Universidad de San Carlos de Guatemala
Escuela de Ciencias y Sistemas
Lenguajes Formales y de Programación
Sección A+

Manual Técnico
Proyecto 1

Arnoldo Luis Antonio González Camey
Carné: 201701548

INTRODUCCIÓN

Este manual va dirigido a todo público que cuente con conocimientos técnicos básicos sobre la programación, y el uso de Python, y que quiera ver la interacción entre listas, clases, programación orientada a objetos, lectura de archivos, así como la utilización de token para la lectura de los archivos y de gramática de tipo 2 para corroborar la estructura del archivo y de las funciones, todo mediane una interfaz gráfica creada por medio de la librería Tkinter; este manual describe el funcionamiento de cada uno de los métodos utilizados en la elaboración del programa. Todo el código fue realizado en el editor de código fuente Visual Studio Code.

REQUERIMIENTOS MÍNIMOS

El sistema operativo necesario es Windows, Linux, Mac o cualquier otro sistema operativo y un procesador de 32 o 64 bits. Un navegador a discreción del usuario, preferiblemente Google Chrome, Microsoft Edge o Mozilla Firefox. Contar con un editor de código fuente como Visual Studio Code, y tener la capacidad de acceder a los comandos del sistema operativo utilizado. A su vez tener instalado el lenguaje de programación Python en una versión igual o superior a la 3.8.7 recomendablemente.

OBJETIVOS

General

- Brindar la información necesaria para poder comprender el funcionamiento del programa y las distintas características que posee.

Específicos

- Representar la funcionalidad técnica del programa.

- Definir claramente el procedimiento de cada método.

Manual Técnico

- Clase Token

Es una clase en la cual se inicializan los atributos del token tal como el lexema, el tipo, la fila, la columna y el id; a su vez se crean unas constantes que son los tipos de Token que pueden existir.

```
class Token():

    lexema_valido = ''
    tipo = 0
    fila = 0
    columna = 0

    CLAVES = 1
    REGISTROS = 2
    IMPRIMIR = 3
    IMPRIMIRLN = 4
    CONTEO = 5
    PROMEDIO = 6
    CONTARSI = 7
    DATOS = 8
    SUMAR = 9
    MAX = 10
    MIN = 11
    EXPORTARREPORTE = 12
    LETRA = 13
    NUMERO = 14
```

```
CADENA = 15
IGUAL = 16
CORCHETE_IZQUIERDO = 17
CORCHETE_DERECHO = 18
PUNTO_Y_COMA = 19
PARENTESIS_IZQUIERDO = 20
PARENTESIS_DERECHO = 21
LLAVE_IZQUIERDA = 22
LLAVE_DERECHA = 23
COMA = 24
COMENTARIO_MULTILINEA = 25
COMENTARIO_LINEA = 26
DECIMAL = 27
ERROR = 28
ULTIMO = 29

def __init__(self, lexema, tipo, fila, columna):
    self.lexema_valido = lexema
    self.tipo = tipo
    self.fila = fila
    self.columna = columna
```

- Método agregar_color

Este método es utilizado para guardar el color en la clase Color, para ello se solicitan los parámetros de x, y, pintar, código e índice_imagen, luego dentro del método se crea un nuevo color y se guarda en la lista de colores.

```
def agregar_color(self, x, y, pintar, codigo, indice_imagen):
    nuevo = Color(x, y, pintar, codigo, indice_imagen)
    self.colores.append(nuevo)
```

