MANUAL DE TECNICO

Proyecto No. 1

LENGUAJES FORMALES Y DE PROGRAMACIÓN A+

José Eduardo Galdámez González
Carne: 202109732

MANUAL DE TECNICO:

El siguiente programa fue creado desde el lenguaje de Python, con lo que fue la programación orientada en objetos (*POO*) en este caso se trabajo con interfases graficas para que la vista del usuario se dinámica. El objetivo al hacer el proyecto es la aplicación de *POO*, manejo de archivos y lo que es analizador léxico basado en lo que son *diagramas de árbol* junto a sus *AFD's,* para el planteamiento de este programa hubo un análisis previo el cual iremos explicando el cómo se planteó en base a estos diagramas y como después se aplico dentro de lo que es la estructura principal del código.

Antes de explicar a detalle la estructuración primero empezaremos con las bases dentro de nuestro código a nivel general:

Apartado Grafico:

Para el apartado grafico se utilizó *Tkinter* que es una librería del lenguaje de programación Python y funciona para la creación y el desarrollo de aplicaciones de escritorio. Esta librería facilita el posicionamiento y desarrollo de una interfaz gráfica de escritorio con Python.

Para poder trabajar con esa librería hicimos el importe a nuestro código:

```
from tkinter import *
```

Estructuración de código:

Ya sabiendo que se trabajó con la librería *Tkinter*, damos paso a como se inicializo el programa y su estructura.

Para comenzar tenemos lo que es el importe de todas librerías que se trabajaron:

```
#IMPORTE DE LIBRERIAS
import math
from tkinter import filedialog, ttk
from tkinter import *
import tkinter.font as tkFont
from tkinter import messagebox
import tkinter as tk
import webbrowser as wb
```

Junto con están las variables que se trabajaron de forma global dentro de nuestro programa.

```
#VARIABLES
global open_status_name
open_status_name=False
data=[]
operaciones=[]
resultados=[]
errores=[]
lineaError=[]
contLinea=0
contOpe=0
contTexto=0
fin=0
```

Ya con esto nuestra forma de diseño fue en base a una Clase Principal la cual tendrá como función almacenar el inicializador de la aplicación junto a su ventana principal con todas sus funciones de la aplicación.

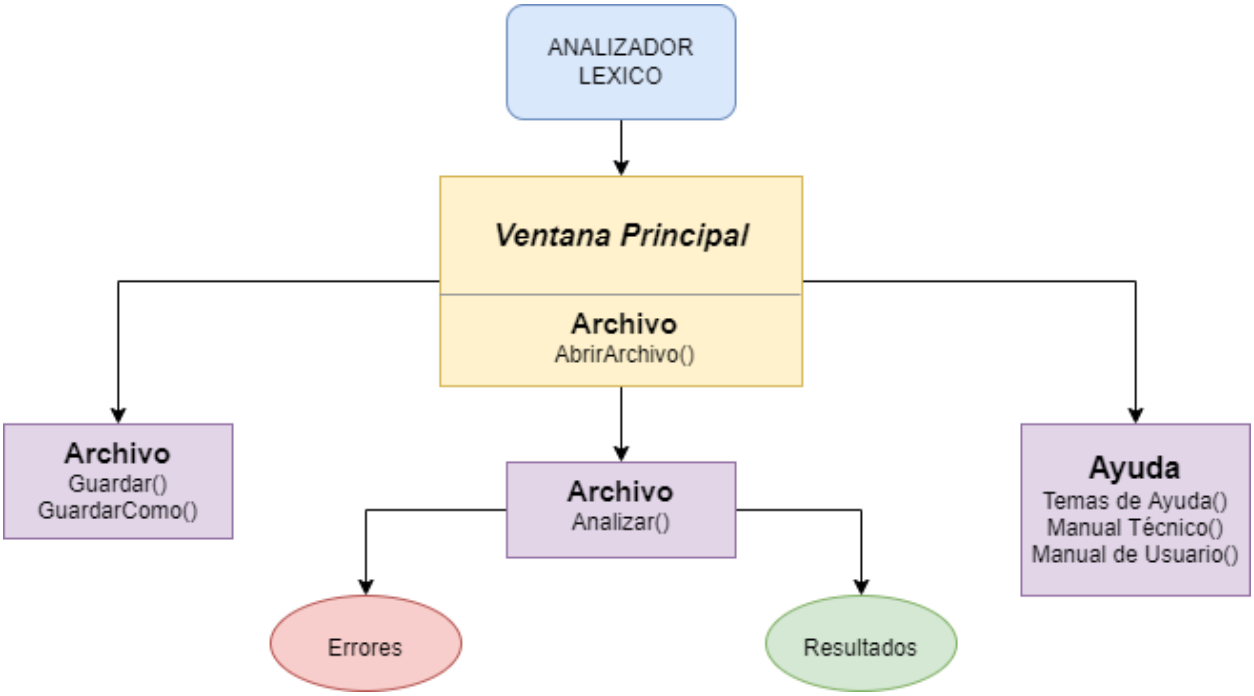
Clase Principal

```
#CLASE PARA EL PROGRAMA
class Main:
```

