Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Laboratorio de Lenguajes Formales y de Programación

Manual Técnico

Proyecto 1: Analizador léxico

Catedrático: Mario Josué Solis Solórzano

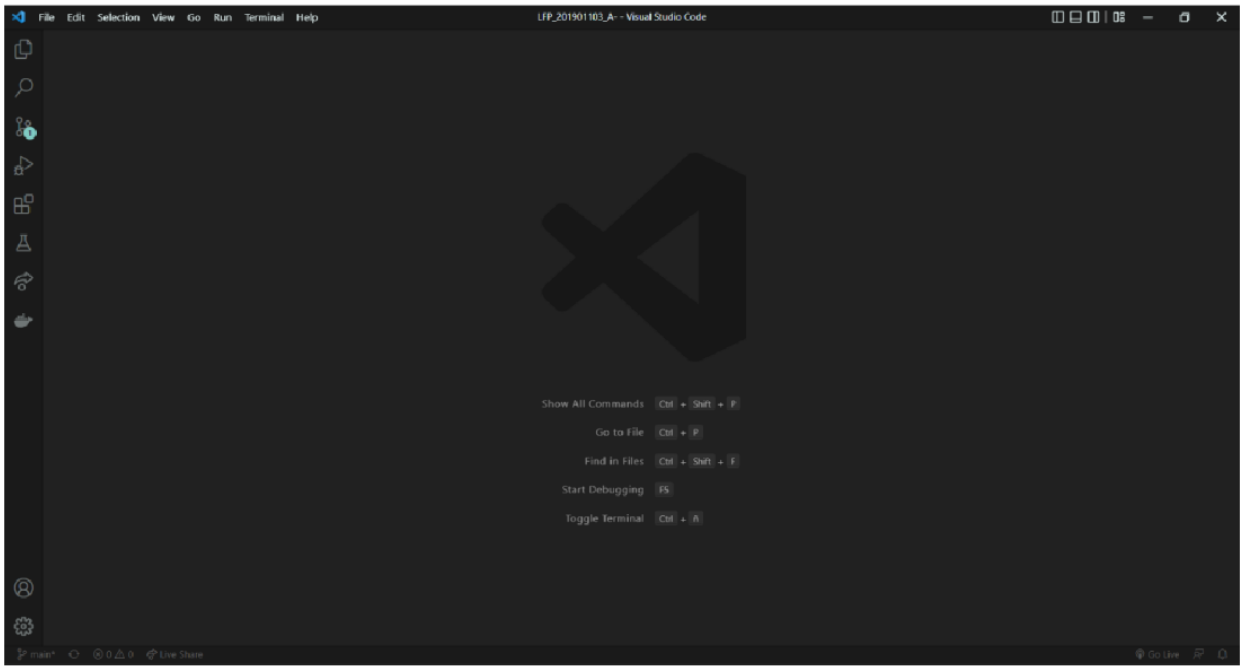
Sección: A-

Nombre: Josué Daniel Rojché García

Carné: 201901103

Requerimientos del Sistema

En la realización del software el Editor de texto que se utilizó fue Visual Studio Code.



El programa fue realizado con la versión de Python 3.11.1 para sistemas operativos de 64 bits.

Analizador Léxico

Para que el programa funcione correctamente se realizaron diversas validaciones y manejo de errores para que no finalice indebidamente durante la ejecución de esta. Por lo cual explicaremos lo que realiza el código.

Primero se importan las librerías necesarias, las cuales se utilizarán más adelante, por ejemplo: la librería tkinter, la cual servirá para las ventanas graficas

```
1  import os
2  import tkinter as tk
3  from tkinter import CENTER, RIGHT, Y, Scrollbar, filedialog, Tk, ttk
4  from tkinter.messagebox import showerror, showinfo, showwarning
5  import webbrowser
6  from automataAFD import AFD
```

Menú Principal

Para mostrar el menú principal se crea el main, el cual será el que se ejecute inicialmente, y comenzará por mostrar la pantalla principal.

La pantalla se crea con tk.Tk(), con title se le agrega un titulo a la pestaña, con geometry indicamos el tamaño que tendrá dicha ventana indicando el ancho por altura, con configure se indica el color de fondo que tendrá la ventana, resizable sirve para indicar que la ventana sea de tamaño fijo y que el usuario no pueda agrandarla o hacerla más pequeña.

Para crear una barra de menú se utiliza tk.Menu(), el cual sirve para agregarle los datos que sean necesarios, en este caso se crea el espacio para el manejo de los archivos por lo que se utiliza add_cascade() con sus respectivos parámetros, el primer parámetro indica que se agregará al menú de barra, y el label le indica el nombre que tendrá para esas opciones, para agregar las subopciones se utiliza add_command() el cual lleva como parámetros los siguientes, label el cual indica el nombre de la opción a utilizar, luego command al cual se le asigna el método al que accederá para realizar la acción que corresponda, luego con state se indica que la opción no estará disponible si no se carga un archivo con anterioridad.

Con tk.Text() se agrega dos áreas de texto que servirán para editar el archivo y mostrar los errores del archivo al momento de realizar las opciones que correspondan a partir que se cargue el archivo en el sistema, con configure() se le indica el color de fondo que tendrá con bg, y state para indicar que al principio no se pueda editar en el área de texto.

Con tk.label() se crea un texto en una línea que estará plasmado para indicar cual cuadro de texto es para el archivo cargado y el que muestra los errores.

Con config() se agrega el parámetro que agrega a la ventana la barra de menú y con mainloop() se indica que que la ventana será visible para el usuario.

```python
if __name__ == '__main__':
    menu = tk.Tk()
    menu.title("PROYECTO NO.1")
    menu.geometry("900x400")
    menu.configure(bg="#212F3C")
    menu.resizable(False, False)
    #Crea la barra del menu
    barra_Menu = tk.Menu()
    #Crea el primer elemento el cual será para archivo y se enlaza a la barra
    menuArchivo = tk.Menu(barra_Menu, tearoff=False)
    #Al add_command se le puede pasar el parametro accelerator="Ctrl+N" para agregar un atajo con el teclado
    menuArchivo.add_command(label="Abrir",accelerator="Ctrl+N",command= abrir)
    menu.bind_all("<Control-n>",abrir )
    menuArchivo.add_command(label="Guardar", command= guardar, state=tk.DISABLED)
    menuArchivo.add_command(label="Guardar Como", command=guardarComo, state=tk.DISABLED)
    menuArchivo.add_command(label="Analizar", command= analizar, state=tk.DISABLED)
    menuArchivo.add_command(label="Errores", command= errores, state=tk.DISABLED)
    menuArchivo.add_separator()
    menuArchivo.add_command(label="Salir", command=menu.quit, activebackground="Red")

    menuAyuda= tk.Menu(barra_Menu, tearoff=False)
    menuAyuda.add_command(label="Manual de Usuario", command= manualUsuario)
    menuAyuda.add_command(label="Manual Tecnico", command= manualTecnico)
    menuAyuda.add_command(label="Temas de Ayuda", command= temasAyuda)

    # a la barra menú le agregamos el menuArchivo
    barra_Menu.add_cascade(menu= menuArchivo, label= "Archivo")
    barra_Menu.add_cascade(menu= menuAyuda, label= "Ayuda")

    textLeer = tk.Text()
    textLeer.configure(bg="#C8C885", state="disabled")
    textLeer.place(x= 5, y =5, height= 350, width= 440)

    textErores = tk.Text()
    textErores.configure(bg="#BFBF02", state="disabled") #, state="disabled"
    textErores.place(x= 455, y =5, height= 350, width= 440)
```

```python

    label1 = tk.Label(menu, text="Archivo abierto", bg="#212F3C", fg="#FFFFFF",width= 20, font=("Arial", 13)).place(x= 125, y =370)

    label1 = tk.Label(menu, text="Archivo Errores", bg="#212F3C", fg="#FFFFFF",width= 20, font=("Arial", 13)).place(x= 575, y =370)

    menu.config(menu=barra_Menu)
    menu.mainloop() # Permite mostrar la ventana
```

Método abrir

Primero se crea la variable almacenar la cual servirá para guardar todo el texto que tiene el archivo a abrir, la variable url guardará la dirección donde se encuentra dicho archivo, y para manejar el analizador realizamos la instancia hacia la clase AFD().

