UNIVERSIDAD
NACIONAL
DE COLOMBIA

**Universidad Nacional de Colombia-Sede Bogotá**
**Facultad de Ingeniería**
**Departamento de Ingeniería Eléctrica y Electrónica**
**Programación de computadores**

# *Manual de usuario de*
# Calculadora Gráfica

**Presentado por:**

**Byron Pedraza**

**Juan Manuel Toro Rojas**

**Manuel Alejandro Velasco Marinez**

**09/09/2024**

UNIVERSIDAD NACIONAL DE COLOMBIA

# Tabla de contenido


1. **Objetivos**
2. **Alcance**
3. **Requerimientos técnicos**
4. **Herramientas utilizadas para el desarrollador**
5. **Instalación**
6. **Configuración**
7. **Analisis de codigo**
8. **Diseño de arquitectura**
9. **Instrucciones**


# 1. Objetivos

El objetivo de este informe es mostrar a  grandes rasgos el diseño, sistema y funcionamiento de la calculadora gráfica, la cual tiene dos modos: básico y científico, y también puede abrir una calculadora gráfica externa. Además, se integra con Firebase para almacenar un historial de cálculos, esta nos ayudará a calcular todo tipo de variables algebraicas, ejercicios matemáticos trigonométricos, y rectas entre otros

# 2. Alcance

Mientras la calculadora cuente con internet, y tenga previo la información instalada y guardada la calculadora podrá crear gráficos de rectas, cálculos simples y complejos de aritmética, temas como rectas, puntos,,parábolas, partes de trigonometría avanzada como cosenos, senos,y demás funcionando perfectamente ante operaciones sencillas y complejas, la funcionalidad de esta calculadora aunque sencilla es muy veraz.

# 3. Requerimientos Técnicos

- Para el funcionamiento de la calculadora,necesitamos que tenga visual studio, que se encuentre en funcionamiento Microsoft C++, en temas de correr la calculadora es muy ligera y sencilla, funcionando con muy poco espacio de memoria, cabe recordar que tiene un límite el.historial y que funciona con internet.

# 4. Herramientas Utilizadas para el Desarrollo

- Python
- Firebase-admin
- Fire database
- Librerías
- Microsoft C++

# 5. Instalación

**Enlace al repositorio de github**

https://github.com/BayronP33/Proyecto-Calculadora

**Guía de Instalación de Librerías para Python Requisitos Previos:**

- Tener Python instalado.

- Tener Visual Studio Code (VS Code) instalado.
- Instalación de Librerías Python
- En VS Code, abre la terminal integrada presionando Ctrl+`
- Instalar las librerías usando pip:
    - Escribir los siguientes comandos en la terminal para instalar las librerías:
    - Pip,
    - Install,
    - Symp,
    - Numpy,
    - Matplotlib,
    - Firebase-admin.
- Explicación:
    - sympy para cálculos simbólicos.
    - numpy para operaciones numéricas.
    - matplotlib para gráficos.
    - firebase-admin para interactuar con Firebase.
- Descargar Microsoft C++ Build Tools:
    - matplotlib requiere componentes escritos en C y C++, por lo que es necesario instalar Microsoft C + + Build Tools para compilar estos componentes.
    - Ve a la página de Microsoft C + + Build Tools y descarga el instalador.
- Instalar Microsoft C++ Build Tools:
- Ejecuta el instalador descargado.
- En la pantalla de instalación, selecciona "Desarrollo para escritorio con C++".
- Asegúrate de incluir las opciones necesarias para la compilación, como las herramientas de línea de comandos y las bibliotecas estándar.
- Sigue las instrucciones en pantalla para completar la instalación.

- Una vez finalizada, reinicia tu computadora si es necesario.

## 6. Configuración

En principio, nuestra calculadora se hace un ingreso adicional del usuario, se encontrarán luego con la calculadora básica que en general es muy simple y sencilla de usar o la calculadora científica, la cual esta usa las funciones trigonométricas y ecuaciones un poco más avanzadas, en esta podrás observar las gráficas que se crean mediante la solución de la respuesta dada.