- Método get_tipo y get_tipo_sintactico

Este método es utilizado para obtener el tipo de token que esta guardado, se evalúa el tipo actual y se retorna la cadena con el nombre del tipo en cuestión.

```
def get_tipo(self):
    if self.tipo == self.CLAVES or self.tipo == self.REGISTROS or self.tipo == self.I
        return 'Palabra Reservada'
    elif self.tipo == self.CONTEO or self.tipo == self.PROMEDIO or self.tipo == self.
        return 'Palabra Reservada'
    elif self.tipo == self.SUMAR or self.tipo == self.MAX or self.tipo == self.MIN or
        return 'Palabra Reservada'
    elif self.tipo == self.LETRA:
        return 'Letra'
    elif self.tipo == self.NUMERO:
        return 'Número'
    elif self.tipo == self.CADENA:
        return 'Cadena'
    elif self.tipo == self.IGUAL or self.tipo == self.CORCHETE_IZQUIERDO or self.tipo
        return 'Signo'
    elif self.tipo == self.PUNTO_Y_COMA or self.tipo == self.PARENTESIS_IZQUIERDO or
        return 'Signo'
    elif self.tipo == self.COMA or self.tipo == self.LLAVE_DERECHA or self.tipo == se
        return 'Signo'
    elif self.tipo == self.COMENTARIO_MULTILINEA:
        return 'Comentario Multilinea'
    elif self.tipo == self.COMENTARIO_LINEA:
        return 'Comentario una Linea'
    elif self.tipo == self.DECIMAL:
        return 'Decimal'
    elif self.tipo == self.ULTIMO:
        return 'Ultimo'
```

```
def get_tipo_token_sintactico(self, tipo):
    if tipo == self.CLAVES:
        return 'Tk_Claves'
    elif tipo == self.REGISTROS:
        return 'Tk_Registro'
    elif tipo == self.IMPRIMIR:
        return 'Tk_Imprimir'
    elif tipo == self.IMPRIMIRLN:
        return 'Tk_ImprimirLn'
    elif tipo == self.CONTEO:
        return 'Tk_Conteo'
    elif tipo == self.PROMEDIO:
        return 'Tk_Promedio'
    elif tipo == self.CONTARSI:
        return 'Tk_ContarSi'
    elif tipo == self.DATOS:
        return 'Tk_DATOS'
    elif tipo == self.SUMAR:
        return 'Tk_Sumar'
    elif tipo == self.MAX:
        return 'Tk_Max'
    elif tipo == self.MIN:
        return 'Tk_Min'
    elif tipo == self.EXPORTARREPORTE:
        return 'Tk_Exportar_Reporte'
    elif tipo == self.LETRA:
        return 'Tk_Letra'
    elif tipo == self.NUMERO:
        return 'Tk_Numero'
    elif tipo == self.CADENA:
        return 'Tk_Cadena'
```

```
elif tipo == self.IGUAL:
    return 'Tk_Igual'
elif tipo == self.CORCHETE_IZQUIERDO:
    return 'Tk_Corchete_Izquierdo'
elif tipo == self.CORCHETE_DERECHO:
    return 'Tk_Corchete_Derecho'
elif tipo == self.PUNTO_Y_COMA:
    return 'Tk_Punto_y_Coma'
elif tipo == self.PARENTESIS_IZQUIERDO:
    return 'Tk_Parentesis_Izquierdo'
elif tipo == self.PARENTESIS_DERECHO:
    return 'Tk_Parentesis_Derecho'
elif tipo == self.COMA:
    return 'Tk_Coma'
elif tipo == self.LLAVE_DERECHA:
    return 'Tk_LLave_Derecha'
elif tipo == self.LLAVE_IZQUIERDA:
    return 'Tk_LLave_Izquierda'
elif tipo == self.COMENTARIO_MULTILINEA:
    return 'Comentario Multilinea'
elif tipo == self.COMENTARIO_LINEA:
    return 'Comentario una Linea'
elif tipo == self.DECIMAL:
    return 'Decimal'
elif tipo == self.ULTIMO:
    return 'Ultimo'
```

- Método get_lexema, get_fila, get_columna

Este método se usa para retornar el lexema, fila o columna del token actual.

```
def get_lexema(self):
    return self.lexema_valido

def get_fila(self):
    return self.fila

def get_columna(self):
    return self.columna
```

- Clase Analizador

Esta clase es la utilizada para generar analizar y llevar el control de los distintos métodos del programa, para ello se hacen los importes de las librerías necesarias como re, webbrowser, os y el importe de la clase Token. Luego se declaran las variables necesarias y la instancia de la clase.

```
from Token import Token
import re
import webbrowser
from os import makedirs

class Analizador():

    reporteHTML_token = ''
    reporteHTML_errores = ''
    lexema = ''
    estado = 0
    tokens = []
    columna = 1
    fila = 1
    tipos = Token("lexema", -1, -1, -1)
```