Inicializador de la App

```
if __name__ == '__main__':
    window=Tk()
    app=Main(window)
    window.mainloop()
```

Con esta estructura ya dimos paso a lo que es el diseño con sus respectivas funciones del programa, primero entendamos como se trabajo y para esto hicimos el siguiente diagrama de clases.



Ya teniendo nuestra estructura principal de cómo se manejará la parte funcional y grafica empezamos con el diseño de la interfaz de usuario colocando en los botones correspondientes las funciones que tendrán al ser presionados.

```
#MAIN VENTANA
def __init__(self, window):
    self.wind=window
    window.title('Menu Principal')
    width=800
    height=500
    window.config(bg = "#88cffa")
    screenwidth = self.wind.winfo_screenwidth()
    screenheight = self.wind.winfo_screenheight()
    alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)
    self.wind.geometry(alignstr)
    self.wind.resizable(width=False, height=False)
    frame=Frame(window, bg = "#88cffa")
    frame.pack(pady=4)
    self.texto = Text(frame, width=110, height=30, font=("Comic Sans MS",18), selectbackground="white", selectforeground="black")
    self.texto.pack(padx=10, pady=10)
    barra_scroll=Scrollbar(frame)
    barra_scroll.pack(side=RIGHT, fill=Y)
    barra_scroll.config(command=self.texto.yview)
    barra_navegar=tk.Menu()
    archivo=Menu(barra_navegar)
    barra_navegar.add_cascade(label="Archivo", menu=archivo)
    archivo.add_command(label="Abrir", command=self.abrirArchivo)
    archivo.add_command(label="Guardar", command=self.guardar)
    archivo.add_command(label="Guardar Como...", command=self.guardarComo)
    archivo.add_command(label="Analizar", command=self.analizar)
    archivo.add_command(label="Errores", command=self.abrirErrores)
    archivo.add_command(label="Resultados", command=self.abrirRes)
    archivo.add_separator()
    archivo.add_command(label="Salir", command=self.salirPrograma)
    ayuda=Menu(barra_navegar)
    barra_navegar.add_cascade(label="Ayuda", menu=ayuda)
    ayuda.add_command(label="Manual Tecnico")
    ayuda.add_command(label="Manual de Usuario")
    ayuda.add_command(label="Temas de ayuda", command=self.ayuda)
    window.config(menu=barra_navegar)
```

Funciones:

Para empezar de primero esta el abrir archivo el cual obtendrá el archivo y colocará la data en el texto, para que lo podemos visualizar la información con esto podremos editarlo y guardarlo ya sea como un nuevo archivo o lo reescribiremos.

```
#OBTENCION DE ARCHIVOS Y GUARDADO
def abrirArchivo(self):
    global open_status_name
    try:
        self.texto.delete("1.0", END)
        a=filedialog.askopenfilename(title='Abrir Archivo', filetypes=(("Text Files","*.txt"),("HTML Files","*.hmtl"),("Python Files","*.py"),("XML Files",("*.xml"))
        self.archivo=open(a, "r",encoding="utf-8")
        stuff=self.archivo.read()
        open_status_name=a
        self.texto.insert(END, stuff)
        self.archivo.close()
        messagebox.showinfo(title="Explorador de archivos", message="El archivo fue encontrado.")
    except:
        messagebox.showerror(title="Explorador de archivos", message="Hubo un error al obtener el archivo")
def guardarComo(self):
    global open_status_name
    self.archivo=filedialog.asksaveasfilename(defaultextension=".*", title="Guardar Archivo Como", filetypes=(("Text Files","*.txt"),("HTML Files","*.hmtl"),("Python
    open_status_name=self.archivo
    self.archivo=open(self.archivo,'w',encoding="utf-8")
    self.archivo.write(self.texto.get(1.0, END))
    self.archivo.close()
    messagebox.showinfo(title="Guardar", message="El archivo fue guardado.")
def guardar(self):

    global open_status_name
    res = messagebox.askquestion('Guardar archivo', '¿Desea guardar este archivo?')
    if res=='yes':
        self.archivo=open(open_status_name,'w',encoding="utf-8")
        self.archivo.write(self.texto.get(1.0, END))
        self.archivo.close()
        messagebox.showinfo(title="Guardar", message="El archivo fue guardado.")
    else:
        messagebox.showerror(title="Guardar", message="El archivo no fue guardado.")
```