## 7. Análisis de código

7.0 análisis de código

**7.0.1. Inicialización (__init__)**

```
def __init__(self, root):
```

7.0.1.1 Configura la ventana principal con un título y tamaño inicial.

7.0.1.2 Define variables para almacenar la ecuación actual, el historial de cálculos, el estado de la calculadora (resultado mostrado o no).

7.0.1.3 llama al método “crear_interfaz” para construir la interfaz gráfica.

**7.0.2. Método crear_interfaz.**

```
def crear_interfaz(self):
    self.ventana_principal.title("Calculadora")
    self.ventana_principal.geometry("1000x600")

    self.marco_principal = tk.Frame(self.ventana_principal, bg="#8D99AE")
    self.marco_principal.pack(expand=True, fill='both')

    self.marco_entrada = tk.Frame(self.marco_principal, bg="#8D99AE")
    self.marco_entrada.grid(row=0, column=0, columnspan=10, ipadx=8, ipady=8, padx=10, pady=10, sticky="nsew")

    self.campo_entrada = tk.Entry(self.marco_entrada, textvariable=self.ecuacion, font=("Arial", 20), justify='right', bg="#2
    self.campo_entrada.pack(fill='both', expand=True)

    self.marco_historial = tk.Frame(self.marco_principal, bg="#8D99AE")
    self.marco_historial.grid(row=0, column=10, rowspan=6, sticky="nsew", padx=10, pady=10)

    self.panel_historial = tk.Listbox(self.marco_historial, height=10, font=("Arial", 14), bg="#2B2D42", fg="#EDF2F4")
    self.panel_historial.pack(fill='both', expand=True)
```

7.0.2.1 Configura el diseño de la ventana principal usando tk.Frame para dividir la ventana en secciones.

7.0.2.2 Crea un campo de entrada para mostrar y editar la ecuación.

7.0.2.3 Configura un panel de historial y un botón para cambiar entre diferentes modos (básico, científico, y graficadora).

### 7.0.3. Método “mostrar_opciones_modo”.

```
def mostrar_opciones_modo(self):
    if self.opciones_modo.winfo_ismapped():
        self.opciones_modo.pack_forget()
    else:
        self.opciones_modo.pack(pady=10)
```

7.3.1 Muestra u oculta las opciones para cambiar el modo de la calculadora.

### 7.0.4. Métodos modo_basico y modo_cientifico.

```
def modo_basico(self):
    self.vista.limpiar_botones()
    self.vista.ventana_principal.geometry("800x600")
    botones = [
        ('(', 1, 0), (')', 1, 1), ('7', 1, 2), ('8', 1, 3), ('9', 1, 4), ('÷', 1, 5),
        ('^', 2, 0), ('√', 2, 1), ('4', 2, 2), ('5', 2, 3), ('6', 2, 4), ('*', 2, 5),
        ('Del', 3, 0), ('AC', 3, 1), ('1', 3, 2), ('2', 3, 3), ('3', 3, 4), ('-', 3, 5),
        ('π', 4, 0), ('%', 4, 1), ('0', 4, 2), (',', 4, 3), ('=', 4, 4), ('+', 4, 5)
    ]
    self.vista.crear_botones(botones, font=("Arial", 20))
```

```
def modo_cientifico(self):
    self.vista.limpiar_botones()
    self.vista.ventana_principal.geometry("1000x600")
    botones = [
        ('sin', 1, 0), ('cos', 1, 1), ('tan', 1, 2), ('7', 1, 3), ('8', 1, 4), ('9', 1, 5), ('%', 1, 6), ('*', 1, 7), (
        ('asin', 2, 0), ('acos', 2, 1), ('atan', 2, 2), ('4', 2, 3), ('5', 2, 4), ('6', 2, 5), ('e', 2, 6), ('+', 2, 7
        ('sinh', 3, 0), ('cosh', 3, 1), ('tanh', 3, 2), ('3', 3, 3), ('2', 3, 4), ('1', 3, 5), ('!', 3, 6), ('÷', 3, 7
        ('(', 4, 0), (')', 4, 1), ('a/b', 4, 2), ('0', 4, 3), ('Del', 4, 4), ('AC', 4, 5), ('x', 4, 6), ('|a|', 4, 7),
        ('10^', 5, 0), ('Deg', 5, 1), ('Rad', 5, 2), ('=', 5, 3), ('.', 5, 4), ('log', 5, 5), ('ln', 5, 6), ('π', 5, 7
    ]
    self.vista.crear_botones(botones, font=("Arial", 12))
```

