Informe de Funcionamiento del Programa de Calculadora Gráfica

Integrantes:
Ivan Camilo Idrobo Rojas
Sebastian Steve Espitia Puentes
Cristian Daniel Silva Hernandez
Joan Santiago Colmenares Santamaria

Índice:

Como Instalar

Se intuye que el usuario ya tiene descargado el lenguaje de programación de Python y este fue añadido a PATH.

Para instalar el programa se debe de primero descargar las librerías de tkinter, firebase, pyrebase, matplotlib y numpy:

Instalación Librerías

Para instalar las librerías mencionadas, se debe de ejecutar el CMD como administrador para que se pueda lograr de manera correcta las instalaciones, a continuación se mostraran las líneas de comando necesarias para descargar las librerías de manera correcta.

-Tkinter:

```
pip install tk
```

-firebase:

```
pip install firebase_admin
```

-pyrebase:

```
pip install pyrebase4
```

-matplotlib:

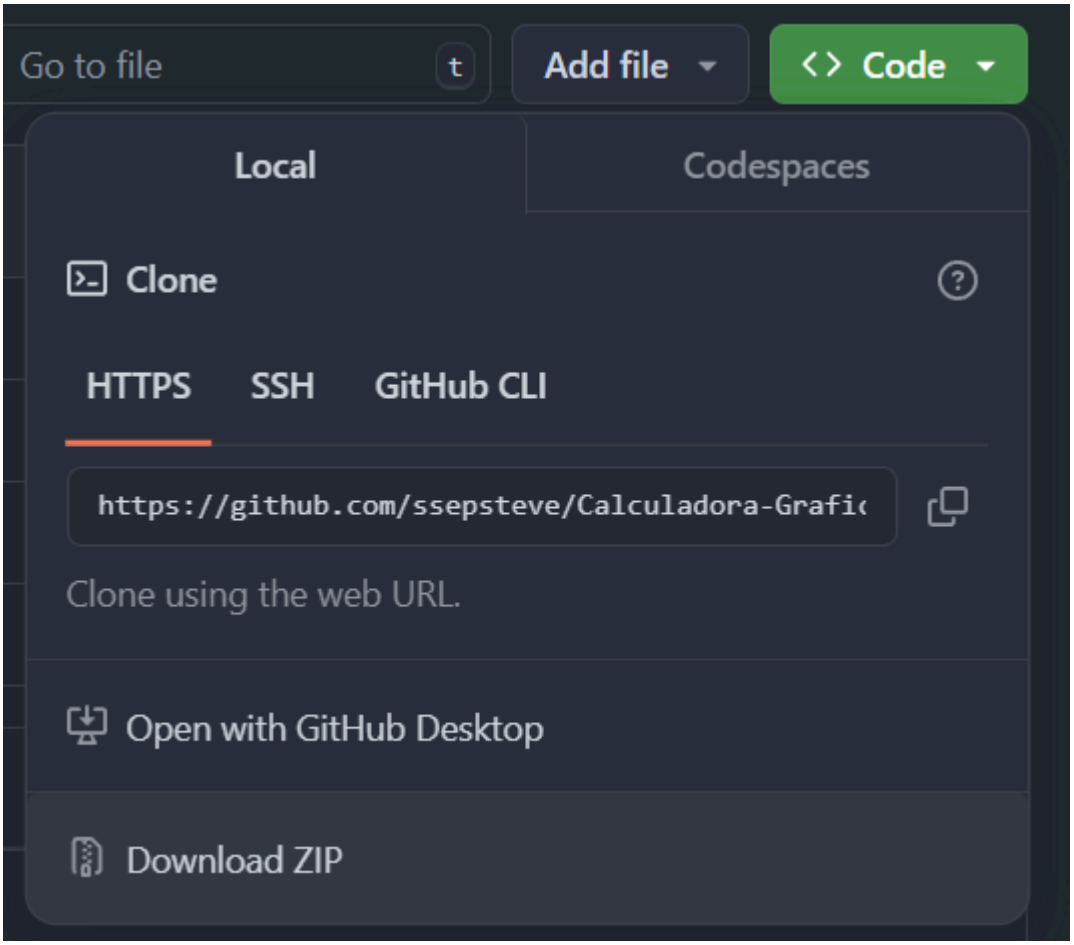
```
pip install matplotlib
```

-numpy:

```
pip install numpy
```

Instalación Programa

Para instalar el programa se debe de acceder al siguiente github:
https://github.com/ssepsteve/Calculadora-Grafica-PC-Grupo-2.git
Luego se debe de descargar el comprimido del programa de la siguiente manera:



Cuando se haya descargado el comprimido ZIP, se debe de luego mover a donde desee el usuario tener el programa.
Cuando ya se haya ubicado y descomprimido en el programa deseado, se debe de hacer modificaciones al código:

Modificación Rutas De Acceso

Para que él programe funcione de manera correcta, se debe de modificar los archivos, GUIBotones, GUIinicioSesion, Controller_Model y connector, ya que al mantener la estructura en carpetas de MVC, se debe de especificar la ruta de acceso de cada carpeta en donde se encuentra cada archivo, por lo que se deben de cambiar las rutas de acceso con respecto a donde ubico el archivo el programa:

GUIBotones:

```
import tkinter.messagebox
import sys
sys.path.insert(1,"C:/Ruta/A/La/Carpeta:/2controller")
import operaciones
import tkinter
```

GUIinicioSesion:

```
import sys
import tkinter.messagebox
sys.path.insert(1,"C:/Ruta/A/La/Carpeta:/2controller")
import controller_Model as com
import tkinter
from tkinter import ttk
```

Controller_Model:

```
import sys
sys.path.insert(1,"C:/Ruta/A/La/Carpeta:/1Model")
import funcionesDB
```

Connector

Dentro de connector se debe de modificar la línea 9 de código en donde se debe de cambiar la ruta de acceso por la ruta correspondiente al archivo JSON de la carpeta model.

```
8   def fireBaseInitializeCreds():
9       cred = credentials.Certificate("C:/Ruta/A/El/Archivo:/certificates.json")
10      firebase_admin.initialize_app(cred, {
```

Ejecución Programa:

Finalmente, tras seguir los pasos anteriores se debe de ejecutar el programa al ejecutar el archivo GUITkinter, ya que este funciona como el programa de inicialización como tal.

1. Estructura del Programa: Dependencias y Funcionalidades por Carpetas

1.1. Librerías utilizadas:

- NumPy: Sirve para crear vectores y/o matrices multidimensionales de gran escala, además se incluye colecciones de funciones matemáticas fundamentales para la calculadora.
- TKinter: Se usa para crear las interfaces gráficas del programa.
- Matplotlib: Sus funciones se encargan de generar las gráficas en dos dimensiones a partir de listas de datos o arreglos, además de implementar la posibilidad de desplegar código LaTex para las funciones matemáticas.
- Firebase_admin: Se usa para interactuar con el conjunto de servicios de Firebase, más específicamente la base de datos en tiempo real.
- Pyrebase: Se usa para agregar la funcionalidad de authentication al programa de la calculadora.
- Sys: Se usa para importar los diferentes archivos del programa al hacer uso de la función sys.path.insert
- Time: Se usa para poder temporizar el tiempo de ejecución de ciertas funciones de model

1.2. Estructura Modelo Vista Controlador

El programa de la calculadora gráfica está organizado en tres carpetas principales: “1model”, “2controller” y “3view”. Cada carpeta tiene un propósito específico, siguiendo una arquitectura Modelo-Vista-Controlador (MVC), que facilita la organización del código y la interacción entre componentes. A continuación, se detalla la función de cada carpeta y las dependencias entre los scripts:

- “**1models**”: Contiene las credenciales y métodos para la conexión y manipulación de la base de datos, métodos los cuales permiten la del CRUD.

- “**2controller**”: Actúa como un intermediario entre el modelo y la vista. Esta carpeta contiene los scripts que manejan la lógica de funcionamiento de la aplicación, es decir, cómo se procesan, organizan y manipulan las funciones matemáticas antes de ser enviadas al modelo o a la vista.

- “**3view**”: Encargada de la interfaz de usuario que contiene la visualización de la sección de inicio de sesión, las funciones matemáticas y las gráficas. Esta carpeta interactúa solamente con la **2controller** para enviar datos y asimismo recibir datos procesados.

1.3. Diagrama de Dependencias entre Carpetas y Scripts:

1.models

└── controller_Model.py

└── certificates.json

└── funcionesDB.py

2.controller

└── controller_Model.py

└── operaciones.py (importa desde controller_Model.py)

3.view

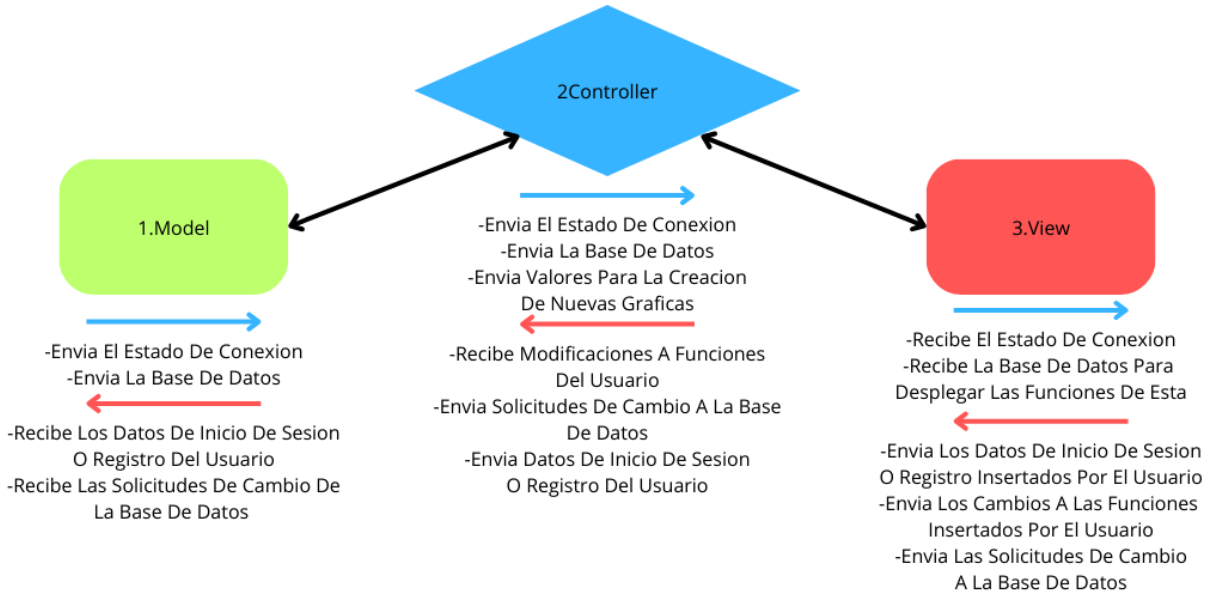
└── GUItkinter.py (importa desde controller.py y operaciones.py)