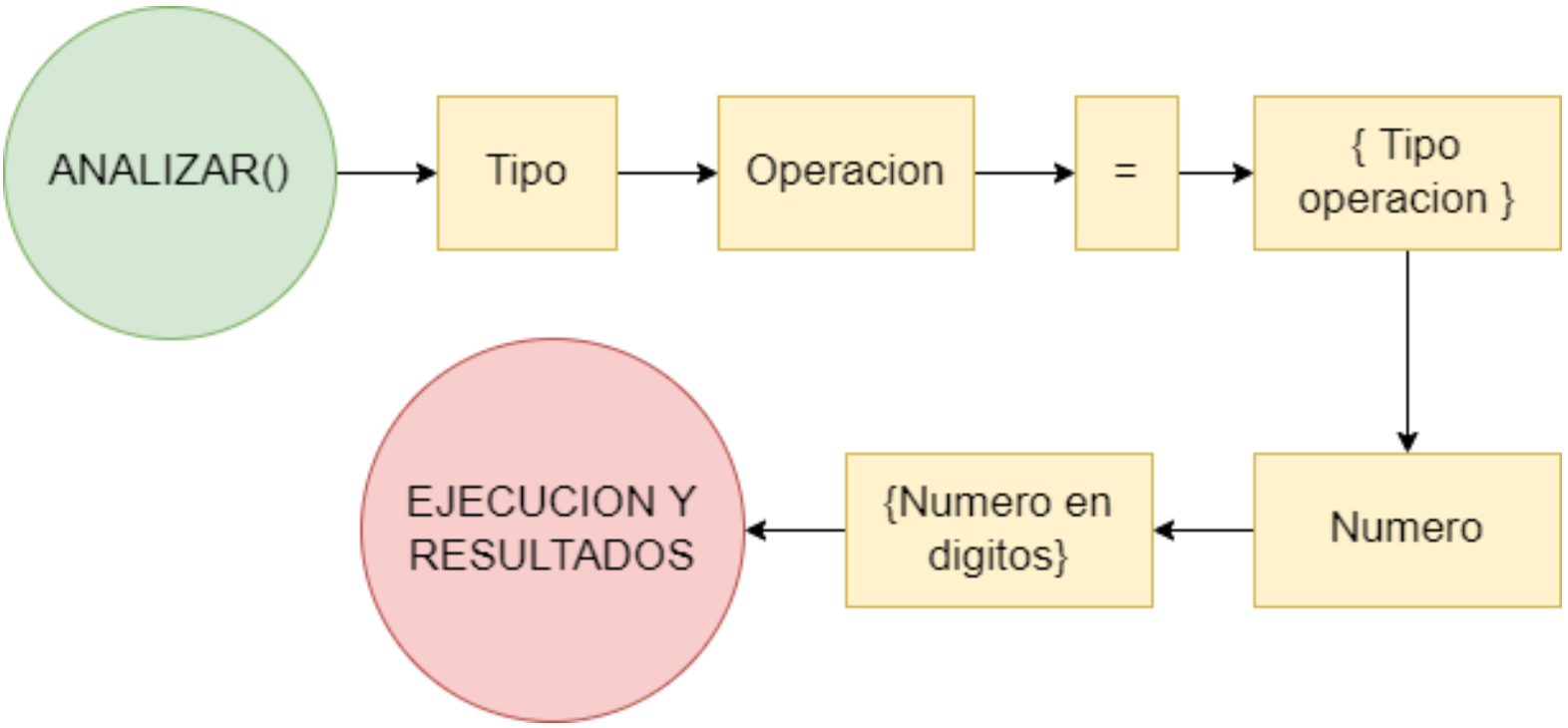
Por otra parte, tenemos el apartado donde las funciones serán encargadas de abrir los resultados después de analizar o los errores, también tendremos acceso a los manuales técnicos y de usuario.

```
#BARRA AYUDA
def ayuda(self):
      messagebox.showinfo(title="PROGRAMA ANALZADOR LEXICO", message="El siguiente programa consiste en analizar el archivo de cualquier tipo y detectar si tiene la
def abrirErrores(self):
    try:
        wb.open_new_tab('Error.html')
    except:
        messagebox.showerror(title="Errores", message="El archivo no ha sido analizado o no se encotraron errores.")
def abrirRes(self):
    try:
        wb.open_new_tab('Resultados.html')
    except:
        messagebox.showerror(title="Resultados", message="El archivo no ha sido analizado o se han encotrado errores.")

#ABRIR MANUALES
def abrirMT(self):
    wb.open_new('ManualTecnico.pdf')
def abrirMU(self):
    wb.open_new('ManualDeUsuario.pdf')
```