7.0.4.1 Configura la interfaz de botones de la calculadora según el modo seleccionado.

7.0.4.2 En el modo básico, se muestran botones para operaciones simples y funciones básicas.

7.0.4.3 En el modo científico, se añaden botones para funciones matemáticas avanzadas como trigonometría, logaritmos, y más.

### 7.0.5. Método crear_botones

```
def crear_botones(self, botones, font):
    for (texto, fila, columna) in botones:
        color_fondo = "#2B2D42"
        color_fuente = "white"
        if texto in {'+', '-', '*', '÷', '(', ')', 'π', 'V', '^', 'sin', 'cos', 'tan', 'log', 'ln', 'asin', 'acos', 'atan', 'sinh
            color_fuente = "#FCBF49"
        elif texto in {'AC', 'Del'}:
            color_fuente = "#EF233C"
        elif texto in {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '%', ',', '=', 'a/b', '10^', 'e', '|a|', 'nPr', 'nCr'}:
            color_fuente = "#EDF2F4"

        boton = tk.Button(self.marco_principal, text=texto, command=lambda t=texto: self.controlador.al_hacer_clic_en_boton(t),
                          font=font, bg=color_fondo, fg=color_fuente, width=4, height=2)
        boton.grid(row=fila, column=columna, sticky="nsew", ipadx=10, ipady=10, padx=5, pady=5)
        self.marco_principal.grid_rowconfigure(fila, weight=1)
        self.marco_principal.grid_columnconfigure(columna, weight=1)
```

7.0.5.1 Crea y organiza los botones en la interfaz gráfica según la disposición y el estilo especificado.

7.0.5.2 Asigna colores y acciones a los botones.

### 7.0.6. Método limpiar_botones

```
def limpiar_botones(self):
    for widget in self.marco_principal.winfo_children():
        if isinstance(widget, tk.Button):
            widget.destroy()
```

7.0.6.1 Elimina todos los botones de la interfaz para poder crear una nueva configuración de botones.

### 7.0.7 Método lanzar_graficadora

```
def lanzar_graficadora(self):
    subprocess.Popen(['python', 'control_cal_grafica.py'])
```

7.0.7.1 Abre un archivo externo (Calculadora_grafica.py) usando subprocess.

### 7.0.8. Método al_hacer_clic_en_boton

```
def al_hacer_clic_en_boton(self, caracter):
    if self.vista.resultado_mostrado:
        if caracter not in ('+', '-', '*', '÷', '^'):
            self.vista.ecuacion.set("")  # Reiniciar solo si no es un operador
            self.vista.resultado_mostrado = False
```

7.0.8.1 Maneja la lógica cuando se hace clic en un botón.
7.0.8.2 Actualiza la ecuación en el campo de entrada según el botón presionado.
7.0.8.3 Realiza cálculos o modifica la ecuación en función de los caracteres introducidos (números, operaciones, funciones).