└── GUIBotones.py

└── GUIinicioDeSesion.py

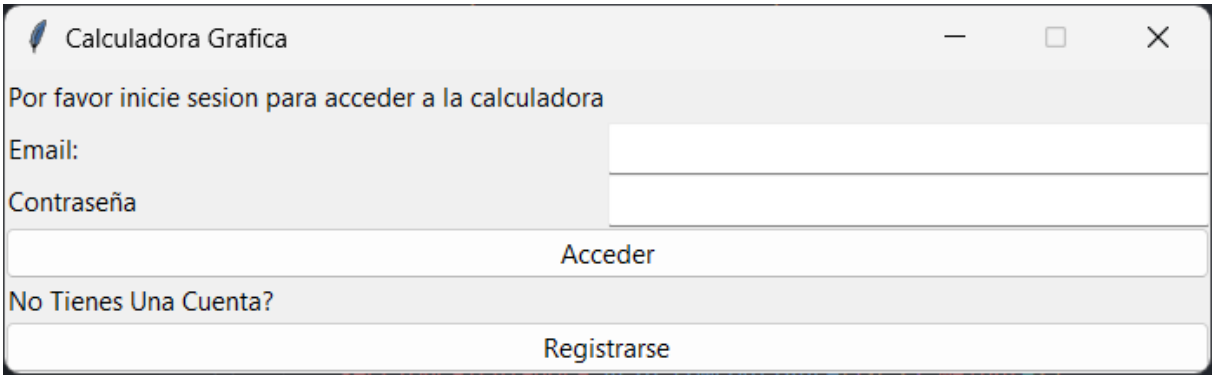
La comunicación entre las carpetas se da de la siguiente manera, las flechas azules indican la comunicación a la carpeta del lado derecho, ya sea recibir o enviar datos de esa

carpeta, mientras que las flechas rojas indican la comunicación a la carpeta del lado izquierdo, también indicando tanto la recepción o envío de datos:

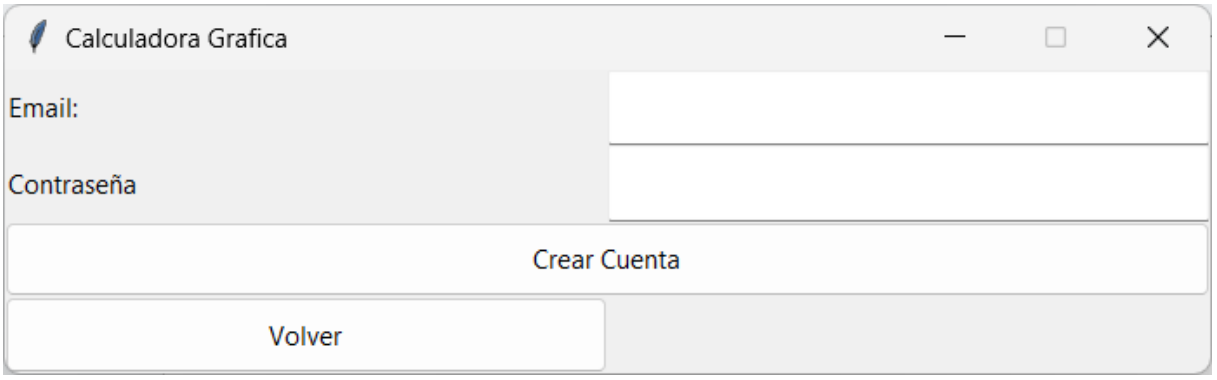


1.4. Vista Del Programa

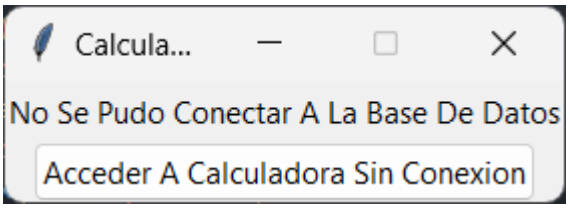
Sección Inicio De Sesión Cuando Hay Conexión A Internet:



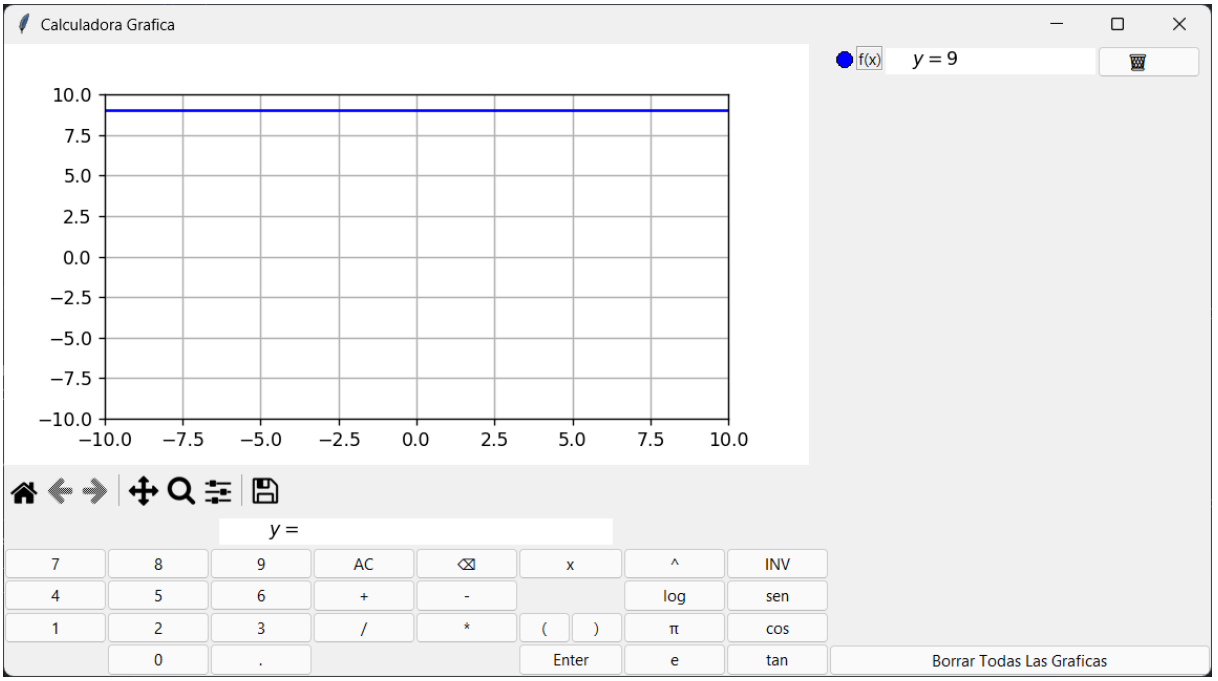
Sección Creada Al Presionar El Botón Registrarse:



Sección De Inicio De Sesión Cuando No Hay Conexión A Internet:



Sección Calculadora Gráfica.



1.5 Estructura Base De Datos

La estructura de la base de datos es la siguiente:

En esta se organizan los usuarios con respecto a su código de usuario de authentication UID, dentro de cada nodo correspondiente a cada usuario están las gráficas que el usuario creo y dentro de estas están la información de estas

2. Detalle de Cada Script y Su Funcionalidad:

2.1. **Carpeta 1model:**

2.1.1. **connector.py**

- **Propósito:** Se conecta a la base de datos y a los servicios de Authentication de firebase, dependiendo si la conexión es exitosa o no, la variable dbStatusOn se cambia a True o False dependiendo de si se pudo dar la conexión.

  Explicación Detallada “connector.py”:

  La función del archivo “conector.py” es, en palabras simples, conectarse a la base de datos (siempre y cuando se tenga acceso a internet), si no se tiene acceso a internet se modifica la variable dbStatusOn a False.

  Para acceder a la base de datos se importa la librería de firebase y se accede a la dirección de la llave de acceso .JSON se accede a la base de datos, y si no se tiene acceso a internet, por medio de una llave se accede al espacio de memoria.

```python
def fireBaseInitializeCreds():
    cred = credentials.Certificate("C:/Users/joans/Downloads/Proyecto-Programacion-De-Computadores
    firebase_admin.initialize_app(cred, {
            'databaseURL': 'https://calc-graficadora-v-experiment-default-rtdb.firebaseio.com'
    })
    ref = db.reference('/')
    ref.get()
```

La función “initializeAuth” es la que se conecta a la sección de Authentication de firebase

```python
def initializeAuth():
    global auth
    firebaseConfig = {
        "apiKey": "AIzaSyDjwFcPsfplapZYKbNugyGfFKQXyhE-os4",
        "authDomain": "calc-graficadora-v-experiment.firebaseapp.com",
        "databaseURL": "https://calc-graficadora-v-experiment-default-rtdb.firebaseio.com",
        "projectId": "calc-graficadora-v-experiment",
        "storageBucket": "calc-graficadora-v-experiment.appspot.com",
        "messagingSenderId": "900247884273",
        "appId": "1:900247884273:web:a4ef8561f544af10bef450",
        "measurementId": "G-NE8D5B6S8J"
    }
    firebase = pyrebase.initialize_app(firebaseConfig)
```

La función “checkFIrebase” verifica por cinco segundos si hay conexión con la base de datos

```python
def checkFirebase():
    try:
        tiempoDemora = timeit.timeit(stmt="fireBaseInitializeCreds()",setup="from connector import fireBaseInitializeCreds",number=1)
        if float(tiempoDemora) > 5:
            print(f"Tiempo De Conexion A Firebase Excedido {round(float(tiempoDemora),2)} Segundos")
            return False
    except Exception as e:
        print(f"Error De Conexion a Firebase: {e} ")
        return False
    else:
        return True
```

En caso tal de que no se haya podido realizar la conexión con firebase o se demoró más de 5 segundos en conectarse, se retorna False

La función “checkAuth” cumple una función similar a la función “checkFirebase” pero para revisar la conexión a Authentication, si anteriormente no se pudo conectar a firebase, se intuye en el programa que no hay conexión, por lo que se retorna False:

```python
def checkAuth():
    global dbStatusOn
    if dbStatusOn == True:
        try:
            tiempoDemora = timeit.timeit(stmt="initializeAuth()",setup="from connector import initializeAuth",number=1)
            if float(tiempoDemora) > 5:
                print(f"Tiempo De Conexion A Authentication Excedido: {round(float(tiempoDemora),2)} Segundos")
                return False
        except Exception as e:
            print(f"Error de conexion a authentication: {e}")
            return False
        else:
            return True
    else:
        return False
```

La función “checkBoth” se encarga de ejecutar ambas funciones de chequeo de conexión, con el fin de modificar la variable dbStatusOn dependiendo de

que retornan estas funciones, esta función se ejecuta cada vez que se inicializa el programa.

```
def checkBoth():
    global dbStatusOn
    dbStatusOn = checkFirebase()
    if dbStatusOn == True and checkAuth() == True:
        return True
    else: return False

dbStatusOn = checkBoth()
```

2.1.2. **funcionesDB.py:**

- **Propósito:** Contiene todas las funciones básicas que interactúan directamente con la base de datos, esto significa que implementa la funcionalidad de CRUD y también almacena la variable dbOffline que hace referencia al espacio de memoria en donde se almacenan todas las funciones de forma temporal en caso tal de que no haya conexión.