Ahora en el método abrir se utiliza filedialog.askopenfilename() para abrir una ventana que permite acceder a los archivos y seleccionar el que se requiera utilizar en el programa, el cual debe ser con la extensión .json, luego con if se valida si se ha seleccionado algún archivo, si el archivo es cargado entonces abre el archivo con el método open, y con read() lee el contenido del mismo y lo almacena en la variable creada globalmente, además se agregan estos datos al cuadro de texto que se creó inicialmente para poder editar el contenido del mismo, y se habilitan las demás opciones de la barra

de menú con entryconfig, y con showinfo indicamos un mensaje para dar a entender al usuario que que el archivo se cargó en el sistema de forma exitosa.

```
9   almacenar =""
10  urlAlmacenar = ""
11  enviandoAnalisis = AFD()
12  #Metodos y Funciones para la sección de Archivo ********************************************************************************
13  def abrir(event = None):
14      global urlAlmacenar
15      urlArchivo = filedialog.askopenfilename(initialdir="./", title="Seleccione un Archivo", filetypes=(("Archivo json", "*.json"), ("all
        files", "*.*")))
16      if urlArchivo != "":
17
18          leer = open(urlArchivo, "rt")
19          urlAlmacenar = urlArchivo
20          global almacenar
21          almacenar = leer.read()
22          leer.close()
23
24          textLeer.configure(state=tk.NORMAL)
25          textLeer.insert("1.0", almacenar)
26
27          menuArchivo.entryconfig(1,state = tk.NORMAL)
28          menuArchivo.entryconfig(2,state = tk.NORMAL)
29          menuArchivo.entryconfig(3,state = tk.NORMAL)
30          menuArchivo.entryconfig(4,state = tk.NORMAL)
31
32          showinfo(title="Abierto", message="Archivo leído exitosamente")
33      else :
34          #showerror(title="Error", message="El tamaño maximo de organismos es: 1000 \nPorfavor ingrese menos organismos")
35          showwarning(title="Advertencia", message="No ingresó ningun archivo")
```

Método guardar

Se crea una variable que tendrá el método open para acceder al archivo que se abrió y poder editar en él, guardando lo que obtenga del área de texto con el método get, y este se escribe con write en el archivo, luego se cierra esté para evitar el seguir usándolo sin ser necesario.

```
37  def guardar(event = None):
38      saveArchivo = open(urlAlmacenar, "w")
39      saveArchivo.write(textLeer.get("1.0","end"))
40      saveArchivo.close()
```

Método guardar como

Se crea el método para permitir guardar lo que se editó en el cuadro de texto con otro nombre y en la ubicación que el usuario requiera, para ello se comienza creando la variable con filedialog.asksaveasfilename() el cual permite abrir la ventana para seleccionar la ruta donde se guardará el archivo y agregarle en nombre al mismo.

Se valida con if si el usuario ha ingresado algúna ruta y agregado un nombre al archivo, si el usuario ingresa correctamente los datos se utiliza open para crear el archivo y permitir escribir en el con el método write() y este obtiene con get los datos que se encuentran en el cuadro de texto y para finalizar se cierra el archivo con close().

Si el usuario cancela la opción de guardar como, entonces se muestra un mensaje de advertencia indicando que puede perder los datos si no los guarda.

```
42  def guardarComo(event = None):
43      guardar_Como = filedialog.asksaveasfilename(initialdir="./", title="Guardar Como", filetypes=(("Archivo json", ".json"), ("all
    files", "*.*")))
44      if guardar_Como != "":
45          saveComoArchivo = open(guardar_Como +".json", "w") #+".json"
46          saveComoArchivo.write(textLeer.get("1.0","end"))
47          saveComoArchivo.close()
48
49          showinfo(title="Guardado", message="Archivo guardado exitosamente")
50      else :
51          showwarning(title="Advertencia", message="¡Si no guarda el archivo se perderan los datos!")
```

Método analizar

Inicia validando si existen datos en la variable almacenar, si la variable contiene datos, entonces accede al método analizando() de la clase AFD(), y se envia como parámetro a la variable con los datos, con el método analizandoSintacticamente() se realizan las validaciones correspondientes para hacer los cálculos y escribirlos en un archivo .dot, el cual se escribe con el método write(), luego con os.system convertimos a pdf el contenido del archivo .dot para crear la grafica, y con el método webbrowser.open_new() se abre automáticamente el archivo pdf para que el usuario pueda visualizarlo correctamente.

```
53  def analizar(event = None):
54      #https://docs.python.org/es/3/reference/lexical_analysis.html
55      global almacenar
56      if almacenar != "":
57          try:
58              enviandoAnalisis.analizando(almacenar)
59              #enviandoAnalisis.imprimir_tokens()
60
61              operacionesGra = enviandoAnalisis.analizandoSintacticamente()
62              #ESCRIBIENDO .DOT
63              archivoGrafica = open(".\Proyecto1\RESULTADOS_201901103.dot","w")
64              archivoGrafica.write(operacionesGra)
65              archivoGrafica.close()
66              os.system("dot -Tpdf .\Proyecto1\RESULTADOS_201901103.dot -o  .\Proyecto1\RESULTADOS_201901103.pdf")
67              pathOperaciones = ".\Proyecto1\RESULTADOS_201901103.pdf"
68              webbrowser.open_new(pathOperaciones)
69          except Exception as e:
70              pass
71      else:
72          showwarning(title="Advertencia", message="Por favor cargue un archivo")
73
```

Método errores

Se valida con if si la variable almacenar tiene datos, si contiene datos, entonces se accede al método erroresValidados() de la clase AFD(), el cual retornará una cadena con el archivo escrito con los datos de los errores encontrados, después de haber analizado el archivo.