Finalmente encontramos el analizar, este lo dejamos aparte ya que es esta parte es la función principal de la App, ya que este se basó en lo que es un analizador léxico en base a diagramas de árbol junto sus *AFD's.* Para empezar se llama a la función la cual obtendrá la información del texto y a partir de ahí la guardara y la analizara siguiendo estados.

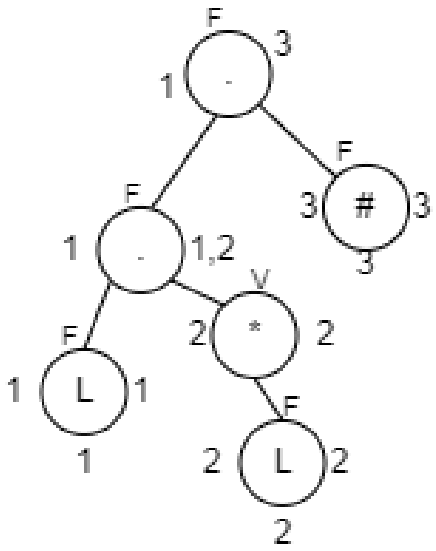
```
#ANALIZAR
def analizar(self):
    global contLinea
    global fin
    fin=0
    try:
        contLinea=0
        data.clear()
        operaciones.clear()
        resultados.clear()
        errores.clear()
        lineaError.clear()
        analizar=open(open_status_name,"r",encoding="utf-8")
        lineas=analizar.readlines()
        for linea in lineas:
            linea=linea.replace(" ", "")
            linea=linea.replace("\n", "")
            linea=linea.replace("\t", "")
            data.append(linea)

        self.tipo()
    except:
        print("TERMINADO")
```

Ya con la data obtenida da inicio a nuestra serie de algoritmos que seguirán los pasos de nuestros diagramas de árbol y *AFD´s,* pero antes dejaremos el cómo se manejaran nuestros *ESTADOS* para que vaya analizando los datos del archivo.

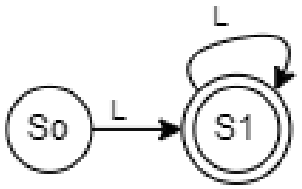


Estado Tipo:

