.

4.1 Librerías utilizadas

- Tkinter- Biblioteca estándar de Python para crear interfaces gráficas de usuario (GUIs)
- Numpy- Librería de Python especializada en el cálculo numérico y el análisis de datos.
- Firebase Admin- Permite a los desarrolladores integrar Firebase en sus servicios y aplicaciones.
- Matplotlib- Es una biblioteca de Python para la creación de gráficos y visualizaciones.

5. Instalación

1. Ve a la página principal del repositorio en GitHub.
2. Haz clic en el botón verde que dice `Code` en la parte superior derecha.
3. Selecciona `Download ZIP` del menú desplegable.
4. Una vez descargado el archivo, extráelo en tu computadora.
5. Crea un proyecto en firebase, genere una nueva clave única en formato .json.
6. Copia y pega la contraseña como un archivo en el proyecto y reemplaza las direcciones donde sea necesario.



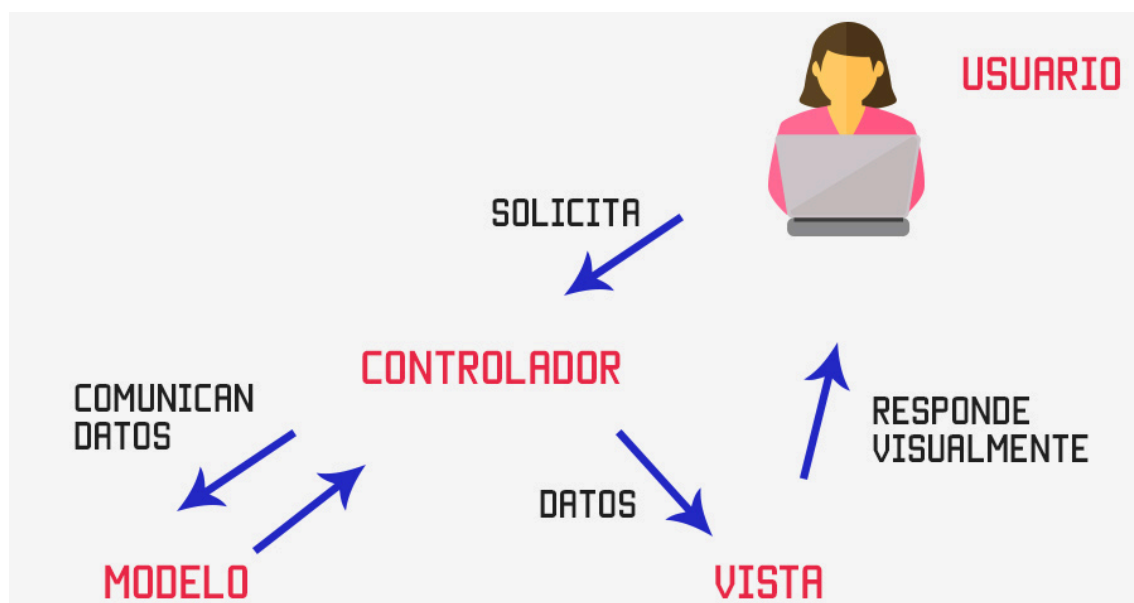
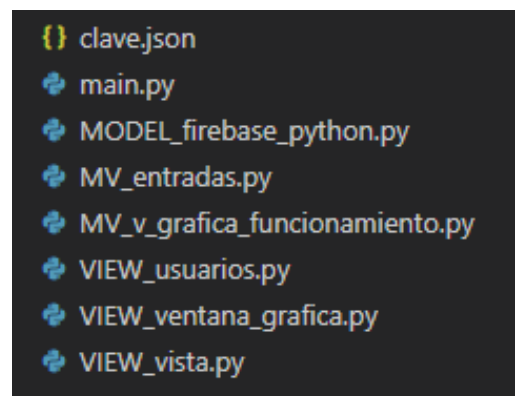
6. Diseño de la arquitectura

El patrón de arquitectura usado fue MVC o Modelo-Vista-Controlador para separar la lógica de la aplicación así:

Model (Modelo): Contiene la estructura de datos y la lógica para manipularlos.

View (Vista): Contiene la interfaz gráfica para el usuario (GUI), le presenta visualmente la información al usuario.

View Model (Controlador): Unión entre las dos, controla la comunicación y el funcionamiento. Para facilitar la importación de archivos se colocó al inicio de cada nombre a qué componente pertenece.



7. Conexión con Firebase

Firebase genera una clave única guardada en un archivo .json para el proyecto y nos da el código para la conexión según el lenguaje de programación que se use, en el caso Python.

Luego se implementaron las funciones de Realtime Database para guardar los datos de las operaciones realizadas por el usuario y Authentication para permitir a los usuarios tener su propia cuenta.

☐ Node.js ☐ Java ☒ Python ☐ Go

```
import firebase_admin
from firebase_admin import credentials

cred = credentials.Certificate("path/to/serviceAccountKey.json")
firebase_admin.initialize_app(cred)
```

8. Análisis de código

8.1. CRUD

El archivo MODEL_firebase_python.py es el encargado principal de la configuración y comunicación de firebase. A partir de ello realiza las funciones CRUD, Crear, Leer, Actualizar, Borrar de la base de datos Firebase.

Importa librerías y crea 2 variables globales: -"user_uid" y "correo". La primera se encarga de guardar el código de Identificación del usuario, el segundo del Correo.

Con `sanitize_path` se limpian los caracteres comúnmente usados por guión bajo "_" para que se pueda leer por parte de firebase.

```
from firebase_admin import db
import numpy as np
import tkinter as tk
from tkinter import messagebox
import re

user_uid = None
correo = None

def sanitize_path(path):
    return re.sub(r'[/\#\$\%[\]\.\@]', '_', path)

def configurar_usuario(uid, email):
    global user_uid
    global correo
    user_uid = uid
    correo = email
    sanitized_email = sanitize_path(correo)

    # Obtener la referencia al nodo específico del usuario
    ref = db.reference(f'usuarios/{sanitized_email}')
```

`configurar_usuario` configura y depura el usuario actual almacenando su UID y correo electrónico, y prepara una referencia a su nodo en Firebase para asignar el historial según el usuario.

Para realizar las funciones CRUD:

RealtimeSet Almacena una nueva entrada de cálculo (expresión y resultado) en la base de datos de Firebase con “entrada1” y “resultado” respectivamente de la calculadora.

La depuración, referencia y la comprobación de “user_uid” y “correo” se hace en todas las siguientes funciones CRUD.

```
def RealtimeSet(entrada1, resultado):  
    if not user_uid or not correo:  
        raise ValueError("Debe iniciar sesión primero.")  
  
    sanitized_email = sanitize_path(correo)  
    ref = db.reference(f'usuarios/{sanitized_email}')  
    nueva_entrada = ref.push()  
    nueva_entrada.set({  
        'expresion': entrada1,  
        'resultado': resultado  
    })
```

RealtimeGet Obtiene el historial lista respectivo del usuario y lo muestra en la calculadora en caso de tenerlo.