- Método agregar_token

Este método es utilizado para agregar un nuevo token al arreglo de token, para ello se crea un nuevo token y con la función append se guarda el nuevo token en la lista, luego se regresan a su estado original la variable lexema y estado.

```
def agregar_token(self, tipo):
    nuevo_token = Token(self.lexema, tipo, self.fila, self.columna)
    self.tokens.append(nuevo_token)
    self.lexema = ''
    self.estado = 0
```

- Método analizar_estados

Este método es utilizado para analizar los estados del archivo que se ha cargado para guárdalo en el token correspondiente, primero se hace la instancia del método separar, seguido se le suma a la entrada un carácter que servirá como identificador de que el archivo se ha terminado de leer. Se procede a realizar un ciclo for en el cual se recorre la longitud de la entrada, luego se igual a la variable actual a la entrada en la posición que indica el contador. Se evalúa si el estado es 0, si es así se procede a evaluar si es una letra, si lo es cambia el estado a 1, añade 1 a la columna y concatena el carácter actual a la variable lexema, si no se evalúa si es un dígito, si lo es cambia el estado a 2 y realiza los cambios pertinentes, si no evalúa si el carácter actual es una comilla, si lo es cambia el estado a 3 y aumenta los valores, si no se cumple verifica si el carácter actual es alguno de los signos reconocidos por el autómata si es así ingresa, modifica las variables y agrega el token, luego evalúa si es una comilla simple, si lo es cambia el estado a 5, suma una columna y concatena el lexema, se realiza lo mismo con el numeral, luego se evalúa si es un espacio, salto de línea, una tabulación y cambia las variables correspondiente, luego verifica si el carácter actual es igual al carácter de control o a la longitud de la cadena menos uno, si nada se cumple agrega el token como un error. Luego continua con los otros estados realizando las validaciones y moviéndose entre sí hasta formar el token en cuestión.

```
def analizador_estados(self, entrada):
    self.estado = 0
    self.lexema = ''
    self.tokens = []
    self.fila = 1
    self.columna = 1
    entrada = self.separar(entrada)
    entrada += '`'
    actual = ''
    longitud = len(entrada)
    for contador in range(longitud):
        actual = entrada[contador]
        if self.estado == 0:
            if actual.isalpha():
                self.estado = 1
                self.columna += 1
                self.lexema += actual

            elif actual.isdigit():
                self.estado = 2
                self.columna += 1
                self.lexema += actual

            elif actual == '"':
                self.estado = 3
                self.columna += 1
                self.lexema += actual

            elif actual == '=':
                self.columna += 1
                self.lexema += actual
                self.agregar_token(self.tipos.IGUAL)
```

```
elif actual == '[':
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.CORCHETE_IZQUIERDO)

elif actual == ']':
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.CORCHETE_DERECHO)

elif actual == '(':
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.PARENTESIS_IZQUIERDO)

elif actual == ')':
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.PARENTESIS_DERECHO)

elif actual == '{':
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.LLAVE_IZQUIERDA)

elif actual == '}':
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.LLAVE_DERECHA)
```

```
elif actual == ',':
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.COMA)

elif actual == ';':
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.PUNTO_Y_COMA)

elif actual == "'":
    self.estado = 5
    self.columna += 1
    self.lexema += actual

elif actual == '#':
    self.estado = 6
    self.columna += 1
    self.lexema += actual

elif actual == ' ':
    self.columna +=1
    self.estado = 0

elif actual == '\n':
    self.fila += 1
    self.estado = 0
    self.columna = 1

elif actual =='\r':
    self.estado = 0
```

```
    elif actual == '\t':
        self.columna += 5
        self.estado = 0

    elif actual == '`' and contador == longitud - 1:
        self.lexema = '`'
        self.agregar_token(self.tipos.ULTIMO)
        print('Análisis terminado')

    else:
        self.lexema += actual
        self.agregar_token(self.tipos.ERROR)
        self.columna += 1

elif self.estado == 1:
    if actual.isalpha():
        self.estado = 1
        self.columna += 1
        self.lexema += actual
    else:
        if not self.es_palabra_reserva(self.lexema):
            self.agregar_token(self.tipos.ERROR)

elif self.estado == 2:
    if actual.isdigit():
        self.estado = 2
        self.columna += 1
        self.lexema += actual
    elif actual == '.':
        self.estado = 7
        self.columna += 1
        self.lexema += actual
```

```
    else:
        self.agregar_token(self.tipos.NUMERO)

elif self.estado == 3:
    if actual != '"':
        self.estado = 3
        self.columna +=1
        self.lexema += actual

    elif actual == '"':
        self.estado = 8
        self.lexema += actual
        self.agregar_token(self.tipos.CADENA)

elif self.estado == 5:
    if actual == "'":
        self.estado = 9
        self.columna += 1
        self.lexema += actual

    else:
        self.agregar_token(self.tipos.ERROR)

elif self.estado == 6:
    if actual == '\n':
        self.fila += 1
        self.estado = 0
        self.columna = 1
        self.agregar_token(self.tipos.COMENTARIO_LINEA)
```

```
        else:
            self.estado = 6
            self.columna += 1
            self.lexema += actual

    elif self.estado == 7:
        if actual.isdigit():
            self.estado = 10
            self.columna += 1
            self.lexema += actual
        else:
            self.agregar_token(self.tipos.ERROR)

    elif self.estado == 9:
        if actual == "'":
            self.estado = 11
            self.columna += 1
            self.lexema += actual
        else:
            self.agregar_token(self.tipos.ERROR)

    elif self.estado == 10:
        if actual.isdigit():
            self.estado = 10
            self.columna += 1
            self.lexema += actual
        else:
            self.agregar_token(self.tipos.DECIMAL)
```

```
elif self.estado == 11:
    if actual != "'":
        self.estado = 11
        self.columna += 1
        self.lexema += actual
    else:
        self.estado = 12
        self.columna += 1
        self.lexema += actual

elif self.estado == 12:
    if actual == "'":
        self.estado = 13
        self.columna += 1
        self.lexema += actual
    else:
        self.agregar_token(self.tipos.ERROR)

elif self.estado == 13:
    if actual == "'":
        self.estado = 14
        self.columna += 1
        self.lexema += actual
        self.agregar_token(self.tipos.COMENTARIO_MULTILINEA)
    else:
        self.agregar_token(self.tipos.ERROR)
```