LL*



i	SIG(i)
1 L	2,3
2 L	2,3
3 #1	

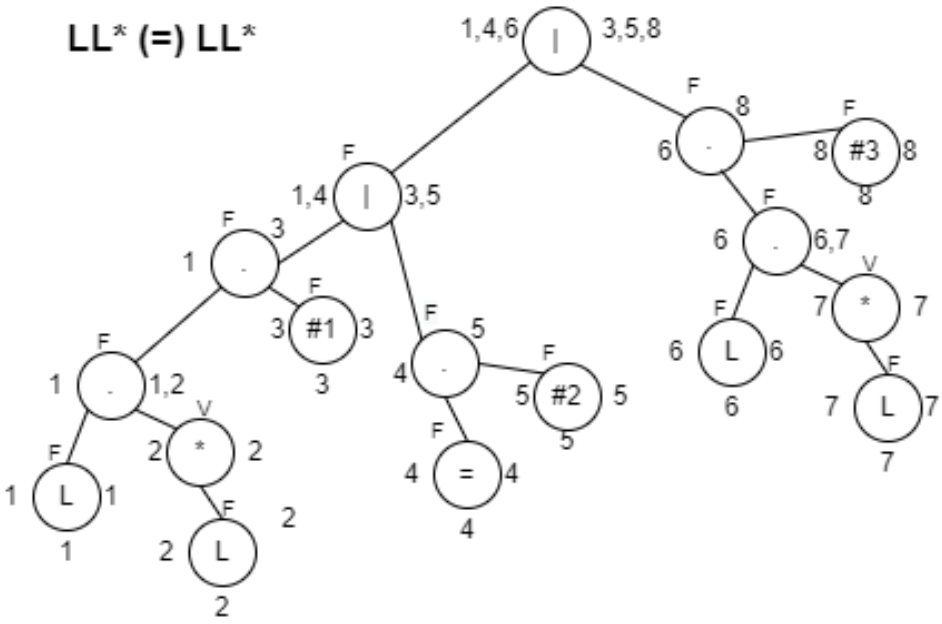


En este caso podemos ver que lo que hará nuestra función será solo ira viendo si cumple que venga L (letra) seguido de L (letra).

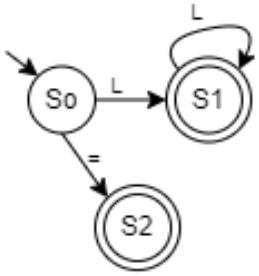
Ya en código ira guardando cada dato que cumpla esta condición según nuestro árbol, de LL*, y después comparara si cumple con la condición de ser Tipo y que avance con línea y el siguiente estado, si en dado caso no cumple este generará un error y se detendrá el análisis y con eso mandra la función de reporte de error para especificar que error tuvo el archivo.

```
def tipo(self):
    global contLinea
    cadena_igual=[]
    elemento=""
    for elem in data[contLinea]:
        cadena_igual.append(elem)
    for j in range(len(cadena_igual)):
        if cadena_igual[j].isalpha():
                elemento=elemento+str(cadena_igual[j])

    if elemento=="Tipo":
        contLinea=contLinea+1
        self.ope()
    else:
        errores.append("No se encontro Tipo.")
        lineaError.append(contLinea+1)
        self.reporteErro()
```

Estado Operación, Igual y Tipo Operación:



i	SIG(i)
1 L	2,3
2 L	2,3
3 #1	
4 =	5
5 #2	
6 L	7,8
7 L	7,8
8 #3	



En este lo hicimos juntos ya que estos estarán en una misma línea y para que vaya avanzando ira primero L (letra) seguido de L (letra), ya cumplida esa parte avanzara y obtendrá el igual (=) para finalmente volver a L (letra) seguido de L (letra).

Ya en código se basará en esta dinámica, en lo único que hará será ir cumpliendo la validación, por ejemplo:

Operación → Correcto ***avanza***

= → Correcto ***avanza***

SUMA → Correcto ***salto de línea***

```
def ope(self):
    global contLinea
    cadena_ope=[]
    contLinea
    contCade=0
    cadena_ope.clear()
    elemento=""
    elemento2=""
    global fin
    for elem in data[contLinea]:
        cadena_ope.append(str(elem))

    for j in range(len(cadena_ope)):
        if cadena_ope[j].isalpha() and contCade<9:
            elemento=elemento+str(cadena_ope[j])
            contCade=contCade+1
    for j in range(len(cadena_ope)):
        if cadena_ope[j].isalpha():
            elemento2=elemento2+str(cadena_ope[j])
            contCade=contCade+1
```

```
if elemento=="Tipo":
    contLinea=1+contLinea
    fin=1
    self.generarHtml()
if elemento=="Operacion" and fin<1:
    self.igual()
if elemento!="Operacion" and fin<1:
    errores.append("No se encontro Operacion.")
    lineaError.append(contLinea+1)
    self.reporteErro()
```

*** Avanza ***

```
def igual(self):
    global contLinea
    cadena_igual=[]
    contCade=0
    elemento=""
    for elem in data[contLinea]:
        cadena_igual.append(elem)
    for j in range(len(cadena_igual)):
        if cadena_igual[j].isalpha() and contCade<9:
                contCade=contCade+1
        if contCade==9:
            elemento=cadena_igual[j+1]
            contCade=contCade+1


    if elemento=='=':
        self.tipoOpe()
    else:
        errores.append("No se encontro =.")
        lineaError.append(contLinea+1)
        self.reporteErro()
```