Estos datos se muestran en el área de texto con el método insert(), y se desactiva dicha área para que el usuario no pueda editar en él. Luego con open indicamos en donde se guardará el archivo .json con los datos, y se escriben los datos con el método write() y

cerramos el archivo con close(). Además si la variable no contiene datos entonces se muestra un mensaje indicando que se cargue un archivo con anterioridad.

```
74  def errores(event = None):
75      global almacenar
76      if almacenar != "":
77          textErores.configure(state=tk.NORMAL)
78          escribiendoE = enviandoAnalisis.erroresValidados()
79          textErores.insert("1.0", escribiendoE)
80          textErores.configure(state="disabled")
81
82          archivoErrores = open(".\Proyecto1\ERRORES_201901103.json","w")
83          archivoErrores.write(escribiendoE)
84          archivoErrores.close()
85
86      else:
87          showwarning(title="Advertencia", message="Por favor cargue un archivo")
```

Método manual usuario

Este método tiene la finalidad de abrir el manual de usuario automáticamente, para que el usuario pueda visualizarlo, para ello se utiliza webbrowser.open_new() y se envia como parámetro la ruta donde se encuentra el manual pdf.

```
91  def manualUsuario(event = None):
92      pathUsuario = ".\Proyecto1\Documentacion\Manual Usuario.pdf"
93      webbrowser.open_new(pathUsuario)
```

Método manual tecnico

Este método tiene la finalidad de abrir el manual tecnico automáticamente, para que el usuario pueda visualizarlo, para ello se utiliza webbrowser.open_new() y se envia como parámetro la ruta donde se encuentra el manual pdf.

```
95  def manualTecnico(event = None):
96      pathTecnico = ".\Proyecto1\Documentacion\Manual Tecnico.pdf"
97      webbrowser.open_new(pathTecnico)
```

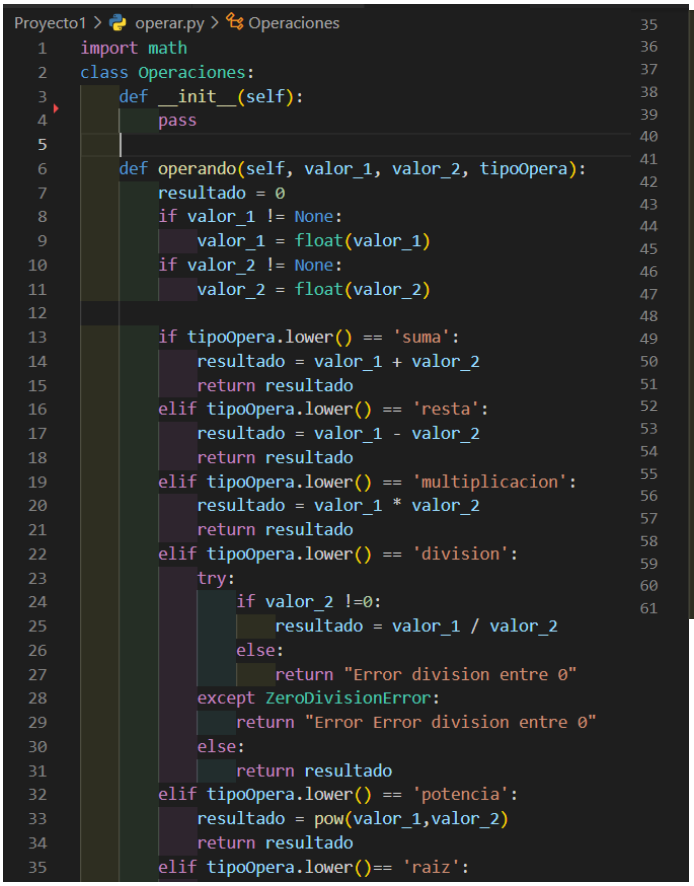
Método temas ayuda

Simplemente se muestra una ventana emergente informativa, indicando los datos del desarrollador de la aplicación.

```
99  def temasAyuda(event = None):
100     showinfo(title="Información del desarrollador", message="Josué Daniel Rojché García\nCarnet: 201901103\nLenguajes Formales y de
        Programación\nSección: A-")
```

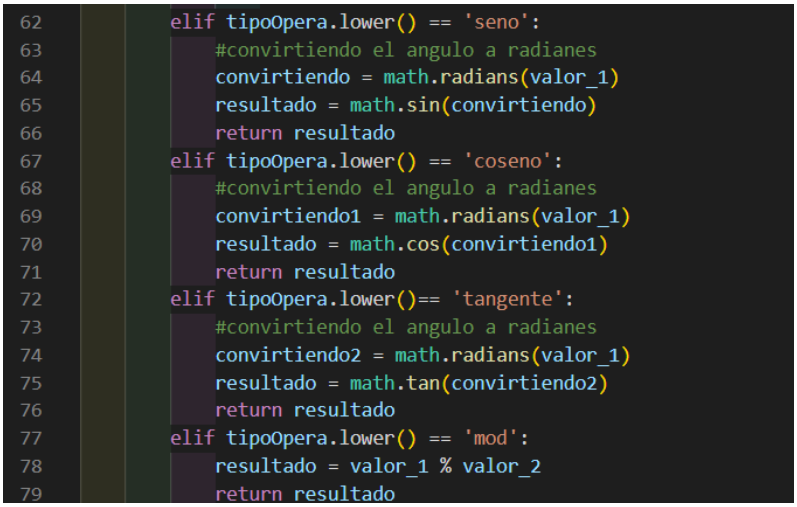
Clase operar

Se utiliza la librería math para realizar algunos cálculos, el método operando recibe como parámetros valor 1, valor 2 y el tipo de operación a realizar, dentro del método se validan con if que los datos no sean nulos, y si no lo son se verifica que el tipo de operación coincida con la que corresponde y si existe coincidencia se realiza la operación y se retorna el resultado.



```
Proyecto1 > operar.py > Operaciones
import math
class Operaciones:
    def __init__(self):
        pass

    def operando(self, valor_1, valor_2, tipoOpera):
        resultado = 0
        if valor_1 != None:
            valor_1 = float(valor_1)
        if valor_2 != None:
            valor_2 = float(valor_2)

        if tipoOpera.lower() == 'suma':
            resultado = valor_1 + valor_2
            return resultado
        elif tipoOpera.lower() == 'resta':
            resultado = valor_1 - valor_2
            return resultado
        elif tipoOpera.lower() == 'multiplicacion':
            resultado = valor_1 * valor_2
            return resultado
        elif tipoOpera.lower() == 'division':
            try:
                if valor_2 !=0:
                    resultado = valor_1 / valor_2
                else:
                    return "Error division entre 0"
            except ZeroDivisionError:
                return "Error Error division entre 0"
            else:
                return resultado
        elif tipoOpera.lower() == 'potencia':
            resultado = pow(valor_1,valor_2)
            return resultado
        elif tipoOpera.lower()== 'raiz':
```

```
    elif tipoOpera.lower()== 'raiz':
        try:
            if valor_1 >= 0:
                resultado = math.sqrt(valor_1)
            else:
                return "No acepta negativos"
        except Exception as e:
            return e
        else:
            return resultado
    elif tipoOpera.lower() == 'inverso':
        try:
            if valor_1 !=0:
                if valor_1 > 1:
                    resultado = "1" + "/" + str(valor_1)
                elif valor_1 < -1:
                    resultado = "-1" + "/" + str(valor_1 * -1)
                elif valor_1 < 0 and valor_1 > -1:
                    resultado = 1/valor_1
                elif valor_1 > 0 and valor_1 < 1:
                    resultado = 1/valor_1
            else:
                return "Error no existe el inverso entre 0"
        except ZeroDivisionError:
            return "Error"
        else:
            return resultado
```



```
    elif tipoOpera.lower() == 'seno':
        #convirtiendo el angulo a radianes
        convirtiendo = math.radians(valor_1)
        resultado = math.sin(convirtiendo)
        return resultado
    elif tipoOpera.lower() == 'coseno':
        #convirtiendo el angulo a radianes
        convirtiendo1 = math.radians(valor_1)
        resultado = math.cos(convirtiendo1)
        return resultado
    elif tipoOpera.lower()== 'tangente':
        #convirtiendo el angulo a radianes
        convirtiendo2 = math.radians(valor_1)
        resultado = math.tan(convirtiendo2)
        return resultado
    elif tipoOpera.lower() == 'mod':
        resultado = valor_1 % valor_2
        return resultado
```