- Método separar

Este método es utilizado para crear un espacio entre los signos y los caracteres, para ello se hace uso de la librería re, en ella se declaran los signos que se desean valorar y se devuelve en la nueva entrada.

```
def separar(self, entrada):
    patron = r'(\w)([= - { - } - , - ; - ( - ) - \[ - \] ])'
    return re.sub(patron, r'\1 \2 ', entrada)
```

- Método es_palabra_reservada

Este método se usa para saber el lexema encontrado es una palabra reservada para ello la entrada se cambia a mayúsculas y se declara una lista con las palabras reservada, luego se evalúa si la entrada se encuentra en la lista si es así retorna True sino False.

```
def es_palabra_reserva(self, entrada):
    entrada = entrada.upper()
    if entrada == 'CLAVES':
        self.agregar_token(self.tipos.CLAVES)
        return True
    if entrada == 'REGISTROS':
        self.agregar_token(self.tipos.REGISTROS)
        return True
    if entrada == 'IMPRIMIR':
        self.agregar_token(self.tipos.IMPRIMIR)
        return True
    if entrada == 'IMPRIMIRLN':
        self.agregar_token(self.tipos.IMPRIMIRLN)
        return True
    if entrada == 'CONTEO':
        self.agregar_token(self.tipos.CONTEO)
        return True
    if entrada == 'PROMEDIO':
        self.agregar_token(self.tipos.PROMEDIO)
        return True
    if entrada == 'CONTARSI':
        self.agregar_token(self.tipos.CONTARSI)
        return True
    if entrada == 'DATOS':
        self.agregar_token(self.tipos.DATOS)
        return True
    if entrada == 'SUMAR':
        self.agregar_token(self.tipos.SUMAR)
        return True
    if entrada == 'MAX':
        self.agregar_token(self.tipos.MAX)
        return True
    if entrada == 'MIN':
        self.agregar_token(self.tipos.MIN)
        return True
    if entrada == 'EXPORTARREPORTE':
        self.agregar_token(self.tipos.EXPORTARREPORTE)
        return True
    return False
```

- Método reiniciar_tokens

Este método se usa para borrar el contenido de la lista de tokens

```
def reiniciar_tokens(self):
    self.tokens.clear()
```