*** Avanza ***

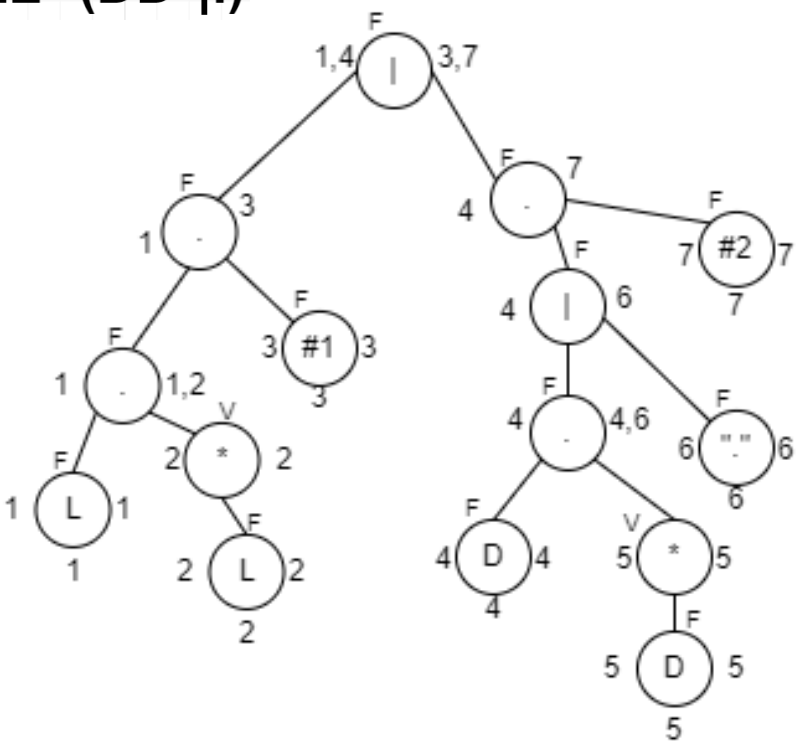
```
def tipoOpe(self):
    global contLinea
    cadena_tipoOpe=[]
    contCade=0
    elemento=""
    for elem in data[contLinea]:
        cadena_tipoOpe.append(elem)
    for j in range(len(cadena_tipoOpe)):
        if cadena_tipoOpe[j].isalpha():
            contCade=contCade+1
            if contCade>9:
                elemento=elemento+str(cadena_tipoOpe[j])


    if elemento=="SUMA":
        contLinea=contLinea+1
        self.numSuma()
    if elemento=="RESTA":
        contLinea=contLinea+1
        self.numResta()
    if elemento=="MULTIPLICACION":
        contLinea=contLinea+1
        self.numMulti()
    if elemento=="DIVISION":
        contLinea=contLinea+1
        self.numDiv()
    if elemento=="POTENCIA":
        contLinea=contLinea+1
        self.numPot()
    if elemento=="RAIZ":
        contLinea=contLinea+1
        self.numRAIZ()
    if elemento=="INVERSO":
        contLinea=contLinea+1
        self.numInv()
```

```
if elemento=="SENO":
    contLinea=contLinea+1
    self.numSen()
if elemento=="COSENO":
    contLinea=contLinea+1
    self.numCos()
if elemento=="TANGENTE":
    contLinea=contLinea+1
    self.numTan()
if elemento=="MOD":
    contLinea=contLinea+1
    self.numMod()
else:
    if fin<1:
        errores.append("No se encontro ningun tipo de operación valida. (SUMA,RESTA,DIVISION...)")
        lineaError.append(contLinea+1)
        self.reporteErro()
```

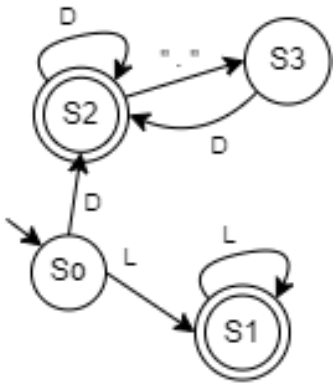
*** Salto de línea ***

Estado Numero:

LL* (DD*|.)



i	SIG(i)
1 L	2,3
2 L	2,3
3 #1	
4 D	4,7
5 D	4,7
6 .	7
7 #2	



En este lo hicimos juntos ya que estos estarán en una misma línea y para que vaya avanzando ira primero L (letra) seguido de L (letra), ya cumplida esa parte avanzara a D (digito) seguido de D (digito) o revisar si está el punto decimal (.) y finalmente volver a D (digito) seguido de D (digito).