Clase Token

En esta clase se ingresan fila, columna y lexema en el constructor, los cuales servirán para agregar en conjunto los datos a la tabla de tokens.

```python
class Token:
    def __init__(self, row, columna, lexema):
        self.row = row
        self.columna = columna
        self.lexema = lexema
```

Clase AFD

En el método constructor se inicializa la lista de letras que podrá leer el automata, también la lista de números que podrá utilizarse, se inicializa las variables necesarias a utilizar, como fila, columna, los estados actual, anterior, estados finales, así como la lista para almacenar los tokens, y la lista para almacenar los errores, etc.

```python
from token import Token
from operar import Operaciones

class AFD:
    def __init__(self):
        self.letras = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'ñ', 'o', 'p', 'q','r', 's', 't', 'u', 'v',
        'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '-']
        self.numeros = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
        self.fila = 0
        self.columna = 0
        self.estadoActual = ''
        self.estadoAnterior = ''
        self.estadoFinal = ['q10', 'q23']
        self.tabla = []
        self.tablaErrores = []
        self.auxiliarTexto = ""
        self.iterar = 1
        self.escribiendoGrafica = ""
        self.enlaceNodosSub = ""
```

Método analizando

Este método recibe como parámetro el texto que se obtiene del archivo a analizar, el texto se almacena en una variable para manejar los datos localmente, por lo que se utiliza un bucle while para recorrer todos los datos del texto, luego se accede a cada carácter del texto y con un if se valida que se omitan los espacios, y saltos de línea que encuentre en el texto, para luego comenzar con las validaciones de if que indicaran en que estado se encuentra y que carácter puede almacenar en base a el diseño del Autómata Finito Determinista, el cual encuentra el diseño al final de este documento.

Básicamente lo que se realiza es repetitivo, solamente se valida que se encuentre en un estado en especifico y luego cuando ingrese al carácter que coincida, envía el carácter como parámetro al método almacenarToken(), el cual sirve para enviar los datos a la lista de tokens. Luego se indica el estado actual como estado anterior, y el estado actual es

el siguiente estado del diseño del AFD, y con else validamos que si el carácter no es admitido por el autómata, entonces se trata de un error, por lo cual es necesario almacenar el carácter en el método almacenarError() y se utiliza una bandera que sirve para indicar que el error se ha encontrado y se quita hasta que encuentre nuevamente la bandera como false, en otro estado.

```
    def analizando(self, texto1):
        tok = ''
        # Eliminando espacios y saltos de linea de la cadena
        # texto = texto.replace("\n","")
        # texto = texto.replace(" ","")
        validandoError = False
        texto = texto1
        # recorriendo el texto
        while len(texto) > 0:
            caracter = texto[0]
            if caracter == '\n':
                self.fila += 1
                self.columna = 0
                texto = texto[1:]
                continue
            elif caracter == ' ':
                self.columna += 1
                texto = texto[1:]
                continue

            # validaciones de acuerdo al caracter que se está leyendo
            # valida cuando inicia leyendo el texto
            if self.estadoActual == '':
                if caracter == '{':
                    self.almacenarToken(caracter)
                    self.estadoAnterior = 'q0'
                    self.estadoActual = 'q1'
                else:
                    validandoError = True
                    self.almacenarError(caracter)
            # valida cuando se encuentra en el estado q1
```

```
                    self.almacenarError(caracter)
            # valida cuando se encuentra en el estado q1
            if self.estadoActual == 'q1':
                if caracter == '{':
                    self.almacenarToken(caracter)
                    self.estadoAnterior = 'q1'
                    self.estadoActual = 'q2'
                elif caracter == '"':
                    self.almacenarToken(caracter)
                    self.estadoAnterior = 'q1'
                    self.estadoActual = 'q3'
                else:
                    validandoError = True
                    self.almacenarError(caracter)
            # valida cuando se encuentra en q3 ************************
            elif self.estadoActual == 'q3':
                if caracter.lower() in self.letras:
                    tok += caracter
                    self.estadoAnterior = 'q3'
                    self.estadoActual = 'q4'
                else:
                    validandoError = True
                    self.almacenarError(caracter)
            # valida cuando se encuentra en q4
            elif self.estadoActual == 'q4':
                if caracter.lower() in self.letras:
                    tok += caracter
                    self.estadoAnterior = 'q4'
                    self.estadoActual = 'q4'
                elif caracter == '"':
                    self.almacenarToken(tok)
                    tok = ''
                    self.almacenarToken(caracter)
                    self.estadoAnterior = 'q4'
                    self.estadoActual = 'q5'
                else:
                    validandoError = True
```

```
                self.estadoActual = 'q5'
            else:
                validandoError = True
                self.almacenarError(caracter)
        elif self.estadoActual == 'q5':
            if caracter == ':':
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q5'
                self.estadoActual = 'q6'
            else:
                validandoError = True
                self.almacenarError(caracter)
        elif self.estadoActual == 'q6':
            if caracter == '"':
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q6'
                self.estadoActual = 'q7'
            else:
                validandoError = True
                self.almacenarError(caracter)
        elif self.estadoActual == 'q7':
            if caracter.lower() in self.letras:
                tok += caracter
                self.estadoAnterior = 'q7'
                self.estadoActual = 'q8'
            else:
                validandoError = True
                self.almacenarError(caracter)
        # valida cuando se encuentra en q8
        elif self.estadoActual == 'q8':
            if caracter.lower() in self.letras:
                tok += caracter
                self.estadoAnterior = 'q8'
                self.estadoActual = 'q8'
```

```
                self.estadoActual = 'q8'
            elif caracter == '"':
                self.almacenarToken(tok)
                tok = ''
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q8'
                self.estadoActual = 'q9'
            else:
                validandoError = True
                self.almacenarError(caracter)
        # valida cuando se encuentra en q9
        elif self.estadoActual == 'q9':
            if caracter == ',':
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q9'
                self.estadoActual = 'q1'
                validandoError = False
            elif caracter == '}':
                # estado de aceptación
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q9'
                self.estadoActual = 'q10'
                validandoError = False
            else:
                validandoError = True
                self.almacenarError(caracter)
        # Aqui termina una parte  ************************
        # Segunda parte
        # validar estado q2
        elif self.estadoActual == 'q2':
            if caracter == '"':
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q2'
                self.estadoActual = 'q11'
            else:
                validandoError = True
                self.almacenarError(caracter)
```

```
        # validacion estado q11
        elif self.estadoActual == 'q11':
            if caracter.lower() in self.letras:
                tok += caracter
                self.estadoAnterior = 'q11'
                self.estadoActual = 'q12'
            else:
                validandoError = True
                self.almacenarError(caracter)
        # validando estado q12
        elif self.estadoActual == 'q12':
            if caracter.lower() in self.letras:
                tok += caracter
                self.estadoAnterior = 'q12'
                self.estadoActual = 'q12'
            elif caracter == '"':
                self.almacenarToken(tok)
                tok = ''
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q12'
                self.estadoActual = 'q13'
            else:
                validandoError = True
                self.almacenarError(caracter)
        # validando estado q13
        elif self.estadoActual == 'q13':
            if caracter == ':':
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q13'
                self.estadoActual = 'q14'
            else:
                validandoError = True
                self.almacenarError(caracter)
        # validando estado q14
        elif self.estadoActual == 'q14':
            if caracter in self.numeros:
                tok += caracter
```

```
                tok += caracter
                self.estadoAnterior = 'q14'
                self.estadoActual = 'q18'
            elif caracter == '"':
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q14'
                self.estadoActual = 'q15'
            elif caracter == '[':
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q14'
                self.estadoActual = 'q2'
            else:
                validandoError = True
                self.almacenarError(caracter)
        # validando estado q15
        elif self.estadoActual == 'q15':
            if caracter.lower() in self.letras:
                tok += caracter
                self.estadoAnterior = 'q15'
                self.estadoActual = 'q16'
            else:
                validandoError = True
                self.almacenarError(caracter)
        # validando estado q16
        elif self.estadoActual == 'q16':
            if caracter.lower() in self.letras:
                tok += caracter
                self.estadoAnterior = 'q16'
                self.estadoActual = 'q16'
            elif caracter == '"':
                self.almacenarToken(tok)
                tok = ''
                self.almacenarToken(caracter)
                self.estadoAnterior = 'q16'
                self.estadoActual = 'q17'
```