### 7.0.9. Método calcular_resultado

```
def calcular_resultado(self, ecuacion, es_grado):
    try:
        # Reemplazar símbolos especiales
        ecuacion = ecuacion.replace('÷', '/').replace('^', '**').replace(',', '.').replace('π', 'math.pi')
        ecuacion = self.reemplazar_raices_cuadradas(ecuacion)
        ecuacion = self.manejar_porcentajes(ecuacion)
        ecuacion = ecuacion.replace('π', 'math.pi').replace('e', 'math.e')
        ecuacion = ecuacion.replace('10^', '10**')

        # Manejar factorial
        ecuacion = re.sub(r'(\d+)!', r'math.factorial(\1)', ecuacion)
        # Manejar permutación
        ecuacion = re.sub(r'nPr\(([^,]+),\s*([^,]+)\)', r'perm(\1, \2)', ecuacion)
        # Manejar combinación
        ecuacion = re.sub(r'nCr\(([^,]+),\s*([^,]+)\)', r'comb(\1, \2)', ecuacion)

        # Reemplazar funciones trigonométricas con math.<funcion>(
        funciones_trigonometricas = ['sin', 'cos', 'tan', 'asin', 'acos', 'atan', 'sinh', 'cosh', 'tanh']
        for funcion in funciones_trigonometricas:
            ecuacion = ecuacion.replace(funcion+'(', 'math.'+funcion+'(')

        if es_grado:
            ecuacion = self.ecuacion_grados_a_radianes(ecuacion)
```

7.0.9.1 Evalúa la ecuación actual y muestra el resultado.
7.0.9.2 Maneja errores como división por cero y errores de sintaxis.
7.0.9.3 Actualiza el historial local y en Firebase con el resultado de la operación.

### 7.0.10. Método ecuacion_grados_a_radianes

```
def ecuacion_grados_a_radianes(self, ecuacion):
    funciones_trigonometricas = ['sin', 'cos', 'tan', 'asin', 'acos', 'atan']
    for funcion in funciones_trigonometricas:
        ecuacion = ecuacion.replace(f"{funcion}(", f"math.{funcion}(math.radians(")
    return ecuacion
```

7.0.10 Convierte funciones trigonométricas de grados a radianes si el modo de ángulo está en grados.

### 7.0.11. Método reemplazar_raices_cuadradas

```
def reemplazar_raices_cuadradas(self, ecuacion):
    patron = re.compile(r'√\[(\d+)\]\(([^)]+)\)')
    while '√' in ecuacion:
        coincidencia = patron.search(ecuacion)
        if not coincidencia:
            break
        n = int(coincidencia.group(1))
        x = coincidencia.group(2)
        reemplazo = f'({x})**(1/{n})'
        ecuacion = ecuacion[:coincidencia.start()] + reemplazo + ecuacion[coincidencia.end():]
    return ecuacion
```

7.0.11 Convierte las raíces cuadradas en una notación compatible con Python para evaluación.

### 7.0.12. Método manejar_porcentajes

```
def manejar_porcentajes(self, ecuacion):
    patron = re.compile(r'(\d+(\.\d+)?)%(\d+(\.\d+)?)')
    coincidencias = patron.findall(ecuacion)

    for coincidencia in coincidencias:
        porcentaje = float(coincidencia[0]) / 100
        numero = coincidencia[2]
        reemplazo = f'({porcentaje}*{numero})'
        coincidencia_completa = coincidencia[0] + '%' + coincidencia[2]
        ecuacion = ecuacion.replace(coincidencia_completa, reemplazo)

    return ecuacion
```

7.0.12 Convierte expresiones porcentuales en una forma evaluable por Python.

### 7.0.13. Método agregar_al_historial

```
def agregar_al_historial(self, operacion):
    self.historial.insert(0, operacion)  # Insertar al principio de la lista
    if len(self.historial) > 30:
        self.historial.pop()  # Eliminar el último elemento si el tamaño es mayor a 30
```

7.0.13.1 Añade el cálculo realizado al historial local y en Firebase.
7.0.13.2 Actualiza el panel de historial de la interfaz gráfica.