RealtimeDelete Borra totalmente todo el historial del usuario.

```
def RealtimeGet():  
    if not user_uid:  
        raise ValueError("Debe iniciar sesión primero.")  
  
    sanitized_email = sanitize_path(correo)  
    ref = db.reference(f'usuarios/{sanitized_email}')  
    historial = ref.get()  
    historial_lista = []  
  
    if historial:  
        for key, x in reversed(list(historial.items())):  
            if isinstance(x, dict):  
                expresion = x.get('expresion', 'Desconocida')  
                resultado = x.get('resultado', 'Datos de gráfica')  
            else:  
                expresion = x  
                resultado = 'Los datos de la gráfica no se muestran en pantalla'  
            historial_lista.append(f"(expresion) = (resultado)")  
  
    return historial_lista  
  
def RealtimeDelete():  
    try:  
        sanitized_email = sanitize_path(correo)  
        ref = db.reference(f'usuarios/{sanitized_email}')  
        ref.delete()  
        messagebox.showinfo("Información", "Los datos del historial han sido eliminados correctamente.")  
    except Exception as e:  
        messagebox.showerror("Error", f"Ha ocurrido un error: {str(e)}")
```

El usuario no solamente puede almacenar el historial de operaciones, sino que también el de gráficas mediante **RealtimeSetGraficas** Que cambia los valores con Numpy, almacena datos de gráficas (pares de valores x , y) en Firebase junto con la expresión que los generó.

```
def RealtimeSetGraficas(expresion, x_val_fb, y_val_fb):
    if not user_uid or not correo:
        raise ValueError("Debe iniciar sesión primero.")

    # Sanitiza el correo electrónico antes de usarlo en el path
    sanitized_email = sanitize_path(correo)

    # Usa el UID y el correo electrónico sanitizado para crear una clave única en Firebase
    ref = db.reference(f'usuarios/{sanitized_email}')
    nueva_entrada = ref.push() # Crea una nueva entrada con una clave única

    # Convertir x_val_fb y y_val_fb a listas si son arrays de NumPy
    x_val_list = x_val_fb.tolist() if isinstance(x_val_fb, np.ndarray) else x_val_fb
    y_val_list = y_val_fb.tolist() if isinstance(y_val_fb, np.ndarray) else y_val_fb

    # Crear una lista de pares x:y como cadenas
    datos_graficas = [f" {x} : {y} " for x, y in zip(x_val_list, y_val_list)]

    nueva_entrada.set({
        'expresion': expresion,
        'valores x : f(x)': datos_graficas
    })

    print("Datos ingresados correctamente en el historial")
```

8.2. Inicio de sesión y registro

El primer archivo que se llama en el main para ejecutar el código es `usuarios.py` donde el usuario puede registrarse y/o iniciar sesión en la base de datos de Firebase mediante un correo electrónico y su respectiva contraseña; luego, si estos datos son correctos se abrirá la ventana de la calculadora `vista.py`.

El bloque encargado de iniciar sesión, `iniciar_sesion` permite reconocer si es un correo válido con “`auth.get_user_by_email(email)`”, y así mismo, comprobar si la contraseña ingresada es correcta y correspondiente con el correo registrado. De ser correcto, abre el archivo `VIEW_vista.py` donde se encuentra la calculadora y adjudica uid al correo, si es incorrecto muestra ventana de error.

```
def iniciar_sesion(email, contraseña):
    try:
        user = auth.get_user_by_email(email)
        if user:
            messagebox.showinfo("Éxito", "Sesión iniciada correctamente.")
            ventana_sesion.destroy()
            uid = user.uid
            import VIEW_vista
            firebase_python.configurar_usuario(uid, email)
        else:
            messagebox.showerror("Error", "Usuario no encontrado.")
    except firebase_admin.auth.UserNotFoundError:
        messagebox.showerror("Error", "Usuario no encontrado.")
    except Exception as e:
        messagebox.showerror("Error", f"Ocurrió un error: {str(e)}")
```

El bloque de registro en la base de datos firebase, `registrar_usuario` válida si la entrada de aquel correo y contraseña es posible. En caso de ser

```
def registrar_usuario(email, contraseña):
    if not email.strip() or not contraseña.strip():
        messagebox.showerror("Error", "Debe ingresar un correo electrónico y una contraseña.")
        return

    try:
        user = auth.create_user(email=email, password=contraseña)
        messagebox.showinfo("Éxito", f"Usuario registrado correctamente con UID: {user.uid}")
    except firebase_admin.auth.EmailAlreadyExistsError:
        messagebox.showerror("Error", "El correo electrónico ya está registrado.")
    except Exception as e:
        messagebox.showerror("Error", f"Ocurrió un error al registrar el usuario: {str(e)}")
```




correcta, “auth.create_user(email=email, password=contraseña)” se encarga de crear la cuenta usuario en firebase; si es incorrecto, muestra si el error es porque la cuenta ya existe o es otro tipo de error.

Calculadora grafica ▾

Authentication

Usuarios Método de acceso Plantillas Uso Configuración Extensions

El acceso con redireccionamiento de origen cruzado en Google Chrome M115 y versiones posteriores ya no es compatible y dejará de funcionar el 24 de junio de 2024.

Buscar por dirección de correo electrónico, número de teléfono o UID de usuario

Agregar usuario

Identificador	Proveedores	Fecha de creación ▾	Fecha de acceso	UID de usuario
correo2.prueba@pagin...	✉	26 ago 2024		iiidly9Wh3OWgCuj78uYHuX6...
correo1.prueba@pagin...	✉	26 ago 2024		IG2tJ8iurVWsDb1Hh3esVUKy...

Filas por página: 50 ▾ 1 - 2 of 2 < >

Realtime Database

[Datos](#) | [Reglas](#) | [Copias de seguridad](#) | [Uso](#) | [Extensions](#)

<https://calculadora-grafica-468ee-default-rtdb.firebaseio.com>

<https://calculadora-grafica-468ee-default-rtdb.firebaseio.com/>

```

└─ usuarios
  └─ correo1_prueba_pagina_com
    └─ -05Gg-WmG_VD0Wl1wMy0
      ├── expresion: "7math.pi"
      └── resultado: "21.991148575128552"
    └─ -05Gg3pDC_R9e88BA175
    └─ -05Gg8njGjxpGbrC6GYn
    └─ -05GiawZ0sdFnChC9kvg
    └─ -05Gv6ganJrHfedjIMw4
    └─ -05Gv9xSEXH_qcc75wcy
    └─ -05GwFMmZdv0ErmTXyFa
  