El código continúa hasta aumentar la columna, eliminar el carácter analizado de la cadena de texto y retornar el estado actual si corresponde al estado de aceptación.

```
319             # validando estado q22
320             elif self.estadoActual == 'q22':
321                 if caracter == ',':
322                     self.almacenarToken(caracter)
323                     self.estadoAnterior = 'q22'
324                     self.estadoActual = 'q1'
325                     validandoError = False
326                     # estado de acpetación
327                 elif caracter == '}':
328                     self.almacenarToken(caracter)
329                     self.estadoAnterior = 'q22'
330                     self.estadoActual = 'q23'
331                 else:
332                     validandoError = True
333                     self.almacenarError(caracter)
334             # validando estado q24
335             elif self.estadoActual == 'q24':
336                 if caracter == ']':
337                     self.almacenarToken(caracter)
338                     self.estadoAnterior = 'q24'
339                     self.estadoActual = 'q24'
340                 elif caracter == ',':
341                     self.almacenarToken(caracter)
342                     self.estadoAnterior = 'q24'
343                     self.estadoActual = 'q2'
344                 elif caracter == '}':
345                     self.almacenarToken(caracter)
346                     self.estadoAnterior = 'q24'
347                     self.estadoActual = 'q22'
348                 else:
349                     validandoError = True
350                     self.almacenarError(caracter)
351 
352             self.columna += 1
353             texto = texto[1:]
354         return self.estadoActual in self.estadoFinal
```

Método almacenarToken

Recibe como parámetro el lexema y se crea una instancia para almacenar la fila y columna que corresponde al lexema encontrado en el automata, luego este se almacena en la tabla de tokens.

```
356     def almacenarToken(self, lexema):
357         newToken = Token(self.fila, self.columna, lexema)
358         self.tabla.append(newToken)
```

Método almacenarError

Recibe como parámetro el lexema y se crea una instancia para almacenar la fila y columna que corresponde al lexema encontrado en el autómata, luego este se almacena en la tabla de errores.

```
748     def almacenarError(self, lexemaError):
749         newToken1 = Token(self.fila, self.columna, lexemaError)
750         self.tablaErrores.append(newToken1)
```

Método erroresValidados

Este método sirve para acceder a la tabla de errores y escribir en una variable la estructura para el archivo de errores .json que requiere generar el usuario en la opción de errores, por lo que simplemente se accede a cada dato y se agrega al espacio que corresponde, luego se elimina la ultima coma y se retorna la cadena.

```
368     def erroresValidados(self):
369         escribiendoEstructura = "{\n"
370         iterador = 1
371         for token in self.tablaErrores:
372             escribiendoEstructura = escribiendoEstructura + "\t{\n"
373             escribiendoEstructura = escribiendoEstructura + \
374                 '\t\t"No.":' + str(iterador) + "\n"
375             escribiendoEstructura = escribiendoEstructura + \
376                 '\t\t"Descripcion-Token":{\n'
377             escribiendoEstructura = escribiendoEstructura + \
378                 '\t\t\t"Lexema": ' + str(token.lexema) + '\n'
379             escribiendoEstructura = escribiendoEstructura + '\t\t\t"Tipo": Error\n'
380             escribiendoEstructura = escribiendoEstructura + \
381                 '\t\t\t"Columna": ' + str(token.columna) + '\n'
382             escribiendoEstructura = escribiendoEstructura + \
383                 '\t\t\t"Fila": ' + str(token.row) + '\n'
384             escribiendoEstructura = escribiendoEstructura + "\t\t}\n"
385             escribiendoEstructura = escribiendoEstructura + "\t},\n"
386             iterador += 1
387             # print(token.row, token.columna, token.lexema)
388         total = escribiendoEstructura[:-2] + "\n}"
389         # escribiendoEstructura = total +"\n}"
390         return total
```

Método analizandoSintacticamente

Este método sirve para obtener los datos de la tabla de tokens y validar que se realicen las operaciones conforme la jerarquía con la que fue ingresado el archivo, respetando la estructura del mismo, por lo que se utiliza un bucle while para recorrer toda la tabla, sin embargo también se requiere de escribir el archivo .dot con los datos obtenidos de los cálculos, para así mostrarlos en forma gráfica. En las validaciones if podrá observar que se va validando las opciones para las asignaciones de los valores que corresponden a sus determinados campos, sin embargo cuando se llega hasta el if donde se obtiene un