- Método obtener_tokens

Este método se usa para guardar en una variable el listado de tokens para el reporte de tokens para ello se recorre la lista de tokens y se verifica que el token actual no es de tipo de error, si esto se cumple se guarda en la variable los atributos necesarios.

```
def obtener_tokens(self):
    font = '<font color=\"#000000\" face=\"Courier\">'
    for x in self.tokens:
        if x.tipo != self.tipos.ERROR:
            self.reporteHTML_token += '<tr><td align=center>'+ font + x.get_tipo() + '</td><td align=center>
```

- Método obtener_errores

Este método se usa para guardar en una variable el listado de errores para el reporte de errores para ello se recorre la lista de tokens y se verifica que el token actual es de tipo de error, si esto se cumple se guarda en la variable los atributos necesarios.

```
def obtener_errores_lexico(self):
    font = '<font color=\"#000000\" face=\"Courier\">'
    for x in self.tokens:
        if x.tipo == self.tipos.ERROR:
            self.reporteHTML_errores += '<tr><td align=center>'+ font + x.get_lexema() + '</td><td align=center>'+
```

- Método crear_reporte_token, crear_reporte_errores

Estos métodos son usados para crear los reportes en HTML para ello se accede al método en cuestión. Dentro del mismo se hace la instancia del método para obtener la información de los tokens y se crea una carpeta donde se guardarán los reportes. Luego dentro de un try – except, se crea una variable file, se abre el archivo con el método open, luego con otras auxiliares, se crea la estructura del HTML y se concatena la variable en cuestión. Luego a través del método write se escribe el archivo, al finalizar se cierra el archivo y se hace uso del método implementado por webbrowser open_new_tab para abrir el reporte después de generarlo.

```
def crear_reporte_token(self):
    self.obtener_tokens()
    makedirs('Reportes', exist_ok = True)
    try:
        file = open('Reportes/Reporte_Tokens.html','w')
        head = '<head><title>Reporte Token</title></head>\n'
        body = '''<body bgcolor=\"#B6F49D\">
                <table width=\"400\" bgcolor=#B6F49D align=left> <tr> <td><font color=\"black\" FACE=\"Courier\">
                <p align=\"left\">Arnoldo Luis Antonio González Camey  –  Carné: 201701548</p></font>
                </td> </tr></table></br></br>
                <h2 align=\"center\"><font color=\"black\" FACE=\"Courier\">Tabla de Tokens</h2>
                <table width=\"1000\" bgcolor=#CDF9BA align=center style="border:5px dashed brown">
                <tr>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Token</strong></td>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Lexema</strong></td>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Fila</strong></td>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Columna</strong></td>
                </tr>'''
        body += self.reporteHTML_token +'</table></body>'
        html = '<html>\n' + head + body + '</html>'
        file.write(html)
        print('Reporte de Tokens generado exitosamente')
    except OSError:
        print("Error al crear el Reporte de Tokens")
    finally:
        file.close()
        webbrowser.open_new_tab('Reportes\Reporte_Tokens.html')
```

```
def crear_reporte_errores(self):
    self.obtener_errores_lexico()
    makedirs('Reportes', exist_ok = True)
    try:
        file = open('Reportes/Reporte_Errores_Lexico.html','w')
        head = '<head><title>Reporte Errores</title></head>\n'
        body = '''<body bgcolor=\"#B6F49D\">
                <table width=\"600\" bgcolor=#B6F49D align=left> <tr> <td><font color=\"black\" FACE=\"Courier\">
                <p align=\"left\">Arnoldo Luis Antonio González Camey  —  Carné: 201701548</p></font>
                </td> </tr></table></br></br>
                <h2 align=\"center\"><font color=\"black\" FACE=\"Courier\">Tabla de Errores Léxicos</h2>
                <table width=\"800\" bgcolor=#CDF9BA align=center style="border:5px dashed brown">
                <tr>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Caracter</strong></td>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Fila</strong></td>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Columna</strong></td>
                </tr>'''
        body += self.reporteHTML_errores +'</table></body>'
        html = '<html>\n' + head + body + '</html>'
        file.write(html)
        print('Reporte de Errores Léxicos generado exitosamente')
    except OSError:
        print("Error al crear el Reporte de Errores Léxicos")
    finally:
        file.close()
        webbrowser.open_new_tab('Reportes\Reporte_Errores_Lexico.html')
```