Ya en código se basará en esta dinámica, en lo único que hará será ir cumpliendo la validación (también hay posibilidad que no lleve punto decimal si es entero), este parara hasta que no haya números y se cierre la operación, dando como terminado esa operación y avanzara de línea para ver si hay más operaciones, por ejemplo:

Numero → Correcto ***avanza***

25 → Correcto ***avanza***

. → Correcto ***avanza***

50 → Correcto ***salto de línea**

```
def numSuma(self):
    global contLinea
    cadena_num=[]
    cadena_Res=""
    contCade=0
    contCierre=0
    elemento=""
    stop=False
    cierre=""
    k=0
    num=0
    a=0
    ope=0
    contCierre=1
    contOp=0
```

```
while stop!=True:
    contCade=0
    cadena_num.clear()
    elemento=""
    elemento2=""
    cierre=""
    num=0
    a=""
    for elem in data[contLinea]:
        cadena_num.append(elem)

    for j in range(len(cadena_num)):
        if cadena_num[j].isalpha() and contCade<6:
            contCade=contCade+1
            elemento=elemento+str(cadena_num[j])
        if contCade<10:
            cierre=cierre+str(cadena_num[j])
        if cadena_num[j].isalpha():
            elemento2=elemento2+str(cadena_num[j])
```

```
if elemento=="Numero":
    for k in range(len(cadena_num)):
        if cadena_num[k].isdigit() or cadena_num[k]==".":
            try:
                a=a+str(cadena_num[k])
                num=float(a)
            except:
                errores.append("No es valido los digitos de la operacion.")
                lineaError.append(contLinea+1)
                self.reporteErro()

if elemento=="Numero":
    contOp=1+contOp
    ope=ope+num

    contLinea=contLinea+1
    if contOp==1:
        cadena_Res=str(num)
    else:
        cadena_Res=cadena_Res+" + "+str(num)
```

```
    if elemento!="Numero" and cierre!="</Operacion>":
        errores.append("La escritura de Numero incorrecte o cierre de operacion no valido.")
        lineaError.append(contLinea+1)
        self.reporteErro()
        stop=True
    if cierre=="</Operacion>":
        contLinea=contLinea+1
        contCierre=contCierre-1
    if contCierre==0:
        stop=True
if cierre=="</Operacion>":
    operaciones.append(ope)
    cadena_Res=cadena_Res+" = "+str(ope)
    print(cadena_Res)
    resultados.append(cadena_Res)
    self.ope()
```

Resultados y Errores:

Finalmente, con la lógica del análisis vamos a lo que son los resultados o errores generados en HTML, como ya vimos en el código puede ir avanzando y ejecutando sus estados o simplemente se quedara en el error de línea.

Reporte de Operaciones: si en dado caso todo fue bien este nos avisará con una alerta y generará el reporte de resultados y lo que hará es que ya guardadas todas las operaciones las imprimirá en el archivo HTML.

```
if elemento=="Tipo":
    contLinea=1+contLinea
    fin=1
    self.generarHtml()
```

```
def generarHtml(self):
    messagebox.showinfo(title="ARCHIVO ANALIZADO", message="Se ha ejecutado las operaciones y el reporte de operaciones en un HTML EN LA CARPETA.")
    archi1=open("Resultados.html","w")
    archi1.write("<!DOCTYPE html>\n")
    archi1.write('<html lang="en">\n')
    archi1.write('<head>\n')
    archi1.write('<meta http-equiv="X-UA-Compatible" content="IE=edge">\n')
    archi1.write('<meta name="viewport" content="width=device-width, initial-scale=1.0">\n')
    archi1.write('<title>RESULTADOS</title>\n')
    archi1.write('<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-iYQeCzEYFbKjA/T2uDLTpkwGzCi
    archi1.write('<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-u1OknCvxWvY5kfmNBILK2hRnQC3Pr17a+RTT6r
    archi1.write('</head>\n')
    archi1.write('<body>\n')
    archi1.write('<h1 class="mx-auto p-3" style="width: 700px;">RESULTADOS DE OPERACIONES.</h1> \n')
    archi1.write('<br>\n')
    archi1.write('<div class="container">\n')
    for i in range(len(resultados)):
        archi1.write('<div class="container">\n')
        archi1.write('<label class="mb-3"> <b>Operacion No. '+str(i+1)+':'+'</b></label>\n')
        archi1.write('<div>\n')
        archi1.write('<label class="mb-3">'+resultados[i]+'</label>\n')
        archi1.write('</div>\n')
        archi1.write('</div>\n')
        archi1.write('<br>\n')
    archi1.write('</div>\n')
    archi1.write('</body>\n')
    archi1.write('</html>\n')
    archi1.close()
```