```

Por último, se crea todo el entorno visual de la ventana, títulos, barras string, y los botones Iniciar Sesión y Registrarse.

```

# Interfaz Tkinter
ventana_sesion = tk.Tk()
ventana_sesion.title("Iniciar Sesión")
ventana_sesion.geometry("300x250")

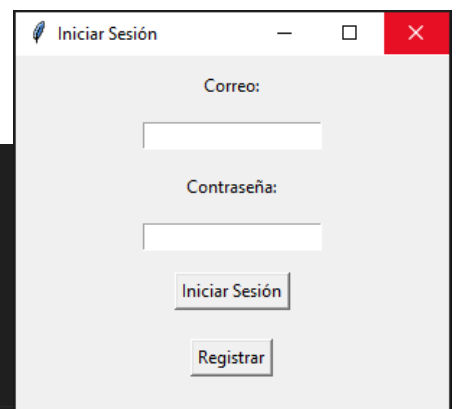
tk.Label(ventana_sesion, text="Correo:").pack(pady=10)
email_entry = tk.Entry(ventana_sesion)
email_entry.pack(pady=5)

tk.Label(ventana_sesion, text="Contraseña:").pack(pady=10)
contraseña_entry = tk.Entry(ventana_sesion, show="*")
contraseña_entry.pack(pady=5)

tk.Button(ventana_sesion, text="Iniciar Sesión", command=lambda: iniciar_sesion(email_entry.get(), contraseña_entry.get())).pack(pady=10)
tk.Button(ventana_sesion, text="Registrarse", command=lambda: registrar_usuario(email_entry.get(), contraseña_entry.get())).pack(pady=10)

ventana_sesion.mainloop()

```



8.3. Mostrar la calculadora

Se abre el archivo vista.py que contiene todo el GUI hecho con Tkinter. Se usó el estilo 'clam' de ttk que permite usar widgets más modernos.

```
#VENTANA CALCULADORA
root=Tk()
root.title("Calculadora")
root.geometry("+200+80")
root.resizable(False,False)
root.columnconfigure(0,weight=1)
root.rowconfigure(0,weight=1)
```

```
estilos=ttk.Style()
estilos.theme_use('clam')
```

Creamos la ventana principal con Tk(), le damos un título, posición (tamaño) y damos la opción de weight=1 para más adelante colocar los botones como filas y columnas y que se vaya acomodando el tamaño.

Se define el estilo para el primer Label, lo creamos y lo asociamos como una entrada de texto con entrada1, es decir que cualquier cambio en esta entrada se verá reflejado en el Label y con entrada1 = StringVar() hacemos que se actualice automáticamente (Utilizan variables de tipo StringVar para actualizar dinámicamente el texto de las etiquetas según la interacción del usuario).

Se definen dos etiquetas para mostrar datos. Label_entrada1 está dentro del mainframe y Label_entrada2 en un frame separado.

```
entrada1=StringVar()
Label_entrada1=ttk.Label(mainframe,textvariable=entrada1,style='Label1.TLabel')
Label_entrada1.grid(column=0, row=0,columnspan=4,sticky=(W,E,N,S))

#LABEL 2
estilos_label2=ttk.Style()

entrada2 = StringVar()
Label_entrada2 = ttk.Label(frame, textvariable=entrada2, style='Label2.TLabel', font="arial 20")
Label_entrada2.grid(column=0, row=0, sticky=(N, S, W, E), padx=25, pady=0)
```

A partir de aquí se crean los botones con sus respectivos estilos, posiciones y las funciones del archivo entradas.py que se encargará de evaluar las operaciones ingresadas por el usuario. Aquí se muestra el ejemplo de algunos botones:

```
#potencias, raices, logaritmo
button_potencia_dos = ttk.Button(mainframe, text="x²", style='Botones_restantes.TButton', command=lambda: MV_entradas.ingresar('x²', entrada1))
button_potencia_y = ttk.Button(mainframe, text="x^y", style='Botones_restantes.TButton', command=lambda: MV_entradas.ingresar('^', entrada1))
button_raiz_cuadrada = ttk.Button(mainframe, text="√", style='Botones_restantes.TButton', command=lambda: MV_entradas.ingresar('√', entrada1))
button_raiz = ttk.Button(mainframe, text="x√n", style='Botones_restantes.TButton', command=lambda: MV_entradas.ingresar('x√n', entrada1))
button_logaritmo = ttk.Button(mainframe, text="log", style='Botones_restantes.TButton', command=lambda: MV_entradas.ingresar('log', entrada1))
```

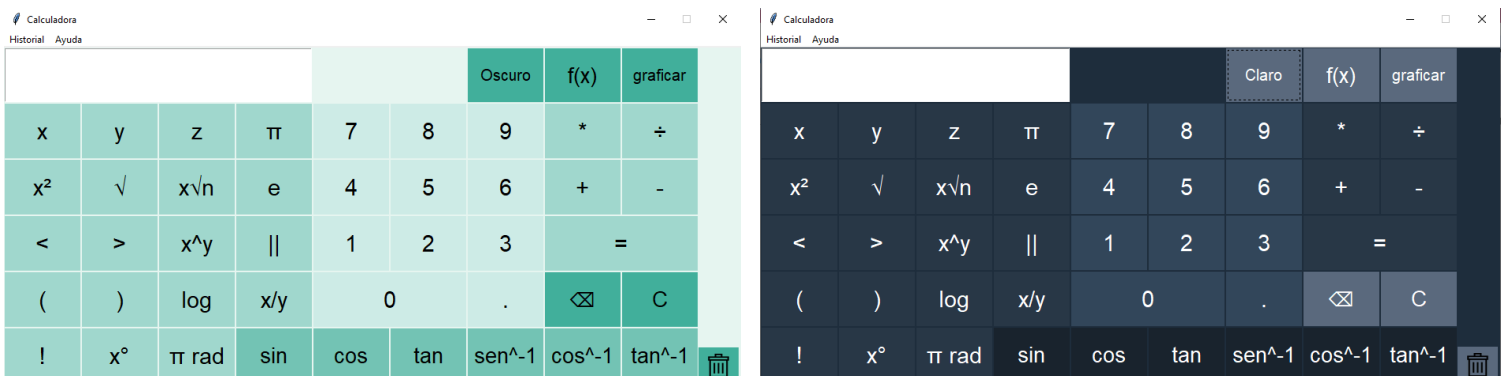
El usuario podrá alternar entre modo oscuro y modo claro con un botón:

```
def claro():
    estilos.configure('mainframe.TFrame', background="#e8f6f3", font="arial 22")
    estilos_label1.configure('Label1.TLabel', font="arial 25", anchor="e", background="#e8f6f3")
    estilos_label2.configure('Label2.TLabel', font="arial 15", anchor="e", background="#e8f6f3", foreground="#0b5345")

    estilos_botones_numeros.configure('Botones_numeros.TButton', font="arial 22", width="5", background="#d0ece7", relief="flat", foreground="#000000")
    estilos_botones_numeros.map('Botones_numeros.TButton', foreground=[('active', '#17202a')], background=[('active', '#a2d9ce')])
```

```
def oscuro():
    estilos.configure('mainframe.TFrame', background="#212f3c", font="arial 22")
    estilos_label1.configure('Label1.TLabel', font="arial 25", anchor="e", background="#d6dbdf")
    estilos_label2.configure('Label2.TLabel', font="arial 15", anchor="e", background="#212f3c", foreground="#d6dbdf")

    estilos_botones_numeros.configure('Botones_numeros.TButton', font="arial 22", width="5", background="#34495e", relief="flat", foreground="FFFFFF")
    estilos_botones_numeros.map('Botones_numeros.TButton', foreground=[('active', '#17202a')], background=[('active', '#aeb6bf')])
```