valor anidado se comienza a ingresar a un método que puede hacerse recursivo si es necesario validar otras operaciones que sean anidadas, de tal manera que se realicen todas las operaciones de manera ordenada y correctamente. Esto es algo que se repite en todo el método, por lo cual solo se agregan las otras imágenes.

```
394     def analizandoSintacticamente(self):
395         self.iniciandoEsGrafica = '''digraph G {
396         rankdir=LR
397         size= 8.5
398         ranksep=1
399         bgcolor = lightgoldenrodyellow
400         margin = 0.1
401         edge[arrowhead = vee];'''
402         self.escribiendoGrafica = ""
403         calculando = Operaciones()
404         # recorrer la tabla y ir validando con ifs anidados para ver los datos que están como frases o como numeros
405         self.iterar = 1
406         # valida si viene un  corchete
407         while self.iterar < len(self.tabla):
408             if self.tabla[self.iterar].lexema == "{":
409                 print(self.tabla[self.iterar].lexema)
410                 self.iterar += 1
411                 # valida si el siguiente es una comilla
412                 if self.tabla[self.iterar].lexema == '"':
413                     self.iterar += 1
414                     if self.tabla[self.iterar].lexema.lower() == 'operacion':
415                         asignacion_Operacion = ""
416                         self.iterar += 1
417                         # valida si el que finaliza es una comilla entonces se creó la variable operacion
418                         if self.tabla[self.iterar].lexema == '"':
419                             self.iterar += 1
420                             if self.tabla[self.iterar].lexema == ':':
421                                 # aqui se valida si lo ve sigue son comillas para la asignacion de la variable
422                                 self.iterar += 1
423                                 if self.tabla[self.iterar].lexema == '"':
424                                     self.iterar += 1
425                                     asignacion_Operacion = self.tabla[self.iterar].lexema
426                                     print(asignacion_Operacion)
427                                     self.iterar += 1
428                                     if self.tabla[self.iterar].lexema == '"':
429                                         self.iterar += 1
```

```
430                                     if self.tabla[self.iterar].lexema == ',':
431                                         self.iterar += 1
432                                         if self.tabla[self.iterar].lexema == '"':
433                                             self.iterar += 1
434                                             if self.tabla[self.iterar].lexema.lower() == 'valor1':
435                                                 valor1 = ""
436                                                 self.iterar += 1
437                                                 if self.tabla[self.iterar].lexema == '"':
438                                                     self.iterar += 1
439                                                     if self.tabla[self.iterar].lexema == ':':
440                                                         self.iterar += 1
441                                                         if self.tabla[self.iterar].lexema == '[':
442                                                             resultado_valor1 = self.operacionAnidada()
443                                                             enlace1 = self.enlaceNodosSub
444                                                             print(resultado_valor1)
445                                                             if self.tabla[self.iterar].lexema == ']':
446                                                                 self.iterar += 1
447                                                                 if self.tabla[self.iterar].lexema == ',':
448                                                                     self.iterar += 1
449                                                                     if self.tabla[self.iterar].lexema == '"':
450                                                                         self.iterar += 1
451                                                                         if self.tabla[self.iterar].lexema.lower() == 'valor2':
452                                                                             valor2 = ""
453                                                                             self.iterar += 1
454                                                                             if self.tabla[self.iterar].lexema == '"':
455                                                                                 self.iterar += 1
456                                                                                 if self.tabla[self.iterar].lexema == ':':
457                                                                                     self.iterar += 1
458                                                                                     if self.tabla[self.iterar].lexema == '[':
459                                                                                         resultado_valor2 = self.operacionAnidada()
460                                                                                         print(resultado_valor2)
461                                                                                         resultadoCalculos = calculando.operando(resultado_valor1, resultado_valor2,
                                                                                            asignacion_Operacion)
462                                                                                         print("Resultado: " + str(resultadoCalculos))
463                                                                                         enlace2 = self.enlaceNodosSub
464                                                                                         self.escribiendoGrafica = self.escribiendoGrafica +'''
465     a0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"];
```

```
    a0 ->'''+ enlace1 +''';
    a0 ->'''+ enlace2 +''';'''
                                                                    if self.tabla[self.iterar].lexema == ']':
                                                                        self.iterar += 1
                                                                        if self.tabla[self.iterar].lexema == ']':
                                                                            self.iterar += 1
                                                                        elif self.tabla[self.iterar].lexema == '}':
                                                                            print(self.tabla[self.iterar].lexema)
                                                                            self.iterar += 1
                                                                else:
                                                                    valor2 = self.tabla[self.iterar].lexema
                                                                    print(valor2)
                                                                    self.iterar += 1
                                                                    resultadoCalculos = calculando.operando(resultado_valor1, valor2,
                                                                    asignacion_Operacion)

                                                                    print("Resultado: " + str(resultadoCalculos))

                                                                    self.escribiendoGrafica = self.escribiendoGrafica + '''
    b0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"];
    b2  [label = "'''+str(valor2)+'''"];
    b0 ->'''+ enlace1 +''';
    b0 -> b2;'''
                                                                    if self.tabla[self.iterar].lexema == '}':
                                                                        print(self.tabla[self.iterar].lexema)
                                                                        self.iterar += 1

                                                            else:
                                                                valor1 = self.tabla[self.iterar].lexema
                                                                print(valor1)
                                                                self.iterar += 1
                                                                if self.tabla[self.iterar].lexema == ',':
                                                                    self.iterar += 1
                                                                    if self.tabla[self.iterar].lexema == '"':
                                                                        self.iterar += 1
                                                                        if self.tabla[self.iterar].lexema.lower() == 'valor2':
                                                                            valor2 = ""
```

```
                                                                        self.iterar += 1
                                                                        if self.tabla[self.iterar].lexema == '"':
                                                                            self.iterar += 1
                                                                            if self.tabla[self.iterar].lexema == ':':
                                                                                self.iterar += 1
                                                                                if self.tabla[self.iterar].lexema == '[':
                                                                                    resultado_valor2 = self.operacionAnidada()
                                                                                    print(resultado_valor2)
                                                                                    resultadoCalculos = calculando.operando(valor1, resultado_valor2,
                                                                                    asignacion_Operacion)
                                                                                    print("Resultado: " + str(resultadoCalculos))
                                                                                    enlace2 = self.enlaceNodosSub
                                                                                    self.escribiendoGrafica = self.escribiendoGrafica + '''
    c0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"];
    c1  [label = "'''+str(valor1)+'''"];
    c0 -> c1;
    c0 ->'''+ enlace2 +''';'''
                                                                                    if self.tabla[self.iterar].lexema == ']':
                                                                                        self.iterar += 1
                                                                                        if self.tabla[self.iterar].lexema == '}':
                                                                                            print(self.tabla[self.iterar].lexema)
                                                                                            self.iterar += 1
                                                                                else:
                                                                                    valor2 = self.tabla[self.iterar].lexema
                                                                                    print(valor2)
                                                                                    resultadoCalculos = calculando.operando(valor1, valor2, asignacion_Operacion)
                                                                                    print("Resultado: " + str(resultadoCalculos))

                                                                                    self.escribiendoGrafica = self.escribiendoGrafica + '''
    d0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"];
    d1  [label = "'''+str(valor1)+'''"];
    d2  [label = "'''+str(valor2)+'''"];
    d0 -> d1;
    d0 -> d2;'''
                                                                                    self.iterar += 1
                                                                                    if self.tabla[self.iterar].lexema == '}':
                                                                                        print(self.tabla[self.iterar].lexema)
```

```
                                                                    self.iterar += 1
            elif self.tabla[self.iterar].lexema == '"':
                    self.iterar += 1
                    if self.tabla[self.iterar].lexema.lower() == 'texto':
                        asignacion_texto = ""
                        self.iterar += 1
                        # valida si el que finaliza es una comilla entonces se creó la variable operacion
                        if self.tabla[self.iterar].lexema == '"':
                            self.iterar += 1
                            if self.tabla[self.iterar].lexema == ':':
                                # aqui se valida si lo ve sigue son comillas para la asignacion de la variable
                                self.iterar += 1
                                if self.tabla[self.iterar].lexema == '"':
                                    self.iterar += 1
                                    asignacion_texto = self.tabla[self.iterar].lexema
                                    print(asignacion_texto)
                                    self.iterar += 1
                                    if self.tabla[self.iterar].lexema == '"':
                                        continue
                    elif self.tabla[self.iterar].lexema.lower() == 'color-fondo-nodo':
                        asignacion_ColorFondo = ""
                        self.iterar += 1
                        # valida si el que finaliza es una comilla entonces se creó la variable operacion
                        if self.tabla[self.iterar].lexema == '"':
                            self.iterar += 1
                            if self.tabla[self.iterar].lexema == ':':
                                # aqui se valida si lo ve sigue son comillas para la asignacion de la variable
                                self.iterar += 1
                                if self.tabla[self.iterar].lexema == '"':
                                    self.iterar += 1
                                    asignacion_ColorFondo = self.tabla[self.iterar].lexema
                                    print(asignacion_ColorFondo)
                                    self.iterar += 1
                                    if self.tabla[self.iterar].lexema == '"':
                                        continue
                    elif self.tabla[self.iterar].lexema.lower() == 'color-fuente-nodo':
                        asignacion_ColorFuente = ""
```

```
                        self.iterar += 1
                        # valida si el que finaliza es una comilla entonces se creó la variable operacion
                        if self.tabla[self.iterar].lexema == '"':
                            self.iterar += 1
                            if self.tabla[self.iterar].lexema == ':':
                                # aqui se valida si lo ve sigue son comillas para la asignacion de la variable
                                self.iterar += 1
                                if self.tabla[self.iterar].lexema == '"':
                                    self.iterar += 1
                                    asignacion_ColorFuente = self.tabla[self.iterar].lexema
                                    print(asignacion_ColorFuente)
                                    self.iterar += 1
                                    if self.tabla[self.iterar].lexema == '"':
                                        continue
                    elif self.tabla[self.iterar].lexema.lower() == 'forma-nodo':
                        asignacion_Forma = ""
                        self.iterar += 1
                        # valida si el que finaliza es una comilla entonces se creó la variable operacion
                        if self.tabla[self.iterar].lexema == '"':
                            self.iterar += 1
                            if self.tabla[self.iterar].lexema == ':':
                                # aqui se valida si lo ve sigue son comillas para la asignacion de la variable
                                self.iterar += 1
                                if self.tabla[self.iterar].lexema == '"':
                                    self.iterar += 1
                                    asignacion_Forma = self.tabla[self.iterar].lexema.lower()
                                    print(asignacion_Forma)
                                    self.iterar += 1
                                    if self.tabla[self.iterar].lexema == '"':
                                        continue
                self.iterar += 1
            self.iniciandoEsGrafica = self.iniciandoEsGrafica + 'node [style=filled,color='+asignacion_ColorFondo+', shape = '+asignacion_Forma+',fontcolor= '
            +asignacion_ColorFuente+'];\n\t\tlabel = "'+asignacion_texto+'";'
            #self.escribiendoGrafica = self.escribiendoGrafica + "}"
            self.iniciandoEsGrafica = self.iniciandoEsGrafica+self.escribiendoGrafica + "}"

            return self.iniciandoEsGrafica
```