**7.0.14. Método actualizar_panel_historial.**

```
def actualizar_panel_historial(self, historial):
    self.panel_historial.delete(0, tk.END)
    historial = list(reversed(historial))  # Reversar para que el más nuevo aparezca arriba
    for item in historial:
        self.panel_historial.insert(tk.END, item)
```

7.0.14.1 Actualiza el panel de historial en la interfaz gráfica con las últimas entradas.

**7.0.15. Bloque if __name__ == "__main__"**

```
if __name__ == "__main__":
    # Crear la ventana principal
    root = tk.Tk()

    # Iniciar la ventana de inicio
    app = VentanaDeInicio(root)

    # Ejecutar el bucle principal de la interfaz
    root.mainloop()
```

7.0.15.1 Crea la ventana principal y una instancia de CalculadoraBasica, luego inicia el bucle principal de la interfaz gráfica.

## 7.1 Introducción:

Una simple calculadora gráfica hecha con python usando las diferentes librerías ya mencionadas, para el funcionamiento y gestión de esta, cálculos desde 1+1 hasta el coseno de pi elevado a seno de 1000.

### 7.1.1 Propósito:

Este proyecto ha sido diseñado con el fin de que se brinde un alto rendimiento y fácil acceso a aquellos usuarios que necesiten hacer cálculos trigonométricos, operaciones algebraicas, entre otros grandes problemas matemáticos.
Buscamos una optimización con firebase y database, actualizando los datos del historial constantemente y fabricando en sí un gran apoyo visual.

### 7.1.2 descripción general

Está principalmente desarrollado para hacer cálculos y procesos gráficos, que tengan que ver con la parte matemática, dirigiendo nuestro proyecto ante cualquier persona que necesite una calculadora básica, científica y que muestre gráficas de la solución.

Modelo (model_defs_calculadora.py, model_cal_grafica.py):

**Propósito:** Gestiona los datos y la lógica de la aplicación. Realiza los cálculos, manipula los datos y mantiene el estado de la aplicación.
Clases:
**ModeloCalculadora:** Maneja las operaciones matemáticas básicas y avanzadas de la calculadora, como el cálculo de resultados y la gestión del historial.
**ModeloGraficadora:** Preprocesa las ecuaciones, genera gráficos y evalúa expresiones matemáticas para la parte gráfica.
Vista (vista_cal_basica.py, vista_cal_grafica.py, vista_ventana_de_inicio.py):

**Propósito:** Se encarga de la presentación de la interfaz de usuario. Muestra los datos del modelo y envía las solicitudes de interacción del usuario al controlador.
**Clases:**
**VistaCalculadora:** Crea y gestiona la interfaz de usuario para la calculadora básica.
**VistaGraficadora:** Maneja la visualización de gráficos y resultados en la interfaz de la graficadora.
**VentanaDeInicio:** Muestra la ventana inicial y permite a los usuarios iniciar la calculadora o cualquier otra funcionalidad relacionada.
Controlador (control_calculadora.py, control_cal_grafica.py, control_firebase.py):

**Propósito:** Actúa como intermediario entre el modelo y la vista. Maneja las interacciones del usuario, actualiza el modelo en consecuencia y actualiza la vista para reflejar los cambios en el modelo.
**Clases:**

**ControladorCalculadora:** Controla la lógica de la calculadora, maneja los clics en los botones y cambia entre los modos básico y científico.
**ControladorGraficadora:** Controla la lógica de la graficadora, maneja la generación de gráficos y el manejo de paneles.
**ControladorFirebase:** Maneja la integración con Firebase para guardar y recuperar el historial de operaciones.
Flujo de Trabajo
Inicio de la Aplicación:

main.py inicia la aplicación creando la ventana principal y lanzando la vista de inicio (VentanaDeInicio).

### 7.1.3 **Personal involucrado**:

**En el desarrollo de este programa de calculadora gráfica nuestro personal fue:**

- Byron Pedraza
- Juan Manuel Toro Rojas
- Manuel Alejandro Velasco Marinez

### **7.2.2 Funcionalidad del producto**:

### 7.2.4 Restricciones

# *8. Diseño de la Arquitectura*

1. El Modelo (Model) es responsable de la lógica de la aplicación y la manipulación de datos. En el caso de la calculadora, el modelo gestiona la lógica de cálculo y el almacenamiento del historial.