8.4. Realizar operaciones

La primera función del archivo entradas.py se encarga de actualizar el historial para ser mostrado en pantalla. La segunda es ingresar, que obtiene los símbolos de los botones digitados por el usuario, hace una serie de modificaciones (por ejemplo insertar el símbolo de multiplicar) para que la función eval los comprenda y retorne el resultado de la operación ingresada mostrándola en pantalla, en el historial local, y guardándola en firebase.

```
def ingresar(tecla, entrada1):
    current_text = entrada1.get()
    op1 = ['+', '-', '*', '/', '^', '(', ')', '.', '!', '<', '>']
    op2 = ['sin', 'cos', 'tan', 'selev-1', 'conoelev-1', 'taelev-1']
    op3 = ['√', 'x²', 'x√n', '||', 'x/y', '±', 'f(x)=', 'x°', 'P rad']

    if tecla.isdigit() or tecla in 'xyzne' or tecla in op1:
        entrada1.set(current_text + tecla)
    elif tecla in op2:
        entrada1.set(tecla + '(' + current_text + ')')
    elif tecla in op3:
        if tecla == '√':
            entrada1.set(tecla + current_text)
        elif tecla == 'x√n':
            entrada1.set(current_text + '√')
        elif tecla == 'x²':
            entrada1.set('(' + current_text + ')^2')
        elif tecla == '||':
            entrada1.set('abs(' + current_text + ')')
        elif tecla == '±':
            entrada1.set(tecla + current_text)
        elif tecla == 'f(x)=':
            entrada1.set(tecla + current_text)
        elif tecla == 'x°': #Radianes a grados
            entrada1.set(math.degrees('(' + entrada1.get() + ')')')
        elif tecla == 'P rad': #Grados a radianes
            entrada1.set('math.radians('(' + entrada1.get() + ')')')

    elif tecla == '=' or tecla == 'Return':
        expresion = current_text
        expresion = expresion.replace('^', '**')
        expresion = expresion.replace('π', 'math.pi')
        expresion = expresion.replace('sin', 'math.sin')
        expresion = expresion.replace('cos', 'math.cos')
        expresion = expresion.replace('tan', 'math.tan')

        expresion = expresion.replace('selev-1', 'math.asin')
        expresion = expresion.replace('conoelev-1', 'math.acos')
        expresion = expresion.replace('taelev-1', 'math.atan')

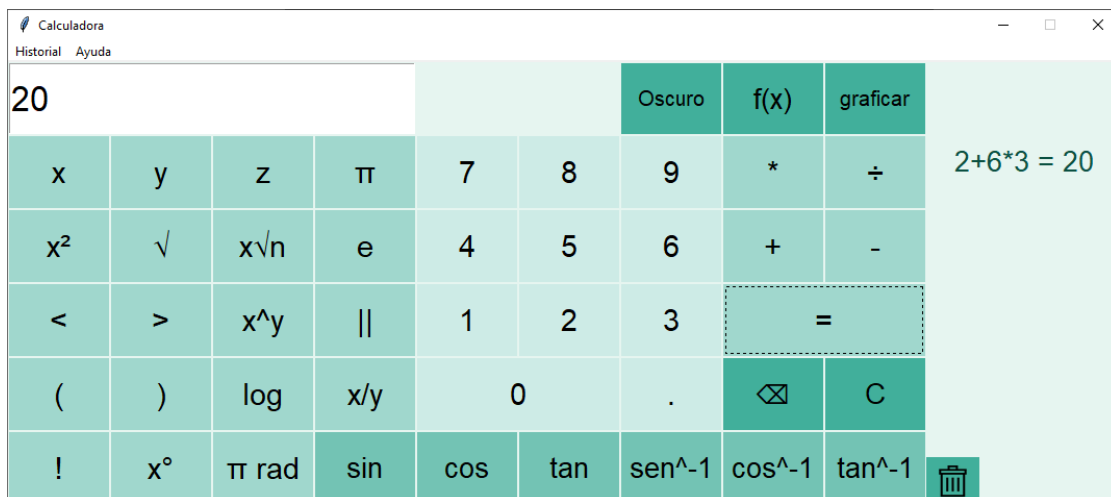
        expresion = expresion.replace('log', 'math.log')
        expresion = expresion.replace('!', 'math.factorial')
        expresion = expresion.replace('√', 'math.sqrt')

        nueva_expresion = vgf.insertar_multiplicacion(expresion)

        try:
            resultado = eval(nueva_expresion)
            entrada1.set(str(resultado))
            firebase_python.RealtimeSet([expresion, str(resultado)])
            nuevo_texto = f"{nueva_expresion} = {resultado}"
            actualizar_historial_pantalla(vista.entrada2, nuevo_texto)
        except SyntaxError:
            entrada1.set("Error")
            messagebox.showerror("Error de Sintaxis", "Usted ingresó una operación o un ca")
        except ZeroDivisionError:
            entrada1.set("Error")
            messagebox.showerror("Error Matemático", "No se puede dividir por cero. Por f")
        except Exception as e:
            entrada1.set("Error")
            messagebox.showerror("Error", f"Se produjo un error inesperado: {str(e)}. Vuel")

    else:
        firebase_python.RealtimeSet(current_text, '')
        firebase_python.RealtimeUpdate(current_text, '', vista.entrada2)
```

Finalmente, se definen las funciones para los botones de borrar, borrar todo y limpiar la pantalla del historial. Al borrar eliminamos el último elemento que hay en entrada 1 y en borrar todo simplemente dejamos una cadena vacía.



8.5. Graficar en 2d y 3d

El archivo `v_grafica_funcionamiento` posee dos funciones importantes, la primera `insertar_multiplicacion`, para agregar el símbolo `*` donde sea necesario.

En la segunda evaluaremos la función dando uso de funciones permitidas de `VIEW_ventana_grafica`, añadimos excepciones para mensajes de error; se usa `eval` y se retorna el resultado para graficar.

8.5.1 VIEW_ventana_grafica

El archivo `ventana_grafica.py` está diseñado para crear gráficos 2D de funciones matemáticas utilizando la librería `matplotlib` en Python.

Importamos librerías, y los archivos:
`MV_v_grafica_funcionamiento` y
`Model_firebase_python` (que nos sirve para subir la función y los datos de la gráfica a la base de datos de firebase).