Método operacionAnidada

Este método básicamente realiza lo mismo que el método anterior, solamente que las validaciones se comienzan desde que encuentra la asignación de la operación para así

llevar el orden de la jerarquía de las operaciones a realizar conforme el archivo leído. Y al encontrar el que corresponde automáticamente retorna el resultado al método anterior.

```
    def operacionAnidada(self):
        calculando = Operaciones()
        self.iterar +=1
        #valida si el siguiente es una comilla
        if self.tabla[self.iterar].lexema == '"':
            self.iterar +=1
            if self.tabla[self.iterar].lexema.lower() == 'operacion':
                asignacion_Operacion = ""
                self.iterar +=1
                #valida si el que finaliza es una comilla entonces se creó la variable operacion
                if self.tabla[self.iterar].lexema == '"':
                    self.iterar +=1
                    if self.tabla[self.iterar].lexema == ':':
                        #aqui se valida si lo ve sigue son comillas para la asignacion de la variable
                        self.iterar +=1
                        if self.tabla[self.iterar].lexema == '"':
                            self.iterar +=1
                            asignacion_Operacion = self.tabla[self.iterar].lexema
                            print(asignacion_Operacion)
                            self.iterar +=1
                            if self.tabla[self.iterar].lexema == '"':
                                self.iterar +=1
                                if self.tabla[self.iterar].lexema == ',':
                                    self.iterar +=1
                                    if self.tabla[self.iterar].lexema == '"':
                                        self.iterar +=1
                                        if self.tabla[self.iterar].lexema.lower() == 'valor1':
                                            valor1 = ""
                                            self.iterar += 1
                                            if self.tabla[self.iterar].lexema == '"':
                                                self.iterar +=1
                                                if self.tabla[self.iterar].lexema == ':':
                                                    self.iterar +=1
                                                    if self.tabla[self.iterar].lexema == '[':
                                                        resultado_valor1 = self.operacionAnidada()
                                                        enlace1 = self.enlaceNodosSub
                                                        print(resultado_valor1)
```