  Dependencias: Importa connector

  Variables:

  - UIDUser: Hace referencia al valor de la UID de usuario de authentication
  - dbOffline: Es el diccionario en donde se almacenan las funciones en caso tal de que no haya conexión a internet

  Explicación Detallada de “funcionesDB.py”

  La función “returnDBStatus” retorna el valor booleano de la variable de connector dbStatusOn, si retorna “True” quiere decir que la persona si se pudo conectar a la base de datos, es decir, tiene conexión a internet, si retorna “False” es porque el usuario no tiene conexión a internet, por ende todo se almacenará en el espacio de memoria

```
def returnDBStatus():
    global dbStatusOn
    return dbStatusOn
```

La función login se encarga de hacer la conexión con authentication para referirse a cada usuario, en caso tal de que la conexión haya sido exitosa, se

modifica la variable UIDUser que hace referencia a la variable del mismo nombre encontrada en authentication de firebase

```
def login(email:str,password:str):
    global UIDUser
    if dbStatusOn == True:
        try:
            user = auth.sign_in_with_email_and_password(email,password)
            UIDUser = auth.get_account_info(user["idToken"])["users"][0]["localId"]
        except:
            return False
        else:
            return True
    else:pass
```

La función registró se encarga de crear el nuevo usuario en la funcionalidad de authentication en caso tal de que no haya ocurrido algún error a la hora de hacer uso de la funcionalidad de pyrebase, cuando se crea el usuario también se crea el espacio en la base de datos y se agrega un nodo llamado filler, este nodo es fundamental, ya que a la hora de que el nodo del usuario quede con solamente una función, y se borra esta última, se borraría toda la base de datos si no fuera por el nodo filler.

```
def registro(email:str,password:str):
    global UIDUser
    if dbStatusOn == True:
        try: #Crear Usuario
            user = auth.create_user_with_email_and_password(email,password)
            UIDUser = auth.get_account_info(user["idToken"])["users"][0]["localId"]
        except:
            return False
        else:
            db.reference(f"/Usuarios").update({UIDUser:{"Filler":True}})
            return True

    else:pass
```

La función “devolver Diccionario Completo” sirve para extraer todas las gráficas que se guardaron en la base de datos del usuario con el cual se inició sesión, es importante aclarar si el usuario no tiene conexión a internet, todas funciones guardarán, borrarán, editarán y extraerán información del espacio de memoria ya predispuesto que sería en este caso la variable dbOfffline

```python
def devolverDiccionarioCompleto():
    '''
    Para que la funcion funcione debe de haberse ejecutado login o registro primero
    '''
    global dbOffline
    if dbStatusOn == True:
        if type((db.reference(f"/Usuarios/").get())[UIDUser]) == type(None):
            return {}
        else:
            return (db.reference(f"/Usuarios/").get()[UIDUser])
    else:
        return dbOffline
```

La función “borrarNodo” se usa para borrar cualquier nodo, esta función la llama “borrarGrafica” en controller_Model.

```python
def borrarNodo(llave:str):
    '''
    llave debe de tener formato /Usuarios/UIDuser/Grafica
    '''
    global UIDUser,dbOffline
    if dbStatusOn == True:
        if list(db.reference(f"Usuarios/{UIDUser}/{llave}").get()) != 1:
            db.reference(f"Usuarios/{UIDUser}/{llave}").delete()
        else:
            db.reference(f"Usuarios/{UIDUser}/{llave}").update({"prueba":"prueba"})
    else:
        if llave[0] == "/":
            del dbOffline[llave[1:]]
```

La función “editarNodo” sirve para editar los valores de una gráfica ya establecida en la base de datos (es decir, la gráfica tiene que encontrarse si o si en la base de datos), si el usuario no tiene conexión a internet (es decir, está usando el modo Offline) todo esto se ejecutará con respecto a la variable dbOffline

```python
def editarNodo(llave:str,valor:str):
    global UIDUser,dbOffline
    if dbStatusOn == True:
        if llave in devolverDiccionarioCompleto():
            ref.update({llave:valor})
        else:
            print(f"La llave:{llave} no se encuentra en el diccionario")
    else:
        if llave in dbOffline:
            dbOffline[llave] = valor
        else:
            print(f"La llave:{llave} no se encuentra en el diccionario Offline")
```

la función “crearNodo” se utiliza para añadir una nueva gráfica a la base de datos y en caso de que el usuario no tenga conexión a internet, se crea el nodo en la variable dbOffline.

```python
def crearNodo(diccionario:dict):
    global dbOffline
    if dbStatusOn == True:
        if list(diccionario)[0] in devolverDiccionarioCompleto():
            diccionarioDataBase = devolverDiccionarioCompleto()
            print(f"El inicio del nodo: {list(diccionario)[0]} ya se encuentra en diccionarioDataBase {diccionarioDataBase}")
        else:
            print(f"Se crea la grafica con los siguientes valores: {diccionario}")
            db.reference(f"/Usuarios/{UIDUser}/").update(diccionario)
    else:
        if list(diccionario)[0] in dbOffline:
            print(f"El inicio del nodo: {list(diccionario)[0]} ya se encuentra en diccionarioDataBaseOffline {dbOffline} ")
        else:
            print(f"Se crea la grafica con los siguientes valores: {diccionario}")
            for key in diccionario:
                dbOffline[key] = diccionario[key]
```

La función “borrarTodo” sirve para borrar TODAS las gráficas que están en la base de datos, esta función la llama “borrarTodasLasGraficas” en controller

```python
def borrarTodo():
    global dbOffline
    if dbStatusOn == True:
        ref.delete()
    else: dbOffline = {}
```

Con el fin de que no se tenga que ejecutar una y otra vez la función de editarNodo para cada vez que se quiere editar una función matemática, se crea la función “editarFuncionOLatexGrafica” la cual dependiendo del número de la gráfica, el tipo de dato que se quiere editar y el valor por el cual se quiere cambiar, se modifica el dato con el valor correspondiente.

```python
78
79  def editarFuncionOLatexGrafica(numeroGrafica:int,tipo:str,parametro:str):
80      '''
81      Para que esta funcion funcione debe de haberse primero ejecutado la funcion login o registro
82
83      '''
84      global UIDUser, dbOffline
85      tiposDisponibles = ["funcion","latexDisplay"]
86      if dbStatusOn == True:
87          if tipo in tiposDisponibles:
88              if f"Grafica{numeroGrafica}" in devolverDiccionarioCompleto():
89                  print(UIDUser)
90                  db.reference(f"/Usuarios/{UIDUser}/Grafica{numeroGrafica}").update({tipo:parametro})
91              else:
92                  print(f"Grafica{numeroGrafica} no se encuentra en la base de datos")
93      else:
94          if tipo in tiposDisponibles:
95              if f"Grafica{numeroGrafica}" in dbOffline:
96                  dbOffline[f"Grafica{numeroGrafica}"][tipo] = parametro
97              else:
98                  print(f"Grafica{numeroGrafica} no se encuentra en la base de datos offline")
```

2.2. **Carpeta 2controller:**

2.2.1. **controller_Model.py:**

- **Propósito:** Este script recibe y envía las solicitudes de modificación de la base de, también retorna la base de datos y variables que facilitan la creación de nuevas funciones. Hereda funciones que vienen de funcionesDB con el fin de que se mantenga la estructura de MVC.

  Dependencias: Importa funcionesDB

  Explicación Detallada de "Controller_Model.py"

  La función "devolverNumeroUltimaGrafica" lo que realiza es extraer el último número con el que se guardó la última gráfica que se añade a la Data Base, esta función es muy importante, para que no se repitan números de gráficas y así no ocurra un error en la Data Base

```
def devolverNumeroUltimaGrafica(): #Getter

    if funcionesDB.devolverDiccionarioCompleto() != {}:
        diccionarioActualDB = funcionesDB.devolverDiccionarioCompleto()
        if funcionesDB.returnDBStatus() == True:
            diccionarioActualDB.pop("Filler")
        diccionarioActualLlaves = list(diccionarioActualDB.keys())
        listaNumerosGraficas = [] #= [1,4,6,3,2]
        for key in diccionarioActualLlaves:
            listaNumerosGraficas.append(int(key[7:]))
        listaNumerosGraficas.sort()
        if listaNumerosGraficas != []:
            valorUltimaGrafica = listaNumerosGraficas[-1]
        else: valorUltimaGrafica = 0
        return valorUltimaGrafica
    else:
        print("La base de datos esta vacia")
        return 0
```

  La función "cambiarGrafica" se dispone para actualizar los valores de una gráfica, esto es necesario para cuando el usuario quiera cambiar una función, para que todo esto sea posible, esta función llama funciones del archivo "FuncionesDB"

```
def cambiarGrafica(numeroGrafica:int,funcion:str,latexDisplay:str): #Setter
    funcionesDB.editarFuncionOLatexGrafica(numeroGrafica,"funcion",funcion)
    funcionesDB.editarFuncionOLatexGrafica(numeroGrafica,"latexDisplay",latexDisplay)
```

La función “añadirUnanuevaGrafica” añade una nueva gráfica a la base de datos, para ello extrae el nuevo número de gráficas y lo añade como llave o “Key” y el valor de esa llave son las variables, que evidentemente son únicas para cada gráfica, para que todo esto sea posible, esta funciona llama a una función de la carpeta “funcionesDB”.

```
def añadirUnaNuevaGrafica(funcion:str,color:str,parentesisAbiertosUsados:int,parentes
    if parentesisAbiertosUsados==parentesisCerradosUsados and igualesUsados==1:
        diccionarioGrafica = {f"Grafica{numeroGrafica}":
                              {"color":color,
                               "funcion":funcion,
                               "igualesUsados":igualesUsados,
                               "parentesisAbiertosUsados":parentesisAbiertosUsados,
                               "parentesisCerradosUsados":parentesisCerradosUsados,
                               "nomenclatura":nomenclatura,
                               "latexDisplay":latexDisplay}
                             }
        funcionesDB.crearNodo(diccionarioGrafica)
    else:
        print("Error al añadir una nueva grafica")
```

Las funciones “borrarGrafica” y “borrarTodasLasGraficas” lo que hacen (como su nombre lo indica) es borrar tanto una o todas las gráficas de la base de datos (depende de como lo requiera el usuario), estas funciones llaman a funciones del archivo “FuncionesDB” para que así se efectúe el cambio.

```
def borrarGrafica(numeroGrafica):
    if (f"Grafica{numeroGrafica}") in funcionesDB.devolverDiccionarioCompleto():
        nodo = str(f"/Grafica{numeroGrafica}")
        funcionesDB.borrarNodo(nodo)
    else: pass
```

```
def borrarTodasLasGraficas():
    funcionesDB.borrarTodo()
```

También hereda las funciones de inicio de sesión, registro, devolver estado de conexión y devolver todo el diccionario de funcionesDB:

```python
def login(email:str,password:str):
    return funcionesDB.login(email,password)

def register(email:str,password:str):
    return funcionesDB.registro(email,password)

def devolverEstadoConexion():
    return funcionesDB.returnDBStatus()

def devolverTodoElDiccionario(): #Getter
    return funcionesDB.devolverDiccionarioCompleto()
```

2.2.2. **operaciones.py**:

- **Propósito:** Este script contiene las funciones para construir y manipular expresiones matemáticas. Define cómo las funciones de usuario se construyen, interpretan, y almacenan en la base de datos.
- **Dependencias:** Importa `controller_Model.py` para interactuar con la base de datos.
- ***Variables:***
    1. opVar: Es el String en donde se almacena las operaciones, funciones y números que el usuario indica
    2. latexDisplay: Es el string que almacena los comandos equivalentes de LaTeX para las funciones que el usuario pone en opVar, por ejemplo si el usuario pone una multiplicación, en LaTeX display se añade un \cdot (·)
    3. parentesisAbiertosUsados: Es un int que almacena cuantas veces el usuario uso paréntesis abiertos
    4. parentesisCerradosUsados: Es un int que almacena cuantas veces el usuario uso paréntesis cerrados
    5. numeroIguales: Es un int que almacena cuantos iguales se han usado (siempre va a ser 1)
    6. numeroGrafica: Es un int el cual se refiere a que grafica se está modificando, si el int es diferente a 0, se refiere a que se está editando una gráfica ya existente
- **Funciones Clave:**
    1. Manipulación de Expresiones: “addNum()”, “addOperator()”, “parentesisAbierto()”, “parentesisCerrado()”, etc., para construir la expresión matemática del usuario.
    2. Funciones de Reset y Control: “AC()”, “erase()”, la primera para borrar todo lo que está el cuadro de texto y la segunda para borrar un carácter o función especial

3. Interacción con Base de Datos: “mandarABaseDeDatos()”, “cambiarDeBaseDatosSoloLaFuncion()”, para almacenar o modificar gráficas en la base de datos.
4. Funciones Trigonométricas y Logarítmicas: “trigonométrica()”, “logaritmos()”, para manejar funciones especiales.
5. Validación y Conversión: “strToFunction()”, para convertir la expresión en un formato para usarlo con la función eval.