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Button
import math
from tkinter import messagebox, simpledialog, colorchooser

import MV_v_grafica_funcionamiento as v_grafica_funcionamiento
import MODEL_firebase_python as firebase_python
```

Hay un problema, y es que no se reconoce las expresiones, por ejemplo, “28y” como 28*y. Para evitar que el ingreso acostumbrado de datos matemáticos por parte del usuario dé error, se deberá cambiar aquella expresión.

Por ello, crearemos una función de nombre: “`insertar_multiplicacion`”, mediante la cual detecta y añade el operador de multiplicación donde sea necesario, esto para evitar errores de sintaxis y que la librería no reconozca la expresión. Ejemplo, convierte “5x” en “5*x”.

```
def insertar_multiplicacion(expresion):
    nueva_expresion = ""
    longitud = len(expresion)
    funciones = {"sin", "cos", "tan", "log", "sqrt", "selev-1", "conoelev-1", "taelev-1"}

    i = 0
    while i < longitud:
        # Detectar funciones y omitir multiplicación automática
        for funcion in funciones:
            if expresion[i:i+len(funcion)] == funcion:
                nueva_expresion += funcion
                i += len(funcion)
            # Verificar si después de la función hay un paréntesis, agregarlo sin añadir un *
            if i < longitud and expresion[i] == '(':
                nueva_expresion += expresion[i]
                i += 1
            break
        else:
            # Añadir multiplicación donde sea necesario
            if i > 0 and (
                (expresion[i].isdigit() and (expresion[i-1].isalpha() or expresion[i-1] == '(')) or
                (expresion[i].isalpha() and (expresion[i-1].isdigit() or expresion[i-1] == '(')) or
                (expresion[i] == '(' and (expresion[i-1].isdigit() or expresion[i-1].isalpha()))
            ):
                nueva_expresion += '*'

            nueva_expresion += expresion[i]
            i += 1

    return nueva_expresion
```

```
41 def abrir_ventana(entrada1):
42     # Obtener la expresión de la entrada
43     expresion = entrada1.get()
44     expresion = expresion.replace('^', '**')
45     expresion = expresion.replace('f(x)=', '')
46
47     funciones_permitidas = {
48         'sin': math.sin,
49         'cos': math.cos,
50         'tan': math.tan,
51         'selev-1': math.asin,
52         'conoelev-1': math.acos,
53         'taelev-1': math.atan,
54         'log': math.log,
55         'sqrt': math.sqrt,
56         'pi': math.pi,
57         'e': math.e
58     }
59
```

“abrir_ventana” es la función principal que maneja la lógica de entrada y salida, incluyendo la evaluación de la expresión matemática introducida por el usuario, la generación de valores para graficar, el almacenamiento en Firebase, y la visualización de la gráfica.

```
60     # Insertar multiplicaciones en la expresión
61     expresion = v_grafica_funcionamiento.insertar_multiplicacion(expresion)
62
63     # Obtener el rango de los ejes del usuario
64     rangos = simpledialog.askstring(
65         "Rango de Ejes",
66         "Ingrese el rango para los ejes x en el formato 'x_min, x_max:',",
67         initialvalue="-20, 20"
68     )
69
70     if rangos:
71         try:
72             x_min, x_max = map(float, rangos.split(','))
73         except ValueError:
74             messagebox.showerror("Error", "El formato ingresado es incorrecto. Por favor ingrese valores válidos.")
75             return
76     else:
77         return # Si el usuario cancela el diálogo
78
79     # Generar valores de x y y
80     x_vals = np.linspace(x_min, x_max, 1000)
```

Obtendremos la función ingresada por el usuario desde el widget de: “entrada1”, reemplazamos el símbolo “^” para que el programa comprenda la operación y eliminamos “f(x)=” para no generar errores.

“funciones_permitidas” es un diccionario que mapea nombres de funciones a las funciones reales de Python (por ejemplo, 'sin': math.sin). Esto asegura que solo se puedan usar funciones seguras y esperadas en la evaluación.

‘expresion = v_grafica_funcionamiento.insertar_multiplicacion(expresion)’: Llama a una función para añadir operadores de multiplicación implícitos en la expresión, si es necesario.

Se llaman las funciones del archivo anterior y se le pregunta al usuario por los rangos de x en los que desea graficar la función y a partir de la respuesta se generan los valores para el eje y.

Para poner aquella expresión en la gráfica, necesitamos un rango de valores en los que se pueda visualizar. Este se hará consultando al usuario si piensa cambiar los valores por defecto, “-20, 20”.

```

for x in x_vals:
    y = v_grafica_funcionamiento.evaluar_funcion(x, expresion, funciones_permitidas)
    if y is None:
        return
    y_vals.append(y)
# Crear la figura y el eje
fig, ax = plt.subplots()

# Graficar los datos
linea, = ax.plot(x_vals, y_vals, color='blue', label=str(expresion))
ax.set_xlabel('Eje x')
ax.set_ylabel('Eje f(x)')
ax.set_title('Gráfica de ' + str(expresion))
ax.grid(True)
ax.legend()
ax.set_ylim(-5, 5)
ax.axhline(0, color='blue', linestyle='--')
ax.axvline(0, color='blue', linestyle='--')

# Función para cambiar el color de la gráfica
def cambiar_color(event):
    nuevo_color = colorchooser.askcolor(title="Seleccione el Nuevo Color de la Línea")[1]
    if nuevo_color:
        linea.set_color(nuevo_color)
        fig.canvas.draw()

ax_color = plt.axes([0.8, 0.01, 0.1, 0.05]) # Posición del botón en la figura
boton_color = Button(ax_color, 'Color')
boton_color.on_clicked(cambiar_color)

mostrar_3d = messagebox.askyesno('Graficar 3D', '¿Desea mostrar la gráfica en 3D?')

if mostrar_3d: # Esto es lo mismo que decir `if mostrar_3d == True:`
    v_grafica_funcionamiento.grafica_3D(expresion, x_min, x_max, color='blue', resolution=100)
for x in x_val_fb:
    y = v_grafica_funcionamiento.evaluar_funcion(x, expresion, funciones_permitidas)
    if y is None:
        return # Si hay un error, salir de la función
    y_val_fb.append(y)