- Clase Sintactico

Esta clase es la utilizada para realizar el análisis sintáctico y llevar el control de los distintos métodos del programa, para ello se hacen los importes de las librerías necesarias como re, webbrowser, os y el importe de la clase Token. Luego se declaran las variables necesarias y la instancia de la clase.

```
from Token import Token
from Registro import Registro
from Clave import Clave
import webbrowser
from Arboles import Arboles
from os import makedirs
```

```
class Sintactico:

    arboles = Arboles()
    tipos = Token('',-1,-1,-1)
    preanalisis = Token.ERROR
    posicion = 0
    lista = []
    tamano_val_registros = 0
    contador_registros = 0
    valor_actual = 0
    indice_clave = 0
    entro_reg = False
    errorSintactico = False
    valores_registro = []
    valores_clave = []
    registros = Registro(-1)
    reporteHTML_registro = ''
    reporte_error_sintac = ''
    nombres_arboles = []
```

```
    def __init__(self, tkinter, lista, txt_consola):
        self.errorSintactico = False
        self.lista = lista
        self.tkinter = tkinter
        self.txt_consola = txt_consola
        self.lista.append(Token('`', Token.ULTIMO, -1, -1))
        self.posicion = 0
        self.preanalisis = self.lista[self.posicion].tipo
        self.Inicio()
```

- Método Match

Este método es utilizado para conocer si el token que se va a evaluar es el que espera la estructura, para ello se valida si el preanálisis no es igual al tipo esperado si no lo es, entra y guarda el error sintáctico, luego verifica si es preanálisis no es igual al ultimo token, si no lo es aumenta la posición e igual el nuevo valor del preanálisis, si es igual imprime en la consola que el análisis ha finalizado.

```
def Match(self, tipo):
    if self.preanalisis != tipo:
        font = '<font color=\"#000000\" face=\"Courier\">'
        print(self.lista[self.posicion].lexema_valido, "--> Sintactico", " -- Se esperaba "+str(self.tipo_token(tipo)))
        self.reporte_error_sintac += '<tr><td align=center>'+ font + self.lista[self.posicion].lexema_valido + ' --> Se
        self.reporte_error_sintac += '</td><td align=center><font color=\"#155CFF\" face=\"Courier\">Error Sintáctico <
        self.reporte_error_sintac += '</td><td align=center>'+ font + str(self.lista[self.posicion].columna) + '</td></
        self.errorSintactico = True

    if self.preanalisis != Token.ULTIMO:
        self.posicion += 1
        self.preanalisis = self.lista[self.posicion].tipo

    if self.preanalisis == Token.ULTIMO:
        print('Se ha finalizado el analisis sintactico')
```

- Método tipo_token

Este método se usa para retornar el nombre del token

```
def tipo_token(self, tipo):
    nombre_Tk = self.tipos.get_tipo_token_sintactico(tipo)
    return nombre_Tk
```

- Método Inicio

Este método es utilizado como un menú para validar el token siguiente para ello dentro de unas sentencias if y elif se evalua cual es el token en cuestión, al encontrarlo se hace la instancia del método correspondiente y el método repetir.

```
def Inicio(self):
    print('Inicio del analisis Sintáctico')
    if Token.CLAVES == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<CLAVES>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Claves()
        self.Repetir()
    elif Token.REGISTROS == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REGISTROS>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Registros()
        self.Repetir()
    elif Token.COMENTARIO_LINEA == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<COMENTARIO_LINEA>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Comentario()
        self.Repetir()
    elif Token.COMENTARIO_MULTILINEA == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<COMENTARIO_MULTILINEA>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Comentario_Multilinea()
        self.Repetir()
    elif Token.IMPRIMIR == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<IMPRIMIR>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Imprimir()
        self.Repetir()
    elif Token.IMPRIMIRLN == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<IMPRIMIRLN>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.ImprimirLn()
        self.Repetir()
```

```
    elif Token.CONTEO == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<CONTEO>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Conteo()
        self.Repetir()
    elif Token.PROMEDIO == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<PROMEDIO>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Promedio()
        self.Repetir()
    elif Token.CONTARSI == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<CONTARSI>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.ContarSi()
        self.Repetir()
    elif Token.DATOS == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<DATOS>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Datos()
        self.Repetir()
    elif Token.SUMAR == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<SUMAR>"];\n'
        self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Sumar()
        self.Repetir()
```

```
elif Token.MAX == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<MAX>"];\n'
    self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Max()
    self.Repetir()
elif Token.MIN == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<MIN>"];\n'
    self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Min()
    self.Repetir()
elif Token.EXPORTARREPORTE == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<EXPORTAR_REPORTE>"];\n'
    self.contenido_inicio += 'n0 -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Exportar_Reporte()
    self.Repetir()
```