Explicación Detallada de “Operaciones.py”:

Estas son todas las variables en las que desarrolla el programa, así como los datos que se van a guardar en la Data Base

```
#VARIABLES
opVar = "y=" #VARIABLE EN LA CUAL SE REALIZA LAS OPERACIONES
latexDisplay = r"y="
parentesisAbiertosUsados = 0
parentesisCerradosUsados = 0
numeroIguales = 1
numeroGrafica = 0
```

Las siguientes funciones son para agregar números, operaciones, paréntesis (tanto derecho como izquierdo) y los caracteres “x” y “=” respectivamente, en ellas se evalúan algunos casos de errores (como por ejemplo cuando el usuario NO pone una operación antes o después de la “x”), en estos casos la función determina que colocar para que así no haya errores a la hora de evaluar la función. Aclarar que OpVar es la variable en donde se almacenan los números y funciones que ingresó el usuario. En todas las funciones declaramos la palabra reservada “global” simplemente para poder acceder y modificar las variables que están por fuera de esas funciones definidas.

```
def addNum(num:str):
    global opVar,latexDisplay,numeroIguales
    if numeroIguales !=0: #opVar = y=5*x*5
        if opVar[-1] == "x" or opVar[-1] == ")":
            opVar += f"*{num}"
            latexDisplay += fr"{diccionarioOpLatex["*"]}{num}"
        elif (opVar[-1] in listOperators) or (opVar[-1] in digits) or (opVar[-1]=="(") or (opVar[-1]=="."):
            opVar += num
            latexDisplay += num
        else:
            pass
    else:
        opVar = "ERROR"
    return opVar
```

Figura: Función para añadir números a opVar y latexDisplay

```python
def addOperator(operator:str):
    global opVar,latexDisplay
    if operator in listOperators[:4]:
        if opVar[-1] in digits or opVar[-1] == ")":
            opVar += operator
            latexDisplay += fr"{diccionarioOpLatex[operator]}"
        if opVar[-2:]  :=="**"
            opVar = opVar[:-2]+operator
            latexDisplay =  latexDisplay[:-1]+diccionarioOpLatex[operator]
        elif operator == "-" and (opVar[-1]=="=" or opVar[-1]=="("):
            opVar += "-"
            latexDisplay += "-"
        elif opVar[-1] == ".":
            opVar+=f"0{operator}"
            latexDisplay+=f"0{diccionarioOpLatex[operator]}"
        elif opVar[-1] in listOperators[:4]:
            latexDisplay = fr"{latexDisplay[:-len(diccionarioOpLatex[opVar[-1]])]+diccionarioOpLatex[operator]}"
            opVar = opVar[:-1]+operator
        else:
            pass
```

Figura: Función para añadir operaciones a opVar y latexDisplay

```python
    elif operator == ".":
        if opVar[-1] in digits or opVar[-1] == ")":
            opVar += "."
            latexDisplay += "."
        elif opVar[-1] ==")":
            opVar += "*0."
            latexDisplay += f"{diccionarioOpLatex['*']}0."
        elif opVar[-1] == "(":
            opVar += "0."
            latexDisplay += "0."
        else:
            pass
    else:
        pass
    return opVar
```

Figura: Manejo de casos de error a la hora de aplicar un punto como operador

```
def parentesisAbierto():
    global opVar,parentesisAbiertosUsados,parentesisCerradosUsados,latexDisplay
    if numeroIguales != 0:
        if  opVar[-1] in listOperators or opVar[-1] =="(":
            opVar += "("
            latexDisplay += "("
            parentesisAbiertosUsados += 1
        elif (opVar[-1] in digits or opVar[-1] == ")"):
            opVar += "*("
            latexDisplay += r"\cdot ("
            parentesisAbiertosUsados += 1
        else:
            pass
    else:
        opVar = "ERROR"
    return opVar
```

Figura: Función que se encarga de añadir los paréntesis abiertos a la función y modificar la variable con este nombre

```
def parentesisCerrado():
    global opVar,parentesisAbiertosUsados,parentesisCerradosUsados,latexDisplay
    if len(opVar)>1:
        if parentesisAbiertosUsados == parentesisCerradosUsados and parentesisAbiertosUsados !=0:
            pass
        elif parentesisAbiertosUsados > parentesisCerradosUsados and (opVar[-1] in digits or opVar[-1] == ")"):
            opVar += ")"
            latexDisplay += ")"
            parentesisCerradosUsados +=1
        elif parentesisAbiertosUsados > parentesisCerradosUsados and (opVar[-1] in listOperators[:2]): #### EXCEPCIONES PARA / y *
            opVar += "0)"
            latexDisplay += "0)"
            parentesisCerradosUsados +=1
        elif parentesisAbiertosUsados > parentesisCerradosUsados and (opVar[-1] in listOperators[2:4]): #### EXCEPCIONES PARA / y *
            opVar += "1)"
            latexDisplay += "1)"
            parentesisCerradosUsados +=1
        else:
            print(f"Puede que haya algun error ya que numero de parentesis abiertos es:{parentesisAbiertosUsados} y el numero de pare
    else:
        pass
```

Figura: Función que se encarga de añadir los paréntesis cerrados a la función y modificar la variable con este nombre

```
def addXAxis():
    global opVar,latexDisplay
    if numeroIguales  != 0:
        if opVar == "0" and len(opVar)==1: #QUITAR
            opVar = "x"
            latexDisplay = "x"
        elif opVar[-1] == "0" and len(opVar)!=1: #Quitar
            opVar += "*x"
            latexDisplay += r"\cdot x"
        elif opVar[-1] in digits and opVar[-1] != "x":
            opVar += "*x"
            latexDisplay += r"\cdot x"
        elif opVar[-1] == ")" and len(opVar)!=1:
            opVar += "*x"
            latexDisplay += r"\cdot x"
        else:
            opVar += "x"
            latexDisplay += "x"
    else:
        opVar = "ERROR"
    return opVar
```

Figura: Función que se encarga de añadir la variable x a opVar y LatexDisplay

```
def equal():
    global opVar,numeroIguales,latexDisplay
    if opVar == "0":
        pass
    elif opVar[-1] in listOperators:
        pass
    elif opVar[-1] == "y":
        opVar += "="
        latexDisplay += "="
        numeroIguales +=1
    else:
        pass
    return opVar
```

Figura: Función que se encarga de añadir el igual a opVar y LatexDisplay

La función “AC” es necesaria para que se restablezca el opVar en su valor original “y=” cuando se inicia el programa o cada vez que se da enter y se quiera ingresar otra función, y a su vez es donde se reinician las variables que se guardan en la base de datos

```
def AC():
    global opVar,parentesisAbiertosUsados,parentesisCerradosUsados,numeroIguales,numeroGrafica,latexDispl
    opVar = "y="
    latexDisplay = r"y="
    parentesisAbiertosUsados = 0
    parentesisCerradosUsados = 0
    numeroIguales = 1
    numeroGrafica = 0
    return opVar
```

La función de borrar O “erase”, la hicimos para que borre de acuerdo al carácter o cadena de caracteres que se ingresaron, por ejemplo, cuando se ingresa la función trigonométrica “Sen” en vez de borrar carácter por carácter, se borra todo al instante, y esto se repite con todas las funciones que tengan más de 1 carácter.

```
def erase():
    global opVar,parentesisAbiertosUsados,parentesisCerradosUsados,numeroIguales,latexDisplay
    if len(opVar) == 1:
        print(f"Prueba: opVar:{opVar}, len(opVar):{len(opVar)}")
        opVar = "0"
        latexDisplay = "0"
    elif opVar[-1] == "i":
        opVar = opVar[:-2]
        latexDisplay = latexDisplay[:-3]
    elif opVar[-1] == "r":
        opVar = opVar[:-5]
        latexDisplay = latexDisplay[:-1]
    elif opVar[-1] == ".":
        opVar = opVar[:-1]
        latexDisplay = latexDisplay[:-1]
```

```
elif opVar[-1] == "(":
    if len(opVar)>3 and (opVar[-3:] == "ln("):
        opVar = opVar[:-3]
        latexDisplay = latexDisplay[:-4] #\ln(
        parentesisAbiertosUsados -= 1
    elif len(opVar)>4 and opVar[-4:] == "log(":
        opVar = opVar[:-4]
        latexDisplay = latexDisplay[:-5]
        parentesisAbiertosUsados -= 1
    elif len(opVar)>7 and opVar[-7:-1] in ["arccos(","arcsen(","arctan("]:
        opVar = opVar[:-7]
        latexDisplay = latexDisplay[:-9]
        parentesisAbiertosUsados -= 1
    elif len(opVar)>4 and (opVar[-4:] in ["sen(","cos(","tan("]):
        opVar = opVar[:-4]
        latexDisplay = latexDisplay[:-4]
        parentesisAbiertosUsados -= 1
    else:
        opVar = opVar[:-1]
        latexDisplay = latexDisplay[:-1]
```

```
    elif opVar[-1] == ")":
        opVar = opVar[:-1]
        latexDisplay = latexDisplay[:-1]
        parentesisCerradosUsados -=1
    elif opVar[-1] == "=":
        pass
    elif opVar[-1] == "*":
        if  opVar[-2:]=="**"
            opVar = opVar[:-2]
            latexDisplay = latexDisplay[:-1]
        else:
            opVar = opVar[:-1]
            latexDisplay = latexDisplay[:-len(diccionarioOpLatex["*"])]
    elif opVar[-1] == "/":
        opVar = opVar[:-1]
        latexDisplay = latexDisplay[:-len(diccionarioOpLatex["/"])]
    elif opVar[-1] in digits[:12] or opVar[-1] in listOperators[:2]:
        opVar = opVar[:-1]
        latexDisplay = latexDisplay[:-1]
    return opVar
```

La función “mandarABaseDeDatos” realiza el análisis de indeterminaciones o error de sintaxis a la hora de evaluar la función ingresada por el usuario, esto se realiza más específicamente para que el programa en caso de error (por ejemplo, la división por cero) el programa no se quede estancado

```
def mandarABaseDeDatos(color:str,nomenclatura:str): # MODIFICAR PORQUE PUEDE HABER POR EJEMPLO raiz de
    global opVar,parentesisAbiertosUsados,parentesisCerradosUsados,numeroIguales,latexDisplay
    definidaEn1000_y_1000=True
    strAfuncion = strToFunction()
    try:
        a = eval(strAfuncion,{"x":1000,"sen":np.sin,"cos":np.cos,"tan":np.tan,"arcsen":np.arcsin,"arct
    except SyntaxError:
        diferenciaParentesis = parentesisAbiertosUsados - parentesisCerradosUsados
        for i in range(0,diferenciaParentesis):
            parentesisCerrado()
            strAfuncion += ")"
    except ZeroDivisionError:
        definidaEn1000_y_1000 = False
    else:
        definidaEn1000_y_1000 = True

    try:
        a = eval(str(strAfuncion),{"x":-1000,"sen":np.sin,"cos":np.cos,"tan":np.tan,"arcsen":np.arcsin
    except ZeroDivisionError:
        definidaEn1000_y_1000 = False
    else:
        definidaEn1000_y_1000 = True
```