firebase_python.RealtimeSetGraficas(expresion, x_val_fb, y_val_fb)
print("x_vals shape:", x_vals.shape)
print("y_vals shape:", np.array(y_vals).shape)

```

Con los valores consultados, se adjudican el rango de valores a la gráfica:

`x_vals = np.linspace(x_min, x_max, 1000)`: Genera un rango de 1000 valores de x entre `x_min` y `x_max`.

Se evalúa la expresión para cada valor de x utilizando `v_grafica_funcionamiento.evaluar_funcion` y se almacena el resultado en `y_vals`.

El usuario tiene la opción de escoger los colores de la línea de la gráfica con el comando `colorchooser.askcolor`

Ahora hace falta poner todos aquellos datos recolectados, traducidos y seleccionados en una ventana que muestre la gráfica. Graficamos los valores de y en x, titulamos los ejes y nombre de la gráfica, pondremos cuadrícula y color.

Se ajustan los valores de y , para que puedan ser graficados de manera correcta y no haya errores cuando se generen valores infinitos. Creamos un botón para que el usuario pueda cambiar el color en el que se muestra la gráfica (esta opción también está en la ventana que grafica en 3D).

Preguntamos al usuario si desea graficar la función en 3D (en ese caso la llamamos del archivo `v_grafica_funcionamiento`) y finalmente mostramos las ventanas y guardamos los datos generados en firebase.

8.5.2 `v_grafica_funcionamiento`

Está diseñado para crear gráficos 3D de funciones matemáticas utilizando la librería `matplotlib` en Python. El script incluye funciones para manipular expresiones matemáticas, evaluar funciones en puntos específicos y graficar estas funciones en un entorno 3D.

Importamos librerías: `Numpy`, `matplotlib.pyplot`, `tkinter.messagebox`, `mpl_toolkits.mplot3d.Axes3D`.

Insertamos también la función de “`insertar_multiplicacion`”.

Evaluaremos la función dando uso de `funciones_permitidas` de `VIEW_ventana_grafica`, añadimos excepciones para mensajes de error.

```
def evaluar_funcion(x, expresion, funciones_permitidas):
    try:
        resultado = eval(expresion, {'__builtins__': None}, {'funciones_permitidas': x, 'y': y, 'z': z})
        return resultado
    except SyntaxError:
        messagebox.showerror("Error de Sintaxis", "Usted ingresó una operación que no se puede resolver, falta o sobra un caracter. Por favor, revise la expresión y vuelva a intentarlo.")
    except ZeroDivisionError:
        messagebox.showerror("Error Matemático", "No se puede dividir por cero. Por favor, corrija la operación.")
    except Exception as e:
        messagebox.showerror("Error", f"Se produjo un error inesperado: {str(e)}. Vuelva a ingresar la operación correctamente.")
```

Los colores se pueden escoger con la función `cambiar_color`.

```
def cambiar_color(event, ax, x_vals, y_vals, z_vals, fig):
    nuevo_color = colorchooser.askcolor(title="Seleccione el Nuevo Color de la Línea")[1]
    if nuevo_color:
        ax.clear() # Limpiar el eje actual
        ax.plot_surface(x_vals, y_vals, z_vals, color=nuevo_color)
        ax.set_xlabel('Eje x')
        ax.set_ylabel('Eje y')
        ax.set_zlabel('Eje z')
        ax.set_title('Gráfica 3D')
        ax.set_zlim(-5, 5)
        fig.canvas.draw_idle()
```

El usuario puede escoger entre una gráfica 2d y una 3d. Para ello hacemos:

- Confirmación del Usuario: Utiliza un cuadro de mensaje para preguntar al usuario si desea proceder con la gráfica 3D.
- Generación de Valores: Utiliza numpy para crear una malla de valores x e y, y luego evalúa la expresión para obtener los valores z correspondientes.
- Filtrado de Valores: Se filtran valores extremos de z (por ejemplo, más allá de 5 o por debajo de -5) para evitar gráficos no deseados.
- Configuración de la Gráfica: Configura la figura de matplotlib para un gráfico 3D, define los ejes y el rango de z, y finalmente muestra la gráfica.

```
def grafica_3D(expresion, x_min, x_max, color, resolution=200):
    x_vals = np.linspace(x_min, x_max, resolution)
    y_vals = np.linspace(x_min, x_max, resolution)
    x_vals, y_vals = np.meshgrid(x_vals, y_vals)

    # Evaluar la función para z, manejando las discontinuidades
    z_vals = np.array([
        evaluar_funcion(x, expresion, funciones_permitidas={'sin': np.sin, 'cos': np.cos, 'tan': np.tan, 'log': np.log, 'sqrt': np.sqrt, 'pi': np.pi, 'e': np.e})
        for x, y in zip(np.ravel(x_vals), np.ravel(y_vals))
    ]).reshape(x_vals.shape)

    # Filtrar valores que excedan los límites para evitar gráficos no deseados
    z_vals = np.where(np.abs(z_vals) > 5, np.nan, z_vals)