Reporte de Errores: si en dado caso no funciono este nos avisará con una alerta y generará el reporte de errores y mostrará toda la información del error.

```
errores.append("No se encontro Tipo.")
lineaError.append(contLinea+1)
self.reporteErro()
```

```python
def reporteErro(self):
    messagebox.showerror(title="ERROR DETECTADO", message="Hubo error en la ejecucion de analizar el archivo, Reporte Generado.")
    archi1=open("Error.html","w")
    archi1.write("<!DOCTYPE html>\n")
    archi1.write('<html lang="en">\n')
    archi1.write('<head>\n')
    archi1.write('<meta http-equiv="X-UA-Compatible" content="IE=edge">\n')
    archi1.write('<meta name="viewport" content="width=device-width, initial-scale=1.0">\n')
    archi1.write('<title>RESULTADOS</title>\n')
    archi1.write('<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-iYQ
    archi1.write('<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-u1OknCvxWvY5kfi
    archi1.write('</head>\n')
    archi1.write('<body>\n')
    archi1.write('<h1 class="text-center p-3">ERROR DETECTADO.</h1> \n')
    archi1.write('<br>\n')
    archi1.write('<table class="table table-dark table-bordered table-striped">\n')
    archi1.write('<thead>\n')
    archi1.write('<tr>\n')
    archi1.write('<th scope="col">Descripción de error</th>\n')
    archi1.write('<th scope="col">Linea</th>\n')
    archi1.write('</tr>\n')
    archi1.write('</thead>\n')
    archi1.write('<tbody>\n')
    archi1.write('<tr>\n')
    archi1.write('<td>'+errores[0]+'</td>\n')
    archi1.write('<td>'+str(lineaError[0])+'</td> \n')
    archi1.write('</tr> \n')
    archi1.write('</tbody>\n')
    archi1.write('</table>\n')
    archi1.write('<div class="container">\n')
    archi1.write('</div>\n')
    archi1.write('</body>\n')
    archi1.write('</html>\n')
    archi1.close()
```

Planificación:

Todo este proceso que se llevó a cabo tomo tiempo para cada sección la cual se divide de la siguiente manera:

Tarea #1:

Base de Proyecto: esta parte se centró en la creación del menú y la obtención de la información del archivo y mostrarlo en la caja de texto. (Tiempo estimado de trabajo: 2 horas, Tiempo Real: 3 horas)

Tarea #2:

Diagrama de clases y de árbol con sus DFA's: esta parte trabajamos a nivel lógico lo que se iba a trabajar en el programa, dándole estructura a nuestro análisis de datos. (Tiempo estimado de trabajo: 5 horas, Tiempo Real: 4 horas)

Tarea #3:

Construcción de los estados del analizar: en este se trabajó la construcción para que vaya analizando línea por línea y que ejecute sus funciones. (Tiempo estimado de trabajo: 12 horas, Tiempo Real: 14 horas)

Tarea #4

Generación de resultados y reportes: en este se trabajó la construcción para que vaya analizando línea por línea y que ejecute sus funciones. (Tiempo estimado de trabajo: 5 horas, Tiempo Real: 3 horas)

Glosario:

Librerías: es un conjunto de archivos que se utiliza para desarrollar software. Suele estar compuesta de código y datos, y su fin es ser utilizada por otros programas de forma totalmente autónoma. Simple y llanamente, es un archivo importable.

HTML: El Lenguaje de Marcado de Hipertexto (HTML) es el código que se utiliza para estructurar y desplegar una página web y sus contenidos. Por ejemplo, sus contenidos podrían ser párrafos, una lista con viñetas, o imágenes y tablas de datos.

AFD: Un autómata finito no determinista (abreviado AFND) es un autómata finito que, a diferencia de los autómatas finitos deterministas (AFD), posee al menos un estado $q \in Q$, tal que para un símbolo $a \in \Sigma$ del alfabeto, existe más de una transición $\delta(q,a)$ posible.