Las funciones “Logaritmos”y “trigonométrica” son para ingresar funciones especiales, como ya es sabido, en estas funciones se analizan casos de errores, por lo que si hay alguno, la función determina que ingresar al cuadro de texto, para que así no haya errores al evaluar las funciones

```python
def logaritmos(tipoLogaritmo:str):
    global opVar,latexDisplay, parentesisAbiertosUsados
    if tipoLogaritmo in ["log","ln"]:
        if len(opVar)>1:
            if opVar[-1] in digits or opVar[-1]==")":
                opVar += f"*{tipoLogaritmo}("
                latexDisplay += fr"\cdot \{tipoLogaritmo}("
                parentesisAbiertosUsados += 1
            elif opVar[-1] in listOperators or opVar[-1] == "(":
                opVar += f"{tipoLogaritmo}("
                latexDisplay += fr"\{tipoLogaritmo}("
                parentesisAbiertosUsados += 1
            else:
                pass
    else:
        print(f"el logaritmo llamado: {tipoLogaritmo} no es reconocido")
```

```python
def trigonometrica(funcionTrigonometrica:str):
    global opVar,latexDisplay,parentesisAbiertosUsados
    if funcionTrigonometrica in ["sen","cos","tan","arcsen","arccos","arctan"]:
        if funcionTrigonometrica[:3] != "arc":
            if len(opVar)>1 and (opVar[-1] in digits or opVar[-1] == ")"):
                opVar += f"*{funcionTrigonometrica}("
                latexDisplay += fr"\cdot {funcionTrigonometrica}("
                parentesisAbiertosUsados +=1
            elif len(opVar)>1 and (opVar[-1] in listOperators or opVar[-1]=="(") :
                opVar += f"{funcionTrigonometrica}("
                latexDisplay += f"{funcionTrigonometrica}("
                parentesisAbiertosUsados +=1
            else:
                pass
        else:
            if len(opVar)>1 and (opVar[-1] in digits or opVar[-1] == ")"):
                opVar += f"*{funcionTrigonometrica}("
                latexDisplay += fr"\cdot {funcionTrigonometrica[-3:]}^{{-1}}("
                parentesisAbiertosUsados +=1
            elif len(opVar)>1 and (opVar[-1] in listOperators or opVar[-1]=="(") :
                opVar += f"{funcionTrigonometrica}("
                latexDisplay += f"{funcionTrigonometrica[-3:]}^{{-1}}("
                parentesisAbiertosUsados +=1
            else:
                pass
    else:
```

la función “strToFunction” lo que realiza es quitar el “y=” de la función que se recibe, esto debido a que la palabra reservada “eval” no admite este tipo de nomenclatura, por ejemplo, el usuario ingresa “y= 2x+3” y esta función hace que se quite el “y=” lo que resulta al final “2x+3” para que “eval” pueda funcionar correctamente y no haya errores al calcular los valores de “y”

```
def strToFunction():
    global opVar
    opVarSinY = opVar
    for i in opVar:
        if i == "y":
            opVarSinY = opVarSinY.replace("y","")
        elif i == "=":
            opVarSinY = opVarSinY.replace("=","")
        else:
            pass
    if not("x" in opVarSinY): #ESTA ES UNA SOLUCION BAST
        opVarSinY += "+0*x"

    return opVarSinY
```

2.2.3 Explicación detallada de cómo se grafica:

La función “graficarFunciones” se encarga básicamente de colocar el recuadro donde se van a graficar las funciones, es decir, esta función se encarga de poner el “plano cartesiano” además de añadir la barra de herramientas para la gráfica

```
def graficarFunciones(frameGrafica):
    global ax,canvas, fig, toolbar
    fig = mfg.Figure()#
    ax = fig.add_subplot()
    ax.grid()
    canvas = FigureCanvasTkAgg(fig, master=frameGrafica)
    canvas.get_tk_widget().config(width=617,height=318)
    canvas.get_tk_widget().grid(column=0,row=0)
    toolbar = NavigationToolbar2Tk(canvas,frameGrafica,pack_toolbar = False)
    toolbar.update()
    toolbar.grid(column=0,row=1,sticky="W")
```

la función “graficarFuncionDeOpVar” es la función que grafica las funciones que se ingresan en el cuadro de texto, básicamente, lo que realiza es crear una lista de -1000 hasta 1000 en un intervalo de 0.001, estos valores serán los de “X” en el plano, por consiguiente se hace un diccionario para reemplazar todas las funciones especiales que se ingresaron en un lenguaje que entienda Python, y para finalizar se crea otra lista (esta será la lista para el eje “Y”) donde se reemplaza la “X” de la función ingresada por el usuario por cada número que se creó en la lista de “X” (que recordemos que es de -1000 hasta 1000 en un intervalo de 0.01). Con el fin de quitar las asíntotas, se hace otra lista de los valores posibles de y, pero solamente positivos, luego se editan los valores mayores a 1000 en la lista originaria por la variable de numpy llamada nan, con el fin de que matplotlib lo interprete como que no se debe de unir ese punto.

```
def graficarFuncionDeOpVar(color:str,numeroGrafica):
    global canvas, fig
    xlist = np.arange(-1000,1000,0.001)
    dictX = {"x":xlist,"sen":np.sin,"cos":np.cos,"tan":np.tan,"euler":np.e,"pi":np.pi,"log":np.log10,"ln":np.log,"arcsen":np.arcsin,
    ylist = eval(str(operaciones.strToFunction()),dictX)
    ylist2 = np.abs(np.array(ylist))
    ylist[ ylist2 > 1000] = np.nan

    diccionarioAxGraficas[f"axGrafica{numeroGrafica}"] = ax.plot(xlist,ylist,c=color)
    ax.set(xlim=(-10,10),ylim=(-10,10))
    canvas.draw()
```

Ya una vez teniendo las dos listas, se grafica la función teniendo en cuenta la lista de los valores de x y y.

2.3 Carpeta 3view:

2.3.1 **GUItkinter.py**:

Propósito: Este archivo es el main de toda la calculadora como tal, dentro de esta se ejecutan las funciones que se encargan de crear la sección de inicio de sesión y la calculadora como tal
Dependencias: importa las funciones de GUIinicioSesion, GUIBotones y Tkinter.

Dentro del archivo podemos encontrar dos funciones, siendo estas ejecutarCalculadora y run.

La función ejecutar calculadora se encarga de crear los frames, configurarlos y llamar las funciones del archivo GUIBotones que se encargan de colocar los botones, textos, canvas y demás de la calculadora. Luego de crearlos en el frame de la sección de la calculadora como tal, ejecuta la función tkraise para poder cambiar a esta vista

```python
def ejecutarCalculadora():
    global ventana
    ventana.resizable(True,True)
    seccionCalculadora = ttk.Frame(ventana)
    seccionCalculadora.grid(row=0,column=0,sticky="nsew")
    seccionCalculadora.grid_columnconfigure(0,weight=2)
    seccionCalculadora.grid_columnconfigure(1,weight=1)
    seccionCalculadora.grid_rowconfigure((0,2),weight=2)
    seccionCalculadora.grid_rowconfigure(1,weight=1)



    frameGrafica = ttk.Frame(seccionCalculadora)
    frameGrafica.grid(row=0,column=0,sticky="nsew")
    GUIBotones.graficarFunciones(frameGrafica)

    frameOperacion = ttk.Frame(seccionCalculadora)
    frameOperacion.grid(row=1,column=0)
    labelOperacion = tkinter.Label(frameOperacion)
    labelOperacion.grid(row=0,column=0)
    GUIBotones.inicializarLatex(labelOperacion)
    GUIBotones.graficarLatex()
    frameBotones = ttk.Frame(seccionCalculadora)
    frameBotones.grid(row=2,column=0,sticky="nsew")
    frameBotones.grid_columnconfigure((0,1,2,3,4,5,6,7),weight=1)
    frameBotones.grid_rowconfigure((0,1,2,3),weight=1)
    GUIBotones.botonesNumeros(frameBotones,seccionCalculadora)
    GUIBotones.botonesOperaciones(frameBotones,seccionCalculadora)
```

```python
botonEnter = ttk.Button(frameBotones,text="Enter",command=lambda:GUIBotones.enterButtonCommand(frameHistorial,labelOperacion,seccionCalculadora))
botonEnter.grid(row=3,column=5,sticky="nsew")

frameHistorial = ttk.Frame(seccionCalculadora)
frameHistorial.grid(row=0,column=1,rowspan=3)
frameHistorial.grid_rowconfigure((0,1),weight=1)
GUIBotones.botonesHistorial(frameHistorial,labelOperacion,seccionCalculadora)
seccionCalculadora.update()

seccionCalculadora.bind("<Return>",lambda e: GUIBotones.enterButtonCommand(frameHistorial,labelOperacion,seccionCalculadora))
seccionCalculadora.bind("<Configure>",lambda e:GUIBotones.cambiarTamaño(seccionCalculadora))

seccionCalculadora.tkraise()
```

La función run es la función que se ejecuta de primeras siempre que se comienza la aplicación, está lo que hace es configurar la ventana para mostrar la sección de inicio de sesión y ejecutar la función de GUIinicioDeSesion encargada de crear la parte de inicio de sesión.

```python
def run():
    ventana.grid_columnconfigure(0,weight=1,uniform="a")
    ventana.grid_rowconfigure(0,weight=1,uniform="a")
    GUIinicioSesion.creacionSeccionAcceso(ventana,ejecutarCalculadora)
```

2.3.2 **GulinicioSesion.py:**

- *Propósito*: Se encarga de crear las funciones que crean la interfaz de inicio de sesión y agregar sus funcionalidades.

- *Dependencias*: Importa funciones de “controller_Model.py” e importa la librería tkinter para crear los widgets de la interfaz de inicio de sesión.

Primero se definen las funciones encargadas de agregar la funcionalidad de login y registro del programa, estas lo que hacen es revisar si los datos ingresados por el usuario están completos y el correo posee una arroba, si estas condiciones se cumplen, se intenta hacer uso de la función login o registro de funcionesDB y en caso tal de que no se puede ejecutar el inicio de sesión o registro, se manda un mensaje de error.

```
def botonAccederComando(entryUsuario,entryContraseña,ejecutarCalculadora):
    email = str(entryUsuario.get())
    contraseña = str(entryContraseña.get())
    if contraseña == "" or email == "":
        tkinter.messagebox.showerror("Error Registro","Datos Incompletos")
    else:
        if "@" not in email:
            tkinter.messagebox.showerror("Error Registro","Falta un arroba en el email")
        else:
            boolLogin = com.login(email,contraseña)
            if boolLogin:
                y = ejecutarCalculadora()
            else:
                tkinter.messagebox.showerror("Error Inicio De Sesion","No Se Encuentra El Usuario Y/o Contraseña En La Base De Datos")

def botonCrearCuentaComando(entryUsuario,entryContraseña,ejecutarCalculadora,ventana):
    email = str(entryUsuario.get())
    contraseña = str(entryContraseña.get())
    if contraseña == "" or email == "":
        tkinter.messagebox.showerror("Error Registro","Datos Incompletos")
    else:
        if "@" not in email:
            tkinter.messagebox.showerror("Error Registro","Falta un arroba en el email")
        else:
            boolLogin = com.register(email,contraseña)
            if boolLogin:
                y = creacionSeccionAcceso(ventana,ejecutarCalculadora)
            else:
                tkinter.messagebox.showerror("Error Registro","Cuenta Ya Creada y/o Cuenta No Valida")
```

Luego se define la función botonRegistroComando, esta función se encarga de desplegar la sección de registro cuando se hace uso del botón de acceder.

```
def botonRegistroComando(ventana,ejecutarCalculadora):
    seccionRegistro = ttk.Frame(ventana)
    seccionRegistro.grid(row=0,column=0,sticky="nsew")

    seccionRegistro.tkraise()

    seccionRegistro.grid_rowconfigure((0,1,2,3),weight=1,uniform="a")
    seccionRegistro.grid_columnconfigure((0,1),weight=1,uniform="a")

    labelUsuario = ttk.Label(seccionRegistro,text="Email: ")
    labelUsuario.grid(row=0,column=0,sticky="nsew")

    entryUsuario = ttk.Entry(seccionRegistro)
    entryUsuario.grid(row=0,column=1,sticky='nsew')

    labelContraseña = ttk.Label(seccionRegistro,text="Contraseña")
    labelContraseña.grid(row=1,column=0,sticky="nsew")

    entryContraseña = ttk.Entry(seccionRegistro,show="*")
    entryContraseña.grid(row=1,column=1,sticky="nsew")

    botonCrearCuenta = ttk.Button(seccionRegistro,text="Crear Cuenta",command=lambda: botonCrearCuentaComando(entryUsuario,entryContraseña,ejecutarCalculadora,ventana))
    botonCrearCuenta.grid(row=2,column=0,columnspan=2,sticky="nsew")

    botonAtras = ttk.Button(seccionRegistro,text="Volver",command=lambda:creacionSeccionAcceso(ventana,ejecutarCalculadora))
    botonAtras.grid(row=3,column=0,sticky="nsew")
```