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

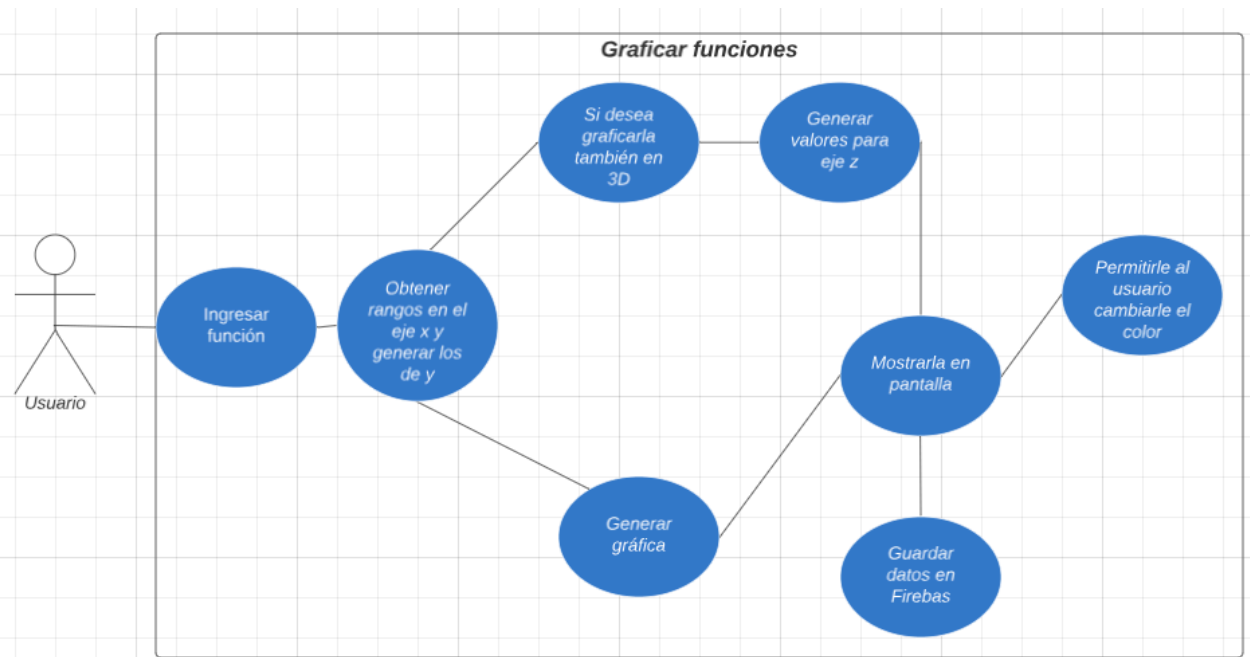
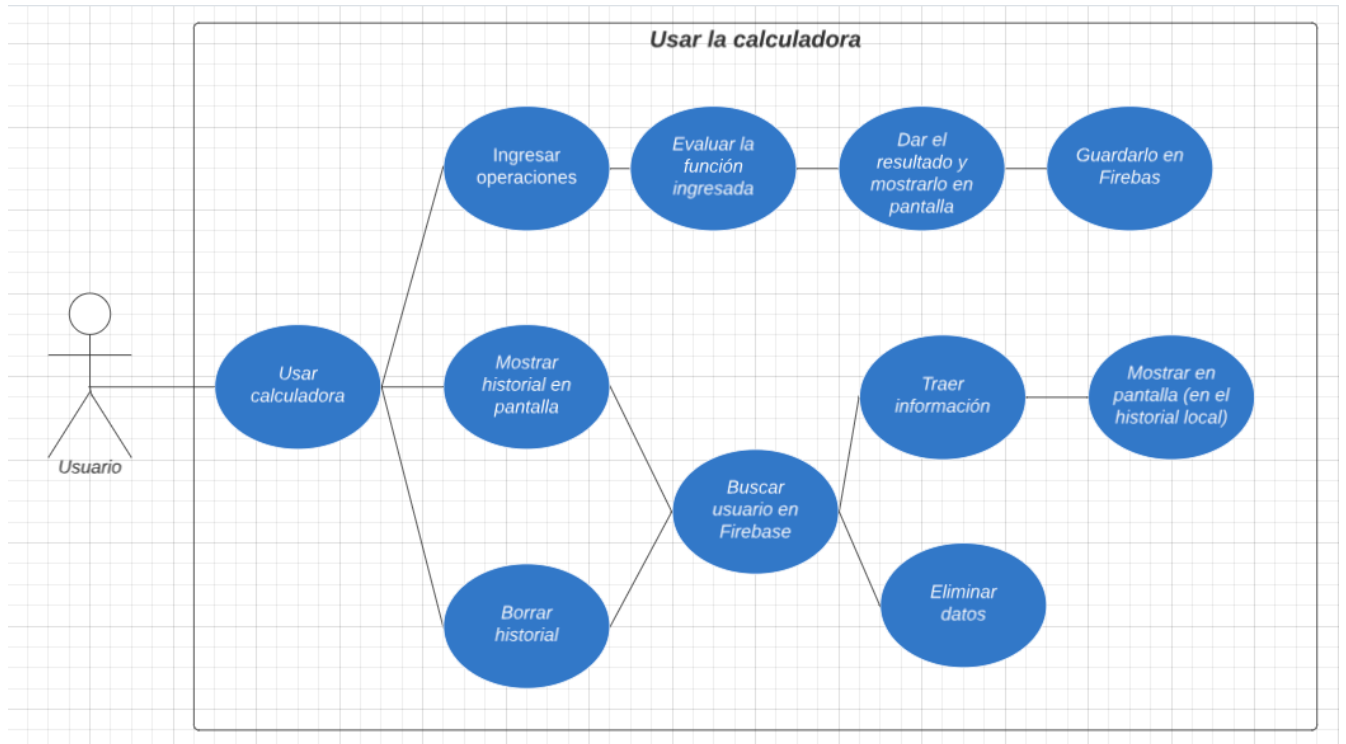
    # Graficar la superficie
    ax.plot_surface(x_vals, y_vals, z_vals, color=color)
    ax.set_xlabel('Eje x')
    ax.set_ylabel('Eje y')
    ax.set_zlabel('Eje z')
    ax.set_title('Gráfica 3D de ' + str(expresion))
    ax.set_zlim(-5, 5)

    # Crear un botón para cambiar el color usando matplotlib.widgets.Button
    ax_color = plt.axes([0.8, 0.01, 0.1, 0.05])
    boton_color = Button(ax_color, 'Color')
    boton_color.on_clicked(lambda event: cambiar_color(event, ax, x_vals, y_vals, z_vals, fig))