- Método Claves

Este método es usado para corroborar la estructura de las claves, para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado, luego se hace la instancia del método Cuerpo_Claves para continuar con la estructura de este. A su vez se concatenan los valores para poder realizar el árbol de derivación. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Claves(self):
    self.contenido_claves = ''
    self.contador_claves = 4
    self.Match(Token.CLAVES)
    self.Match(Token.IGUAL)
    self.Match(Token.CORCHETE_IZQUIERDO)
    self.Cuerpo_Claves()
    self.contenido_claves += 'n'+str(self.contador_claves)+'[label = "Tk_Crch_Dr"];\n'
    self.contenido_claves += 'raiz -> n'+str(self.contador_claves)+';'
    self.Match(Token.CORCHETE_DERECHO)

    if not self.repetido('Arbol Claves'):
        self.nombres_arboles.append('Arbol Claves')
```

- Método Cuerpo_Claves

Este método es usado para corroborar la estructura del cuerpo de las claves, para ello se hace verifica si el token siguiente es una cadena, si lo se guarda en la lista de claves el nombre de la clave y se realiza la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado, luego se valida si el token que sigue es de tipo coma, si lo es se hace la instancia de match y se vuelve a llamar este método. A su vez se concatenan los valores para poder realizar el árbol de derivación.

```
def Cuerpo_Claves(self):
    if Token.CADENA == self.preanalisis:
        nombre = self.lista[self.posicion].lexema_valido
        nombre = nombre.replace('"','')
        nuevo = Clave(self.indice_clave, nombre)
        self.valores_clave.append(nuevo)
        self.indice_clave += 1

        self.contenido_claves += 'n'+str(self.contador_claves)+'[label = "<BLOQUE_CLAVES>"];\n'
        self.contenido_claves += 'n'+str(self.contador_claves)+' -> n'+str(self.contador_claves + 1)+';\n'
        self.contador_claves += 1
        self.contenido_claves += 'n'+str(self.contador_claves)+'[label = "Tk_Cadena"];\n'
        self.contador_claves += 1

        self.Match(Token.CADENA)
    if Token.COMA == self.preanalisis:
        self.contenido_claves += 'n'+str(self.contador_claves)+'[label = "Tk_Coma"];\n'
        self.contenido_claves += 'n'+str(self.contador_claves - 2)+' -> n'+str(self.contador_claves)+';\n'
        self.contenido_claves += 'n'+str(self.contador_claves - 2)+' -> n'+str(self.contador_claves + 1)+';\n'
        self.contador_claves += 1
        self.Match(Token.COMA)
        self.Cuerpo_Claves()
```

- Método Registros

Este método es usado para corroborar la estructura de los registros, para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado, luego se hace la instancia del método Bloque_Registros para continuar con la estructura de este. A su vez se concatenan los valores para poder realizar el árbol de derivación. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Registros(self):
    self.contenido_registros = ''
    self.contador_registros = 4
    self.Match(Token.REGISTROS)
    self.Match(Token.IGUAL)
    self.Match(Token.CORCHETE_IZQUIERDO)
    self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "<BLOQUE_REGISTROS>"];\n'
    self.contador_registros += 1
    self.Bloque_Registros()
    self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "Tk_Crch_Der"];\n'
    self.contenido_registros += 'raiz -> n'+str(self.contador_registros)+';\n'
    self.contador_registros += 1
    self.Match(Token.CORCHETE_DERECHO)
    if not self.repetido('Arbol Registros'):
        self.nombres_arboles.append('Arbol Registros')
```