Finalmente, se define la función creacionSeccionAcceso, esta se encarga de crear la parte de inicio de sesión y registro. Si hay conexión a firebase y authentication, se crea la sección con la parte de inicio de sesión y registro, en caso tal de que no haya conexión, se crea un texto y un botón que permite el acceso a la calculadora sin conexión

```python
def creacionSeccionAcceso(ventana,comandoEjecutarCalculadora):
    ventana.resizable(False,False)
    seccionInicioDeSesion = ttk.Frame(ventana)
    seccionInicioDeSesion.grid(row=0,column=0,sticky="nsew")
    if com.devolverEstadoConexion() == True:
        '''SECCION INICIO DE SESION'''
        seccionInicioDeSesion.grid_rowconfigure((0,1,2,3),weight=1,uniform="a")
        seccionInicioDeSesion.grid_columnconfigure((0,1),weight=1,uniform="a")
        labelInicioDeSesion = ttk.Label(seccionInicioDeSesion,text="Por favor inicie sesion para acceder a la calculadora")
        labelInicioDeSesion.grid(row=0,column=0,sticky="nsew")

        labelUsuario = ttk.Label(seccionInicioDeSesion,text="Email: ")
        labelUsuario.grid(row=1,column=0,sticky="nsew")
        entryUsuario = ttk.Entry(seccionInicioDeSesion)
        entryUsuario.grid(row=1,column=1,sticky="nsew")

        LabelContraseña = ttk.Label(seccionInicioDeSesion,text="Contraseña")
        LabelContraseña.grid(row=2,column=0,sticky="nsew")
        entryContraseña = ttk.Entry(seccionInicioDeSesion,show="*")
        entryContraseña.grid(row=2,column=1,sticky="nsew")

        botonAcceder = ttk.Button(seccionInicioDeSesion,text="Acceder",command=lambda:botonAccederComando(entryUsuario,entryContraseña,comandoEjecutarCalculadora))
        botonAcceder.grid(row=3,column=0,columnspan=2,sticky="nsew")

        labelRegistro = ttk.Label(seccionInicioDeSesion,text="No Tienes Una Cuenta?")
        labelRegistro.grid(row=4,column=0,columnspan=2,sticky="nsew")

        botonRegistro = ttk.Button(seccionInicioDeSesion,text="Registrarse",command=lambda:botonRegistroComando(ventana,comandoEjecutarCalculadora))
        botonRegistro.grid(row=5,column=0,columnspan=2,sticky="nsew")

    else:
        seccionInicioDeSesion.grid_rowconfigure((0,1),weight=1,uniform="a")
        seccionInicioDeSesion.grid_columnconfigure((0,1),weight=1,uniform="a")
        LabelFalloConexion = ttk.Label(seccionInicioDeSesion,text="No Se Pudo Conectar A La Base De Datos")
        LabelFalloConexion.grid(row=0,column=0,columnspan=2)

        botonAccederABDOffline = ttk.Button(seccionInicioDeSesion,text="Acceder A Calculadora Sin Conexion",command=lambda:comandoEjecutarCalculadora())
        botonAccederABDOffline.grid(row=1,column=0,columnspan=2)

    ventana.mainloop()
```

2.3.3 **GUIBotones.py**:

- *Propósito*: Define los componentes de la calculadora gráfica como botones de operaciones y números, la sección de historial de la calculadora, el plano cartesiano de la función, la implementación con LaTeX y demás.

- *Dependencias*: Importa la librería de matplotlib, tkinter, numpy e importa funciones del archivo operaciones.

En el programa se crean diferentes variables que se encargaran de almacenar los widgets creados, tanto botones numéricos, canvas del historial, las operaciones permitidas y demás.

```python
diccionarioBotones = {}
diccionarioHistorial = {}
diccionarioScrollbar = {}
diccionarioAxGraficas = {}
operacionesPermitidas = ["+","-","/","*"]
```

También se crea diferentes listas que definen la secuencia de colores y nomenclaturas de las funciones matemáticas creadas por el usuario

```python
listaColores = ["blue","purple","cyan","orange","lime","darkturquoise","steelblue",
                "grey","silver","red","brown","salmon","tomato","orangered","darkorange",
                "burlywood","gold","chartreuse","greenyellow","limegreen","darkgreen", "aquamarine","turquoise",
                "teal","aqua","cyan","deepskyblue","dodgerblue","royalblue","navy","slateblue",
                "blueviolet","indigo","mediumorchid","violet","purple","magenta","crimson","pink"]
listaNomenclatura = ['f(x)', 'g(x)', 'h(x)', 'p(x)', 'q(x)', 'r(x)', 's(x)', 't(x)',
                     'f1(x)', 'g1(x)', 'h1(x)', 'p1(x)', 'q1(x)', 'r1(x)', 's1(x)', 't1(x)',
                     'f2(x)', 'g2(x)', 'h2(x)', 'p2(x)', 'q2(x)', 'r2(x)', 's2(x)', 't2(x)',
                     'f3(x)', 'g3(x)', 'h3(x)', 'p3(x)', 'q3(x)', 'r3(x)', 's3(x)', 't3(x)',
                     'f4(x)', 'g4(x)', 'h4(x)', 'p4(x)', 'q4(x)', 'r4(x)', 's4(x)', 't4(x)']
```

La primera función que se define es la función borrarTodoMensaje, está lo que hace es desplegar la ventana emergente que pregunta al usuario si está seguro de querer borrar todas las funciones, dependiendo de lo que ejecute el usuario, se borran todas las gráficas de la base de datos y también se borran de la sección de historial

```
def borrarTodoMensaje(frameH,labelOpWidget,window):
    messagebox = tkinter.messagebox.askquestion("Borrar Graficas","Desea Borrar todas las graficas de la base de datos?")
    if messagebox == "yes":
        operaciones.borrarTodasLasGraficasDeDB()
        global diccionarioScrollbar
        for key in diccionarioHistorial:
            if isinstance(diccionarioHistorial[key],mfg.Figure):
                pass
            elif isinstance(diccionarioHistorial[key],mpl.axes._axes.Axes):
                pass
            elif isinstance(diccionarioHistorial[key],FigureCanvasTkAgg):
                pass
            elif len(key) >17:
                if key[:17] == "ovaloColorFuncion":
                    diccionarioHistorial[key] = 0
                else:
                    diccionarioHistorial[key].destroy()
            else:
                diccionarioHistorial[key].destroy()
        diccionarioScrollbar["scrollBar"].destroy()
        ax.clear()
        botonesHistorial(frameH,labelOpWidget,window)
        ax.grid()
    else:
        tkinter.messagebox.showinfo("Volviendo","Volviendo a la calculadora")
```

Luego se define la función graficarLatex que se encarga de modificar el LaTeX principal del programa en donde se muestra la función que está creando el usuario, también crea un caso de error en caso tal de que solo haya un ^ en el LaTeX display, por lo que pone un espacio vacío.

```
def graficarLatex():
    if operaciones.latexDisplay[-1] == "^":
        textoLatex = "$"+operaciones.latexDisplay+"\ $"
    else:
        textoLatex = "$"+operaciones.latexDisplay+"$"
    axLatex.clear()
    axLatex.text(0, 0.3, textoLatex, fontsize=50)
    canvasLatex.draw()
```

Se define la función inicializarLatex que es la encargada de crear el cuadro que va a contener la funcionalidad de LaTeX del programa.

```
def inicializarLatex(labelOpDisplay):
    global axLatex, canvasLatex
    figLatex = mfg.Figure(figsize=(15,1),dpi=20)
    axLatex = figLatex.add_subplot(111)
    canvasLatex = FigureCanvasTkAgg(figLatex, master=labelOpDisplay)
    canvasLatex.get_tk_widget().pack(side="top", expand=True)
    axLatex.get_xaxis().set_visible(False)
    axLatex.get_yaxis().set_visible(False)
    axLatex.set_frame_on(False)
```

Se define la función operacionYDesplegarLatex que lo que se encarga es recibir la función que se quiere ejecutar para modificar la variable de

latexDisplay y opVar, y la ejecuta para así revisar el cambio de la variable latexDisplay y desplegarlo de nuevo en el cuadro de la función.

```
def operacionYDesplegarLatex(funcionOperacion):
    y = funcionOperacion
    if operaciones.opVar == "ERROR":
        tkinter.messagebox.showwarning("Declarar funcion","Se debe de declarar la funcion en forma y=")
        operaciones.AC()
    else:
        graficarLatex()
```

Se define la función graficarFunciones que se encarga de crear el plano cartesiano en donde se desplegaran las funciones.

```
def graficarFunciones(frameGrafica):
    global ax,canvas, fig, toolbar
    fig = mfg.Figure()#
    ax = fig.add_subplot()
    ax.grid()
    canvas = FigureCanvasTkAgg(fig, master=frameGrafica)
    canvas.get_tk_widget().config(width=617,height=318)
    canvas.get_tk_widget().grid(column=0,row=0)
    toolbar = NavigationToolbar2Tk(canvas,frameGrafica,pack_toolbar = False)
    toolbar.update()
    toolbar.grid(column=0,row=1,sticky="W")
```

Se define la función graficarFuncionDeOpVar que como dice su nombre, se encarga de graficar la función almacenada en la variable opVar, esto lo hace al tabular los valores de la función y los almacena en la lista xlist y ylist, para finalmente graficarlos con ayuda de matplotlib.

```
def graficarFuncionDeOpVar(color:str,numeroGrafica):
    global canvas, fig
    xlist = np.arange(-1000,1000,0.001)
    dictX = {"x":xlist,"sen":np.sin,"cos":np.cos,"tan":np.tan,"euler":np.e,"pi":np.pi,"log":np.log10,"ln":np.log,"arcsen":np.arcsin,"arctan":np.arctan,"arccos":np.arccos
    ylist = eval(str(operaciones.strToFunction()),dictX)
    ylist2 = np.abs(np.array(ylist))
    ylist[ ylist2 > 1000] = np.nan

    diccionarioAxGraficas[f"axGrafica{numeroGrafica}"] = ax.plot(xlist,ylist,c=color)
    ax.set(xlim=(-10,10),ylim=(-10,10))
    canvas.draw()
```

Se define la función botonesNumeros la cual se encarga de crear los botones de los números 0 al 9 y les añade sus funcionalidades.

```
def botonesNumeros(frameBotones,ventana):
    global diccionarioBotones
    numero = 7
    for rPosition in range(0,3):
        for cPosition in range(0,3):
            diccionarioBotones[f"Boton{numero}"]=ttk.Button(frameBotones,text=str(numero),command=lambda numero=numero : operacionYDesplegarLatex(operaciones.addNum(str(
            ventana.bind(f"{numero}",lambda e, numero=numero :operacionYDesplegarLatex(operaciones.addNum(str(numero))))
            numero += 1
        numero -= 6
    diccionarioBotones[f"Boton0"] =ttk.Button(frameBotones,text="0",command=lambda :operacionYDesplegarLatex(operaciones.addNum(str(0))))
    diccionarioBotones[f"Boton0"].grid(row=3,column=1,sticky="nsew")
    ventana.bind(f"0",lambda e:operacionYDesplegarLatex(operaciones.addNum(str(0))))
```