    plt.show()
```

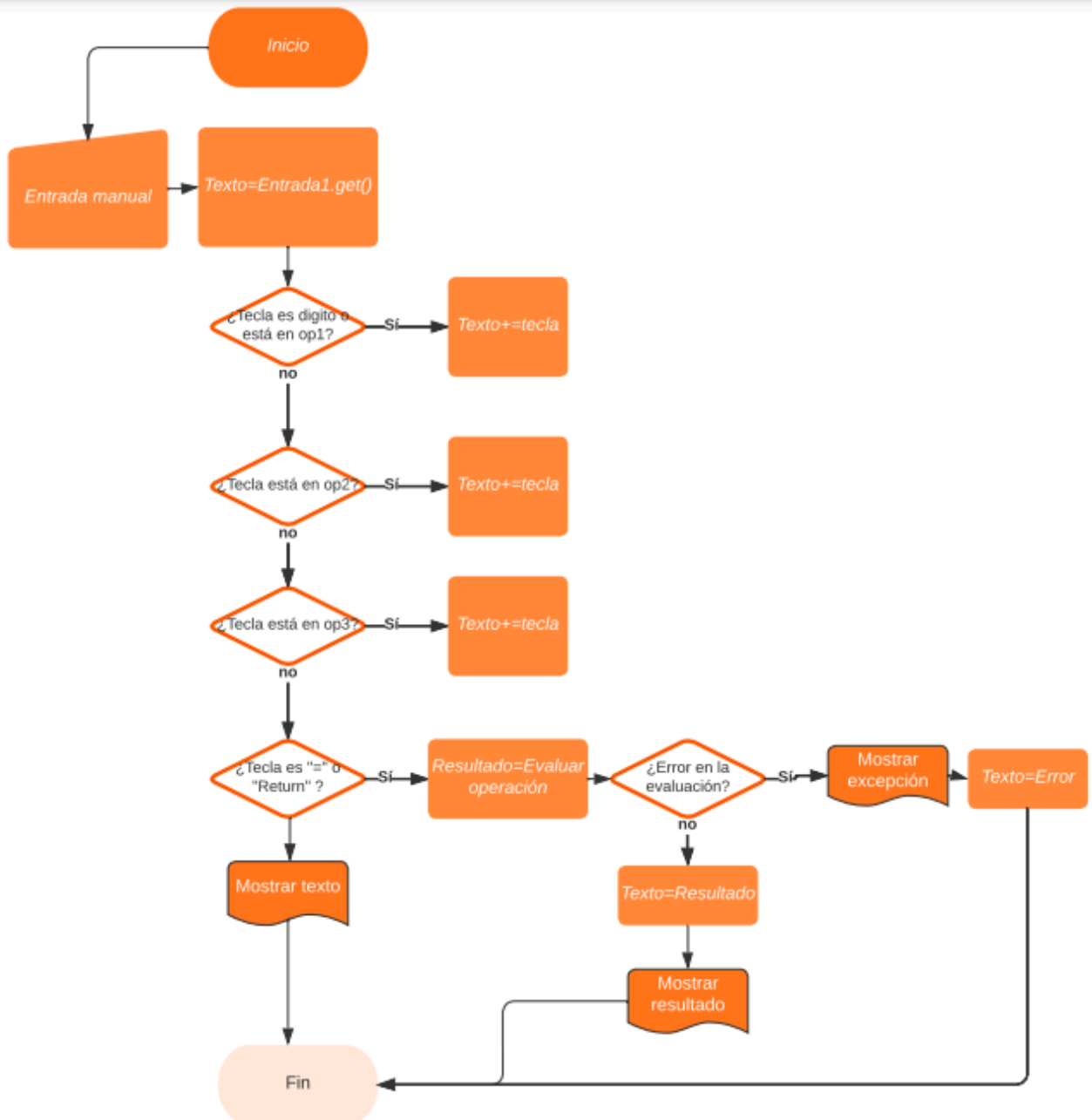
9. Diagramas de casos de uso



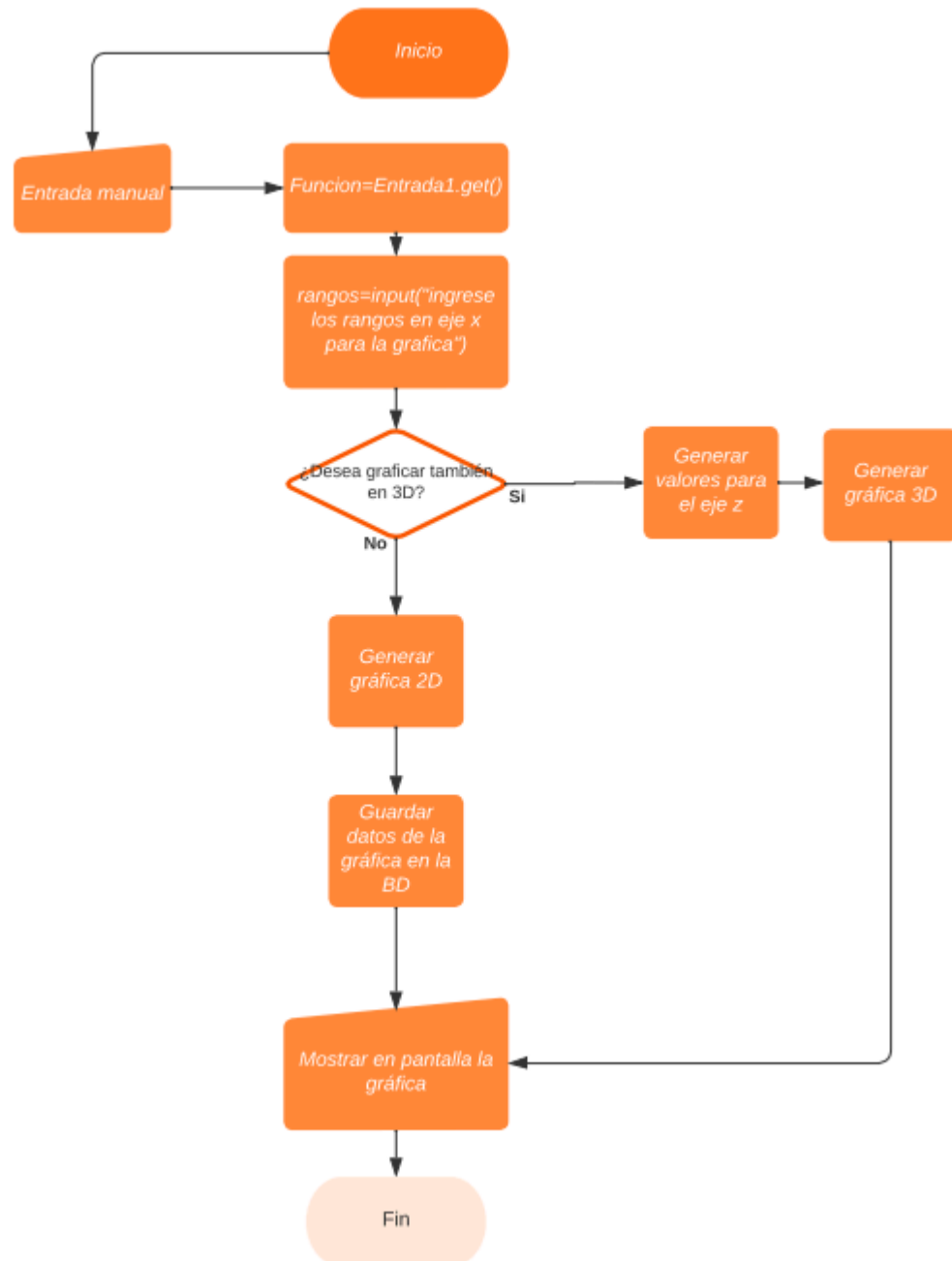


10. Diagramas de flujo de funciones

10.1. Función ingresar()



10.2. Función de graficar



11. Excepciones

Todos tienen en común el error de sintaxis (“Digite nuevamente la información”):

CASO DE USO	EXCEPCIONES
INICIO DE SESIÓN	El usuario no se encuentra en la base de datos.
REGISTRO	El correo electrónico ya se encuentra registrado.
INGRESO DE OPERACIONES	ZeroDivision: No es posible dividir entre cero. La expresión no es lógica, se produjo un error, vuelva a ingresar correctamente la operación.
BORRAR DATOS DE LA BD	Si ocurre algún error de conexión o datos con Firebase.
GRAFICAR FUNCIONES	ZeroDivision: No es posible dividir entre cero. La expresión no es lógica, se produjo un error, vuelva a ingresar correctamente la operación. Al ingresar los rangos en x: El formato ingresado es incorrecto. Por favor ingrese valores válidos.

12. Información a tener en cuenta

- La calculadora sólo grafica funciones de una variable (x), si el usuario digita $x+y$, se entenderá como $x+x$.
- Para hacer uso de la base de datos será necesario contar con Internet para guardar la información.
- Siempre se graficará sólo una función por ventana generada por matplotlib.

13. Conclusiones

El proyecto permitió el desarrollo de una calculadora gráfica que no solo evalúa y representa gráficamente funciones matemáticas en 2D y 3D, sino que también ofrece la capacidad de gestionar temas (claro y oscuro) y registrar el historial de operaciones.

Se ha integrado Firebase para guardar y recuperar datos, lo que añade una capa de persistencia a la aplicación, permitiendo a los usuarios acceder a su historial de funciones graficadas desde cualquier lugar.

Además, se integran excepciones para los casos de uso y se aprovecha como una oportunidad para aplicar conocimientos en programación con el lenguaje Python aprendidos a lo largo del curso, implementando librerías, funciones e interfaz gráfica.