#### **8.0.1 Lógica de Cálculo:**

**Métodos como :**

- calcular_resultado.
- ecuacion_grados_a_radianes,
- reemplazar_raices_cuadradas,
- manejar_porcentajes realizan cálculos
- transformaciones en la ecuación ingresada por el usuario.

- El modelo también maneja la lógica para convertir grados a radianes, calcular raíces y porcentajes, y manejar errores matemáticos.

**8.0.2 Historial:**

8.0.2.1 Métodos como agregar_al_historial, actualizar_panel_historial, y las funciones de Firebase (actualizar_historial_firestore, leer_historial_firestore, actualizar_historial_realtime) gestionan el almacenamiento y la recuperación del historial de cálculos.

2. La Vista (View) es responsable de presentar la interfaz gráfica al usuario y mostrar los resultados de las operaciones. En el código de la calculadora, la vista está compuesta por:

**8.0.2. Interfaz Gráfica:**

8.0.2.1.1 tkinter se usa para crear la ventana principal y sus componentes (botones, campos de texto, paneles).
8.0.2.1.2 Métodos como crear_interfaz, modo_basico, modo_cientifico, y crear_botones configuran y actualizan la interfaz gráfica según el modo seleccionado.

8.0.2.2 Panel de Historial:
8.0.2.2.1 El Listbox en self.panel_historial muestra el historial de cálculos y resultados.
8.0.2.2.2 Actualizar Panel Historial actualiza este panel con las últimas operaciones realizadas.

**8.0.3. El Controlador (Controller)** actúa como intermediario entre el modelo y la vista, gestionando las interacciones del usuario y actualizando el modelo y la vista en consecuencia. En el código:

**8.0.3.1 Interacción del Usuario:**
8.0.3.1.1 El método “al_hacer_clic_en_boton” maneja los eventos de clic en los botones, actualizando la ecuación y realizando cálculos cuando sea necesario.
8.0.3.1.2 lanzar graficadora maneja la apertura de una aplicación externa para graficar.

**8.0.3.2 Actualización de la Vista:**

8.0.3.2. El controlador actualiza la vista cuando se muestran resultados, se cambian modos o se actualiza el historial.

# *9. Instrucciones*

**1. Diagrama del Modelo**

Inicio

1. El sistema recibe una solicitud de operación (operación básica/científica o graficado).

Evaluación de Operación

- Verifica si la operación es básica o científica.
- Si es básica, utiliza math.
- Si es científica, utiliza sympy.
- Actualiza el historial si es necesario.

Evaluación de Ecuaciones

- Convierte la ecuación a un formato adecuado.
- Genera un gráfico con matplotlib y numpy.

Fin

- Devuelve el resultado o gráfico al controlador.

**2. Diagrama de la Vista**

Inicio

- El usuario interactúa con la interfaz gráfica (selecciona operación o ingresa datos).

Vista Básica

- Muestra botones para operaciones básicas.
- Permite la entrada de datos.
- Actualiza el historial visualmente.

Vista Gráfica

- Permite ingresar funciones matemáticas.
- Visualiza los gráficos generados.

Ventana de Inicio

- Presenta opciones para los distintos modos de calculadora (básico, científico, gráfico).

Fin

- Muestra el resultado o gráfico generado.

**3. Diagrama del Controlador**

Inicio

- El controlador recibe la entrada del usuario.

Controlador Calculadora

- Pasa las entradas al modelo.
- Actualiza el historial y muestra el resultado en la vista.

Controlador Gráfico

- Recibe la función para graficar.
- Coordina con el modelo y actualiza la vista gráfica.

Fin

- El usuario visualiza el resultado o gráfico.

**4. Modelo model_defs_calculadora.py**

Inicio

- Recibe una solicitud de cálculo o resolución de ecuación.

Evaluación de Operación

- Verifica si la ecuación está vacía.
- Realiza el cálculo usando math.