Se define la función botonesOperaciones que se encarga de crear los botones de todas las operaciones aritméticas y funciones disponibles de la calculadora.

```
def botonesOperaciones(frameBotones,ventana):
    global diccionarioBotones
    diccionarioBotones[f"BotonPunto"] =ttk.Button(frameBotones,text=".",command=lambda :operacionYDesplegarLatex(operaciones.addOperator(".")))
    diccionarioBotones[f"BotonPunto"].grid(row=3,column=2,sticky="nsew")
    ventana.bind(".",lambda e:operacionYDesplegarLatex(operaciones.addOperator(".")))

    diccionarioBotones[f"BotonAC"] =ttk.Button(frameBotones,text="AC",command=lambda :operacionYDesplegarLatex(operaciones.AC()))
    diccionarioBotones[f"BotonAC"].grid(row=0,column=3,sticky="nsew")

    diccionarioBotones[f"BotonBorrarAtras"] =ttk.Button(frameBotones,text="⌫",command=lambda:operacionYDesplegarLatex(operaciones.erase()))
    diccionarioBotones[f"BotonBorrarAtras"].grid(row=0,column=4,sticky="nsew")
    ventana.bind("<BackSpace>",lambda e:operacionYDesplegarLatex(operaciones.erase()))
```

```
index = 0
for rPosition in range(1,3):
    for cPosition in range(3,5):
        operacion = operacionesPermitidas[index]
        diccionarioBotones[f"Boton{operacion}"]=ttk.Button(frameBotones,text=operacion,command=lambda operacion=operacion:operacionYDesplegarLatex(operaciones.addOper
        diccionarioBotones[f"Boton{operacion}"].grid(row=rPosition,column=cPosition,sticky="nsew")
        ventana.bind(operacion,lambda e, operacion=operacion:operacionYDesplegarLatex(operaciones.addOperator(operacion)))
```

```
diccionarioBotones[f"BotonEjeX"]=ttk.Button(frameBotones,text="x",command=lambda:operacionYDesplegarLatex(operaciones.addXAxis()))
diccionarioBotones[f"BotonEjeX"].grid(row=1,column=5,sticky="nsew")
ventana.bind("x",lambda e:operacionYDesplegarLatex(operaciones.addXAxis()))
frameParentesis = ttk.Frame(frameBotones)
frameParentesis.columnconfigure((0,1),weight=1)
frameParentesis.rowconfigure(0,weight=1)
frameParentesis.grid(row=2,column=5,sticky="nsew")
diccionarioBotones[f"BotonParentesisAbierto"]=ttk.Button(frameParentesis,width=5,text="(",command=lambda:operacionYDesplegarLatex(operaciones.parentesisAbierto()))
diccionarioBotones[f"BotonParentesisAbierto"].grid(row=0,column=0,sticky="nsew")
diccionarioBotones[f"BotonParentesisCerrado"]=ttk.Button(frameParentesis,width=5,text=")",command=lambda:operacionYDesplegarLatex(operaciones.parentesisCerrado()))
diccionarioBotones[f"BotonParentesisCerrado"].grid(row=0,column=1,sticky="nsew")

diccionarioBotones["BotonPotencia"] = ttk.Button(frameBotones,text="^",command= lambda:operacionYDesplegarLatex(operaciones.potencia()))
diccionarioBotones["BotonPotencia"].grid(row=0,column=6,sticky="nsew")
diccionarioBotones["BotonLogaritmo"] = ttk.Button(frameBotones,text="log",command= lambda:operacionYDesplegarLatex(operaciones.logaritmos("log")))
diccionarioBotones["BotonLogaritmo"].grid(row=1,column=6,sticky="nsew")
diccionarioBotones["BotonPi"] = ttk.Button(frameBotones,text="π",command= lambda:operacionYDesplegarLatex(operaciones.specialVars("pi")))
diccionarioBotones["BotonPi"].grid(row=2,column=6,sticky="nsew")
diccionarioBotones["BotonEuler"] = ttk.Button(frameBotones,text="e",command= lambda:operacionYDesplegarLatex(operaciones.specialVars("euler")))
diccionarioBotones["BotonEuler"].grid(row=3,column=6,sticky="nsew")

diccionarioBotones["BotonInverso"] = ttk.Button(frameBotones,text="INV",command=lambda:botonInversa())
diccionarioBotones["BotonInverso"].grid(row=0,column=7,sticky="nsew")
diccionarioBotones["BotonSeno"] = ttk.Button(frameBotones,text="sen",command= lambda:operacionYDesplegarLatex(operaciones.trigonometrica("sen")))
diccionarioBotones["BotonSeno"].grid(row=1,column=7,sticky="nsew")
diccionarioBotones["BotonCoseno"] = ttk.Button(frameBotones,text="cos",command= lambda:operacionYDesplegarLatex(operaciones.trigonometrica("cos")))
diccionarioBotones["BotonCoseno"].grid(row=2,column=7,sticky="nsew")
diccionarioBotones["BotonTan"] = ttk.Button(frameBotones,text="tan",command= lambda:operacionYDesplegarLatex(operaciones.trigonometrica("tan")))
diccionarioBotones["BotonTan"].grid(row=3,column=7,sticky="nsew")
```

Se define la función botonInversa que se encarga de cambiar las funcionalidades y textos de los botones cada vez que se presiona el botón INV.

```
def botonInversa():
    if diccionarioBotones["BotonSeno"].cget("text") == "sen":
        diccionarioBotones["BotonSeno"].configure(text="arcsen",command= lambda:operacionYDesplegarLatex(operaciones.trigonometrica("arcsen")))
        diccionarioBotones["BotonCoseno"].configure(text="arccos",command= lambda:operacionYDesplegarLatex(operaciones.trigonometrica("arccos")))
        diccionarioBotones["BotonTan"].configure(text="arctan",command= lambda:operacionYDesplegarLatex(operaciones.trigonometrica("arctan")))
        diccionarioBotones['BotonLogaritmo'].configure(text="ln",command = lambda:operacionYDesplegarLatex(operaciones.logaritmos("ln")))
    else:
        diccionarioBotones["BotonSeno"].configure(text="sen",command= lambda:operacionYDesplegarLatex(operaciones.trigonometrica("sen")))
        diccionarioBotones["BotonCoseno"].configure(text="cos",command= lambda:operacionYDesplegarLatex(operaciones.trigonometrica("cos")))
        diccionarioBotones["BotonTan"].configure(text="tan",command= lambda:operacionYDesplegarLatex(operaciones.trigonometrica("tan")))
        diccionarioBotones['BotonLogaritmo'].configure(text="log",command = lambda:operacionYDesplegarLatex(operaciones.logaritmos("log")))
```

Se define la función enterButtonCommand la cual se encarga de agregar la funcionalidad al botón enter, está lo que hace es revisar si la función en la cual se trabaja es una ya existente en la base de datos, en caso tal de ser así, la modifica tanto en la base de datos como en la sección del historial. En caso de no existir la función en la base de datos, se crea esta en la base de datos y se agrega a la sección del historial.

```
def enterButtonCommand(frameH,labelOpWidget,window):
    numeroGraficaActual = operaciones.numeroGrafica
    print(f"el numero de la graficaActual es:{numeroGraficaActual}")
    if numeroGraficaActual !=0:
        operaciones.cambiarDeBaseDatosSoloLaFuncion(numeroGraficaActual,operaciones.opVar,operaciones.latexDisplay)

        for key in diccionarioHistorial:
            if isinstance(diccionarioHistorial[key],int):
                pass
            elif isinstance(diccionarioHistorial[key],mfg.Figure):
                pass
            elif isinstance(diccionarioHistorial[key],mpl.axes._axes.Axes):
                pass
            elif isinstance(diccionarioHistorial[key],FigureCanvasTkAgg):
                pass
            else:
                diccionarioHistorial[key].destroy()
        ax.clear()
        ax.grid()
        botonesHistorial(frameH,labelOpWidget,window)
        operaciones.AC()
    else:
        colorGraficaNueva = elegirColorGrafica()
        nomenclaturaGraficaNueva = elegirNomenclaturaGrafica()
        operaciones.mandarABaseDeDatos(colorGraficaNueva,nomenclaturaGraficaNueva)
        botonesHistorial(frameH,labelOpWidget,window)
        operaciones.AC()
        operacionYDesplegarLatex(operaciones.opVar)
```

Se define la función botonAccionGrafica la cual se encarga de agregar la funcionalidad a cada cuadro de LaTeX que haga referencia a una función de la base de datos. Cada vez que se ejecuta esta función, lo que se hace es modificar los parámetros de opVar, latexDisplay y demás variables de la sección de operaciones

```
def botonAccionGrafica(opVarCambio,parentesisAbiertosCambio,parentesisCerradosCambio,numeroIgualesCambio,numeroGrafica,latexDisplay):
    operaciones.changeValue(parentesisAbiertosCambio,"parentesisAbiertos")
    operaciones.changeValue(parentesisCerradosCambio,"parentesisCerrados")
    operaciones.changeValue(numeroIgualesCambio,"numeroIguales")
    operaciones.changeValue(numeroGrafica,"numeroGrafica")
    operaciones.changeValue(latexDisplay,"latexDisplay")
    operacionYDesplegarLatex(operaciones.changeValue(opVarCambio,"opVar"))
```

Se define la función más compleja del programa, la cual es botones Historial, esta agrega los botones, labels y figuras correspondientes a cada función de la base de datos, esta también agrega la funcionalidad de scrollBar a esta sección como tal.

```python
def botonesHistorial(frameH,labelOpWidget,window):
    global alturaLista, alturaLista, canvasScroll, frameBotonesCanvas, alturaFrameCanvas, diccionarioScrollbar
    diccionarioBaseDatos = operaciones.devolverBaseDeDatos()
    numeroGraficas = len(diccionarioBaseDatos)
    alturaItems = 2
    alturaFrameCanvas = 366
    if numeroGraficas != 0:
        alturaLista = numeroGraficas*30
    else:
        alturaLista = 0
    print(f"La altura de la lista es: {alturaLista} y la altura del frameCanvas es {alturaFrameCanvas}")
    canvasScroll = tkinter.Canvas(frameH,width=360,height=conseguirAlturaVentana(window)-30,scrollregion=(0,0,300,alturaLista))
    frameBotonesCanvas = ttk.Frame(window)
    rPosition = 0
    for key in diccionarioBaseDatos:
        if diccionarioBaseDatos[key]==True:
            pass
        else:
            numeroGrafica = int(key[7:])
            colorGrafica = diccionarioBaseDatos[key]["color"]
            nomenclaturaGrafica = diccionarioBaseDatos[key]["nomenclatura"]
            funcionGrafica = diccionarioBaseDatos[key]["funcion"]
            parentesisAbiertosGrafica = diccionarioBaseDatos[key]["parentesisAbiertosUsados"]
            parentesisCerradosGrafica = diccionarioBaseDatos[key]["parentesisCerradosUsados"]
            numeroIgualesGrafica = diccionarioBaseDatos[key]["igualesUsados"]
            latexDisplay = diccionarioBaseDatos[key]["latexDisplay"]
            diccionarioHistorial[f"colorFuncion{numeroGrafica}"] = tkinter.Canvas(frameBotonesCanvas,width=20,height=alturaItems*10)
            diccionarioHistorial[f"colorFuncion{numeroGrafica}"].grid(row=rPosition,column=0,sticky="n")
            diccionarioHistorial[f"ovaloColorFuncion{numeroGrafica}"]= diccionarioHistorial[f"colorFuncion{numeroGrafica}"].create_oval(5,5,20,20,fill=colorGrafica)
            diccionarioHistorial[f"labelNomenclaturaFuncion{numeroGrafica}"] = ttk.Label(frameBotonesCanvas,text=nomenclaturaGrafica,relief="groove")
            diccionarioHistorial[f"labelNomenclaturaFuncion{numeroGrafica}"].grid(row=rPosition,column=1,sticky="n")
            '''-----TEXTO LATEX-----'''
            textoLatex = "$"+latexDisplay+"$"
            diccionarioHistorial[f"contenedorLatexFuncion{numeroGrafica}"] = tkinter.Label(frameBotonesCanvas)
            diccionarioHistorial[f"contenedorLatexFuncion{numeroGrafica}"].grid(row=rPosition,column=2,sticky="n")
            diccionarioHistorial[f"figureFuncion{numeroGrafica}"] = mfg.Figure(figsize=(8,1), dpi=20)
            diccionarioHistorial[f"axFuncion{numeroGrafica}"] = diccionarioHistorial[f"figureFuncion{numeroGrafica}"].add_subplot(111)
            diccionarioHistorial[f"canvasFuncion{numeroGrafica}Latex"] = FigureCanvasTkAgg(diccionarioHistorial[f"figureFuncion{numeroGrafica}"],master=diccionarioHistor
            diccionarioHistorial[f"canvasFuncion{numeroGrafica}Latex"].get_tk_widget().grid(row=rPosition,column=2,sticky="n")
            diccionarioHistorial[f"axFuncion{numeroGrafica}"].get_xaxis().set_visible(False)
            diccionarioHistorial[f"axFuncion{numeroGrafica}"].get_yaxis().set_visible(False)
            diccionarioHistorial[f"axFuncion{numeroGrafica}"].set_frame_on(False)
            diccionarioHistorial[f"canvasFuncion{numeroGrafica}Latex"].get_tk_widget().bind("<Button-1>",lambda event,funcionGrafica=funcionGrafica,parentesisAbiertosGra
            diccionarioHistorial[f"axFuncion{numeroGrafica}"].clear()
            diccionarioHistorial[f"axFuncion{numeroGrafica}"].text(0, 0.3, textoLatex, fontsize=50)
            diccionarioHistorial[f"canvasFuncion{numeroGrafica}Latex"].draw()
            diccionarioHistorial[f"botonBorrarFuncion{numeroGrafica}"] = ttk.Button(frameBotonesCanvas,text="🗑",command= lambda numeroGrafica=numeroGrafica: borrarFunci
            diccionarioHistorial[f"botonBorrarFuncion{numeroGrafica}"].grid(row=rPosition, column= 3,sticky="n")
            operaciones.changeValue(funcionGrafica,"opVar")
```

```python
        graficarFuncionDeOpVar(colorGrafica,numeroGrafica)
        operaciones.AC()
        rPosition +=1

crearOBorrarScrollBar(window)

canvasScroll.create_window((0,0),
                   window=frameBotonesCanvas,
                   anchor="nw",
                   width= 360,
                   height= alturaLista)

canvasScroll.grid(row=0,column=0,sticky="n")

diccionarioHistorial["BorrarTodasLasGraficas"] = ttk.Button(frameH,text="Borrar Todas Las Graficas",command=lambda frameH=frameH,labelOpWidget=labelOpWidget,window=w
diccionarioHistorial["BorrarTodasLasGraficas"].grid(row=1,column=0,sticky="sew")
```