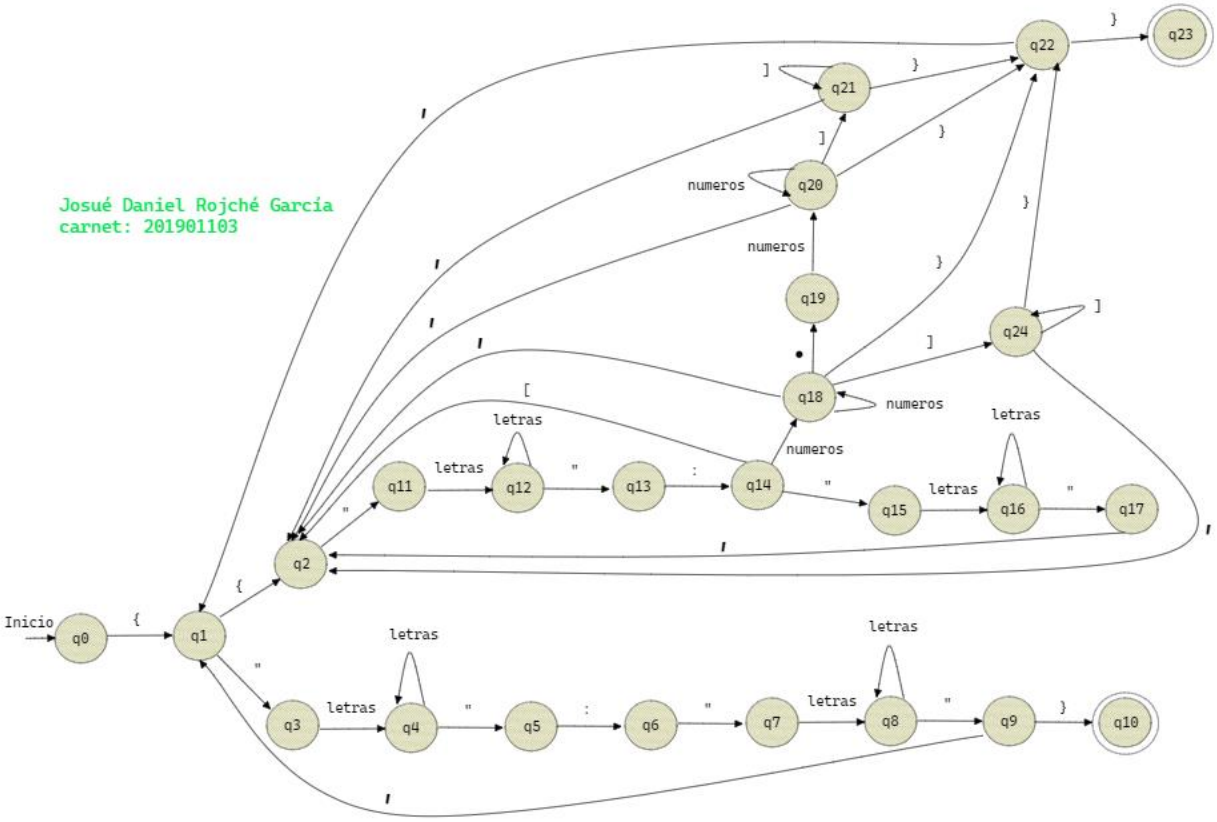
```
                                                    if self.tabla[self.iterar].lexema == ']':
                                                        self.iterar += 1
                                                        if self.tabla[self.iterar].lexema == ',':
                                                            self.iterar += 1
                                                            if self.tabla[self.iterar].lexema == '"':
                                                                self.iterar += 1
                                                                if self.tabla[self.iterar].lexema.lower() == 'valor2':
                                                                    valor2 = ""
                                                                    self.iterar += 1
                                                                    if self.tabla[self.iterar].lexema == '"':
                                                                        self.iterar += 1
                                                                        if self.tabla[self.iterar].lexema == ':':
                                                                            self.iterar += 1
                                                                            if self.tabla[self.iterar].lexema == '[':
                                                                                resultado_valor2 = self.operacionAnidada()
                                                                                print(resultado_valor2)
                                                                                resultadoCalculos = calculando.operando(resultado_valor1, resultado_valor2,
                                                                                asignacion_Operacion)
                                                                                print("Resultado: " + str(resultadoCalculos))
                                                                                enlace2 = self.enlaceNodosSub
                                                                                self.escribiendoGrafica = self.escribiendoGrafica + '''
    e0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"];
    e0 ->'''+ enlace1 +''';
    e0 ->'''+ enlace2 +''';'''

                                                                                self.enlaceNodosSub = "e0"
                                                                                return resultadoCalculos
                                                                            else:
                                                                                valor2 = self.tabla[self.iterar].lexema
                                                                                print(valor2)
                                                                                self.iterar += 1
                                                                                resultadoCalculos = calculando.operando(resultado_valor1, valor2,
                                                                                asignacion_Operacion)
                                                                                print("Resultado: " + str(resultadoCalculos))

                                                                                self.escribiendoGrafica = self.escribiendoGrafica + '''
    f0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"]
```

```
    f2  [label = "'''+str(valor2)+'''"];
    f0 ->'''+ enlace1 +''';
    f0 -> f2;'''
                                                    self.enlaceNodosSub = "f0"
                                                    return resultadoCalculos
                                                else:
                                                    resultadoCalculos = calculando.operando(resultado_valor1, None, asignacion_Operacion)
                                                    print("Resultado: " + str(resultadoCalculos))
                                                    self.escribiendoGrafica = self.escribiendoGrafica + '''
    g0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"];
    g0 ->'''+ enlace1 +''';'''
                                                    self.enlaceNodosSub = "g0"
                                                    return resultadoCalculos
                                            else:
                                                valor1 = self.tabla[self.iterar].lexema
                                                print(valor1)
                                                self.iterar +=1
                                                if self.tabla[self.iterar].lexema == ',':
                                                    self.iterar +=1
                                                    if self.tabla[self.iterar].lexema == '"':
                                                        self.iterar +=1
                                                        if self.tabla[self.iterar].lexema.lower() == 'valor2':
                                                            valor2 = ""
                                                            self.iterar += 1
                                                            if self.tabla[self.iterar].lexema == '"':
                                                                self.iterar +=1
                                                                if self.tabla[self.iterar].lexema == ':':
                                                                    self.iterar +=1
                                                                    if self.tabla[self.iterar].lexema == '[':
                                                                        resultado_valor2 = self.operacionAnidada()
                                                                        print(resultado_valor2)
                                                                        resultadoCalculos = calculando.operando(valor1, resultado_valor2, asignacion_Operacion)
                                                                        print("Resultado: " + str(resultadoCalculos))
                                                                        enlace2 = self.enlaceNodosSub
                                                                        self.escribiendoGrafica = self.escribiendoGrafica + '''
    h0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"];
    h1  [label = "'''+str(valor1)+'''"];
```

```
    h0 -> h1;
    h0 ->'''+ enlace2 +';'
                                                                    self.enlaceNodosSub = "h0"
                                                                    return resultadoCalculos
                                                                else:
                                                                    valor2 = self.tabla[self.iterar].lexema
                                                                    print(valor2)
                                                                    resultadoCalculos = calculando.operando(valor1, valor2, asignacion_Operacion)
                                                                    print("Resultado: " + str(resultadoCalculos))
                                                                    self.escribiendoGrafica = self.escribiendoGrafica + '''
    i0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"]
    i1  [label = "'''+str(valor1)+'''"];
    i2  [label = "'''+str(valor2)+'''"];
    i0 -> i1;
    i0 -> i2;'''
                                                                    self.enlaceNodosSub = "i0"
                                                                    self.iterar +=1
                                                                    return resultadoCalculos
                                                            else:
                                                                resultadoCalculos = calculando.operando(valor1, None, asignacion_Operacion)
                                                                print("Resultado: " + str(resultadoCalculos))
                                                                self.escribiendoGrafica = self.escribiendoGrafica + '''
    j0  [label = "'''+asignacion_Operacion+'''\n'''+str(resultadoCalculos)+'''"];
    j1  [label = "'''+str(valor1)+'''"];
    j0 -> j1;'''
                                                                self.enlaceNodosSub = "j0"
                                                                return resultadoCalculos
```

Autómata Finito Determinista

En la imagen siguiente se ilustra el autómata finito determinista que fue diseñado para darle solución a la lectura correcta del archivo a analizar en el software.



```
Σ: {coma(,),(numeros),(letras), punto(.) ,",},{,],[,:}
    numeros = {0,1,2,3,4,5,6,7,8,9}
    letras = {a,b,c,d,e,f,g,h,i,j,k,l,m,n,ñ,o,p,q,r,s,t,u,v,w,x,y,z,0,1,2,3,4,5,6,7,8,9,-}
Q:{q0,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12,q13,q14,q15,q16,q17,q18,q19,q20,q21,q22,q23,q24}
S:q0
F:{q10, q23}

Transiciones
  (q0, {)          = q1
  (q1, ")          = q3
  (q3, {letras})   = q4
  (q4, {letras})   = q4
  (q4, ")          = q5
  (q5, :)          = q6
  (q6, ")          = q7
  (q7, {letras})   = q8
  (q8, {letras})   = q8
  (q8, ")          = q9
  (q9, ,)          = q1
  (q9, })          = q10
  (q1, {)          = q2
  (q2, ")          = q11
  (q11, {letras})  = q12
  (q12, {letras})  = q12
  (q12, ")         = q13
  (q13, :)         = q14
  (q14, ")         = q15
  (q15, {letras})  = q16
  (q16, {letras})  = q16
  (q16, ")         = q17
  (q17, ,)         = q2
  (q14, [)         = q2
  (q14, {numeros}) = q18
  (q18, {numeros}) = q18
  (q18, ,)         = q2
  (q18, .)         = q19
  (q18, ])         = q24
  (q18, })         = q22
  (q19, {numeros}) = q20
  (q20, {numeros}) = q20
  (q20, ,)         = q2
  (q20, ])         = q21
  (q20, })         = q22
  (q21, })         = q22
  (q21, ,)         = q2
  (q21, ])         = q21
  (q22, ,)         = q1
  (q22, })         = q23
  (q24, ,)         = q2
  (q24, })         = q22
  (q24, ])         = q24
```