- Convierte funciones trigonométricas a radianes si es necesario.
- Calcula factorial, permutaciones o combinaciones si es necesario.

Evaluación de Ecuaciones Lineales

- Verifica si hay incógnitas.
- Resuelve ecuaciones con sympy.

Manejo de Excepciones

- Gestiona errores como división por cero o errores de sintaxis.

Actualización del Historial

- Almacena la operación en el historial, con un máximo de 30 operaciones.

Fin

- Devuelve el resultado al controlador.

**5. Modelo model_cal_grafica.py**

Inicio

- Entrada del usuario (ecuación o parámetros para graficar).

Graficar Función

- Preprocesa la ecuación.
- Verifica si contiene la variable ‘x’.
- Genera los valores y gráficos con matplotlib.

Evaluar Expresión

- Preprocesa la expresión.
- Evalúa usando sympy y devuelve el resultado numérico.

Graficar Círculo

- Valida los parámetros del círculo.
- Genera las coordenadas y grafica con matplotlib.

Fin

- Muestra el gráfico o resultado.

**6. Vista vista_cal_basica.py**

Inicialización de la Vista

1. Se instancia la clase VistaCalculadora.
2. Se crean los widgets de la interfaz.
3. Se muestra la ventana principal.

Interacción de Botones

- El usuario hace clic en un botón.
- El texto se pasa al controlador.
- Se actualiza la vista según el resultado.

Mostrar Opciones de Modo

- El usuario cambia el modo de calculadora.
- Se actualiza la vista para mostrar el nuevo modo.

Actualización del Historial

- Se actualiza el panel de historial tras una operación.

**7. Vista vista_cal_grafica.py**

Inicio

- Lanzamiento de la aplicación.

Inicialización

- Configura la ventana principal.
- Crea el botón “Agregar Panel”.

Interacción del Usuario

- El usuario agrega un panel y define una función para graficar.

Procesamiento de Funciones

- El usuario ingresa una función.
- Evalúa la función o la gráfica.

Actualizar Gráfico

- Actualiza el gráfico según los valores ingresados.

Limpiar Gráfico

- El usuario puede limpiar la gráfica o eliminar paneles.

Fin

- El gráfico y la interfaz están actualizados.

**8. Vista vista_ventana_de_inicio.py**

Inicio

- Se muestra la ventana de inicio.

Interacción del Usuario

- El usuario ingresa un nombre y presiona “Continuar”.

Decisión: ¿Nombre Vacío?

- Si está vacío, muestra un error.
- Si no, abre la calculadora básica.

Fin

- La ventana de inicio se cierra y se abre la calculadora.

**9. Controlador control_calculadora.py**

Inicio

- Se inicia la ventana principal.

Interacción del Usuario

- El usuario presiona botones y realiza operaciones.

Decisión

- Resuelve ecuaciones, cambia modos, o grafica funciones.

Fin

- La aplicación sigue corriendo hasta que el usuario cierre la ventana.

**10. Controlador control_cal_grafica.py**

Inicio

- Se crea la ventana principal.

Graficar Función

- El usuario ingresa una ecuación para graficar.

Limpiar Entrada

- El usuario limpia la entrada.

Agregar Nuevo Panel

- El usuario agrega un panel de funciones.

Fin

- El proceso sigue hasta que el usuario cierra la ventana.

**11. Controlador control_firebase.py**

Inicio

- Se verifica la conexión a Firebase.

Actualizar Firebase

- Actualiza el historial en Firebase, manteniendo un máximo de 90 operaciones.

Sincronización

- Sincroniza el historial y lo actualiza en la vista.

Fin

- El proceso termina y el historial está actualizado.

**12. Main main.py**

Inicio

- Se ejecuta el archivo main.py.

Inicialización

- Crea la ventana principal.

Mostrar Ventana de Inicio

- Se inicializa y muestra la ventana de inicio.

Fin

- La aplicación sigue activa hasta que se cierra la ventana.