Finalmente, se define la función borrarFuncion la cual se encarga de borrar tanto de la base de datos como de la sección de las gráficas y el historial cada función correspondiente.

```python
def borrarFuncion(numeroGrafica,frameH,labelOp,window):
    global diccionarioScrollbar, canvasScroll
    diccionarioDB = operaciones.devolverBaseDeDatos()
    messagebox = tkinter.messagebox.askquestion("Borrar Funcion",f"Esta seguro que quiere borrar la funcion: {diccionarioDB[f"Grafica{numeroGrafica}"]["nomenclatura"]}?")
    if messagebox == "yes":
        for key in diccionarioHistorial:
            if  isinstance(diccionarioHistorial[key],int):
                pass
            elif isinstance(diccionarioHistorial[key],mfg.Figure):
                pass
            elif isinstance(diccionarioHistorial[key],mpl.axes._axes.Axes):
                pass
            elif isinstance(diccionarioHistorial[key],FigureCanvasTkAgg):
                pass
            else:
                diccionarioHistorial[key].destroy()
            if len(key) >17:
                if key[:18] == "ovaloColorFuncion":
                    del diccionarioHistorial[key]
                else: pass
        diccionarioScrollbar["scrollBar"].destroy()
        operaciones.borrarDeBaseDeDatos(numeroGrafica)
        ax.clear()
        ax.grid()
        botonesHistorial(frameH,labelOp,window)
        operaciones.AC()
        operacionYDesplegarLatex(operaciones.opVar)
        canvasScroll.update()
    else:
        tkinter.messagebox.showinfo("Volviendo","Volviendo a la calculadora")
```

3. Funcionamiento General del Programa y Descripción de Cada Parte

Propósito General del Programa:

El programa es una calculadora gráfica interactiva que permite a los usuarios ingresar y manipular funciones matemáticas, graficarlas en un plano cartesiano, y realizar operaciones matemáticas, también agrega la funcionalidad de diferentes usuarios con el fin de que puedan coexistir varios usuarios en la misma aplicación pero con funciones matemáticas diferentes. Utiliza una arquitectura basada en el patrón MVC para separar las responsabilidades de la lógica de negocio, la gestión de datos, y la presentación visual.

3.1. Inicialización (Carpeta 3view):

- Al iniciar, “GUItkinter.py” configura la ventana de inicio de sesión utilizando “GUIiniciosesion.py”.

3.2. Interacción del Usuario con los botones:

- Los usuarios interactúan con la interfaz de inicio de sesión creada con GUIiniciosesion.py y la interfaz de la calculadora creada con GUIBotones.py.

- GUIiniciosesion captura la información para crear una cuenta y si hay una cuenta ya existente o la manera en la que fueron ingresadas es incorrecta, creará un messagebox que le avise al usuario que caso ha sucedido.

- GUIBotones.py es la interfaz de la calculadora gráfica en la que están los botones de las funciones para escribir las funciones, la graficación e historial de las mismas.

3.3. Construcción y Manipulación de Expresiones (Carpeta 2controller):

- “operaciones.py” gestiona la construcción de la expresión matemática y su visualización en formato LaTeX.

- Se manejan casos especiales, como multiplicación implícita, funciones trigonométricas, y uso de paréntesis.

- Los cambios en las expresiones se reflejan inmediatamente en la interfaz gráfica.

3.4. Validación y Almacenamiento de Funciones (Carpeta 2controller y 1models):

- Una vez que el usuario está satisfecho con la función ingresada, puede ingresar el botón enter para graficarla y almacenarla en la base de datos de manera automática.

- “mandarABaseDeDatos()” en “operaciones.py” válida la función y llama a “controller_Model.py” para almacenarla en la base de datos.

3.5. Graficación y Visualización (Carpeta 3view):

- Cuando se solicita una gráfica, “GUItkinter.py” obtiene la función correspondiente desde `operaciones.py` la cual extrae de la base de datos usando `controller_Model.py`.

- La función se evalúa y gráfica utilizando la librería gráfica matplotlib, mostrando el resultado en la interfaz.

3.6. Gestión de Base de Datos (Carpeta 1models y 2Controller):

- Todas las funciones creadas y almacenadas pueden ser gestionadas: borradas, modificadas o recuperadas.

- **“controller_Model.py”** maneja todas las operaciones de CRUD con estructuras de datos específicas para los casos de uso.

Ya que en “funcionesDB” puede crear cualquier nodo con cualquier valor, es necesario “controller_Model” para limitar la creación de estos solamente a nodos con referencias a las funciones y sus graficar correspondientes.

4.Limitaciones

-Dirección local: Es necesario ajustar la localización de las carpetas de manera manual, lo cual significa una dificultad para que el usuario utilice el programa de manera cómoda. Se sugiere intentar la implementación de la librería “os” y solucionar la ruta de acceso con el módulo “os.path”.
-Asíntotas: Asíntotas de diferentes funciones se despliegan para casos en los que para valores en x de intervalos de 0.001 la función tiene valores menores a 1000 y mayores a -1000
-Velocidad De Ejecución: El programa tiene diferentes problemas de velocidad de ejecución, principalmente en los momentos en los que el usuario hace de la función de inicio de sesión o registro, esto depende en parte a la conexión a internet del programa. También está el problema de velocidad a la hora de expandir y disminuir la ventana de la calculadora gráfica.

5. Manejo De Excepciones

5.1. Carpeta 1model

5.1.1. connector:

checkFirebase(): Es una función diseñada para medir el tiempo que tarda en ejecutarse una función relacionada con la inicialización de Firebase y manejar posibles errores.

Manejo de Errores:

Si ocurre una excepción durante la medición del tiempo (por ejemplo, si hay un problema al importar o ejecutar fireBaseInitializeCreds),"Exception as e" captura la excepción en la variable "e", que contiene una descripción del error, se imprime un mensaje de error, y la función devuelve False.

Retorno en Caso de Éxito:

En este caso, si todo salió bien y no hubo ningún error, el tiempo de ejecución fue aceptable (es decir, no se excedió el límite de 6 segundos), la función devuelve True indicando que la conexión a Firebase fue exitosa.

checkAuth():La función "checkAuth()" tiene como objetivo verificar la conexión y el tiempo de respuesta para la autenticación en una aplicación, basándose en el estado de una variable global "dbStatusOn"

Manejo de Errores:

Funciona de igual manera al manejo de excepciones de la función "checkFirebase():" Si ocurre una excepción durante la autenticación con "Exception as e" captura la excepción en la variable "e" la cual contiene una descripción del error, luego se imprime un mensaje con el error correspondiente y la función también devuelve False.

Retorno en Caso de Éxito:

Si no hay excepciones y la autenticación del usuario fue exitosa, la función devuelve True. Al igual que en la función "checkFirebase()".

5.1.2. funcionesDB

login(email: str, password: str): está diseñada para autenticar a un usuario utilizando un correo electrónico y una contraseña, y actualiza una variable global con el ID del usuario.

Manejo de Errores:

Se le asignó un "try" para que con cualquier error probable la función devuelva False y el programa siga ejecutándose de manera correcta.

Registro(email:str, password:str): diseñada para registrar solamente strings relacionados a un nuevo usuario con un correo electrónico y una contraseña, y actualizar la base de datos con la información del nuevo usuario.

Manejo de Errores:

Se aplica la estructura de excepciones a las funciones "auth.create_user_with_email_and_password(email, password):","auth.get_account_info(user["idToken"])["users"][0]["localId"]:" Para evitar el fallo del programa al capturar todas las excepciones generales y devolver el valor controlado "False" indicando que el registro del usuario no fue exitoso.

5.2. Carpeta 2Controller

5.2.1. Operaciones

mandarABaseDeDatos(color:str, nomenclatura:str): Esta función se encarga de mandar a la base de datos la función dependiendo si está definida, por ejemplo si se quiere mandar la función $y = \frac{1}{0}$. Lo que se hace es evaluar en el valor 1000 y -1000 de x la función para revisar si está definida en esos puntos, en caso contrario no se envía a la base de datos

6. Conclusión

Este programa de calculadora gráfica permite a los usuarios trabajar con funciones matemáticas de manera interactiva, ofreciendo una forma estructurada de gestionar las operaciones, almacenamiento, y visualización de funciones complejas. La utilización del patrón MVC asegura una separación clara de responsabilidades, facilitando la escalabilidad y mantenimiento del código. El presente informe proporciona una visión detallada de cómo los diferentes componentes del programa interactúan para lograr una funcionalidad completa.

7. referencias

1. *Tkinter—Python interface to tcl/tk*. (n.d.). Python documentation. Recuperado Augosto 28, 2024, f https://docs.python.org/3/library/tkinter.html
2. *Matplotlib — Visualization with Python*. (s. f.). Recuperado el 27 de agosto de 2024 de https://matplotlib.org/
3. *NumPy*. (s/f). Numpy.org. Recuperado el 27 de Agosto de 2024 de https://numpy.org/
4. *Firestore. (s/f). Firebase. Recuperado el 28 de Agosto de 2024, de https://firebase.google.com/docs/firestore?hl=es-419*
5. *3.12.5 documentation. (s/f). Python.org. Recuperado el 24 de Agosto de 2024, de https://docs.python.org/3/*