- Método Bloque_Registros

Este método es usado para corroborar la estructura del bloque de registros, para ello se hace verifica si el token siguiente es de tipo llave izquierda, si lo se hace la instancia del método Cuerpo_Registros y de este mismo método. A su vez se concatenan los valores para poder realizar el árbol de derivación.

```
def Bloque_Registros(self):
    if Token.LLAVE_IZQUIERDA == self.preanalisis:
        tmp = self.contador_registros
        self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "<CUERPO_REGISTROS>"];\n'
        self.contenido_registros += 'n'+str(self.contador_registros - 1)+' -> n'+str(self.contador_registros)+';\n'
        self.contador_registros += 1
        self.Cuerpo_Registros()
        if Token.LLAVE_IZQUIERDA == self.preanalisis:
            self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "<BLOQUE_REGISTROS>"];\n'
            self.contenido_registros += 'n'+str(tmp - 1)+' -> n'+str(self.contador_registros)+';\n'
            self.contador_registros += 1
        self.Bloque_Registros()
```

- Método Cuerpo_Registros

Este método es usado para corroborar la estructura del cuerpo de los registros, para ello se hace verifica si el tipo de token siguiente es una llave izquierda, si lo se hace la instancia del método valor_registro y del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado, luego se instancia el método guardar_registro. A su vez se concatenan los valores para poder realizar el árbol de derivación.

```
def Cuerpo_Registros(self):
    tmp = self.contador_registros
    self.Match(Token.LLAVE_IZQUIERDA)
    self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "Tk_Llave_Izq"];\n'
    self.contenido_registros += 'n'+str(self.contador_registros - 1)+' -> n'+str(self.contador_registros)+';\n'
    self.contador_registros += 1

    self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "<VALOR_REGISTROS>"];\n'
    self.contenido_registros += 'n'+str(self.contador_registros - 2)+' -> n'+str(self.contador_registros)+';\n'
    self.contador_registros += 1

    self.valor_registro()
    self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "Tk_Llave_Der"];\n'
    self.contenido_registros += 'n'+str(tmp - 1)+' -> n'+str(self.contador_registros)+';\n'
    self.contador_registros += 1
    self.Match(Token.LLAVE_DERECHA)
    self.guardar_registro()
```

- Método valor_registro

Este método es usado para encontrar los datos dentro de la estructura del registro, para ello se verifica por medio de una sentencia if si el tipo de token que sigue es número, cadena o decimal, si lo es entra guarda dentro de la lista valores_registro el dato y hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado, luego se verifica si el tipo de token siguiente es una coma, si lo es ingresa y hace la instancia del Match luego vuelve a realizar la instancia de este método como una llamada recursiva. A su vez se concatenan los valores para poder realizar el árbol de derivación.

```
def valor_registro(self):
    if Token.NUMERO == self.preanalisis:
        nuevo = self.lista[self.posicion].lexema_valido
        self.valores_registro.append(nuevo)
        self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "Tk_Numero"];\n'
        self.contenido_registros += 'n'+str(self.contador_registros - 1)+' -> n'+str(self.contador_registros)+';\n'
        self.contador_registros += 1
        self.Match(Token.NUMERO)
    elif Token.CADENA == self.preanalisis:
        nuevo = self.lista[self.posicion].lexema_valido
        nuevo = nuevo.replace('"','')
        self.valores_registro.append(nuevo)
        self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "Tk_Cadena"];\n'
        self.contenido_registros += 'n'+str(self.contador_registros - 1)+' -> n'+str(self.contador_registros)+';\n'
        self.contador_registros += 1
        self.Match(Token.CADENA)
    elif Token.DECIMAL == self.preanalisis:
        nuevo = self.lista[self.posicion].lexema_valido
        self.valores_registro.append(nuevo)
        self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "Tk_Decimal"];\n'
        self.contenido_registros += 'n'+str(self.contador_registros - 1)+' -> n'+str(self.contador_registros)+';\n'
        self.contador_registros += 1
        self.Match(Token.DECIMAL)
    if Token.COMA == self.preanalisis:
        self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "Tk_Coma"];\n'
        self.contenido_registros += 'n'+str(self.contador_registros - 2)+' -> n'+str(self.contador_registros)+';\n'
        self.contador_registros += 1
        self.contenido_registros += 'n'+str(self.contador_registros)+'[label = "<VALOR_REGISTROS>"];\n'
        self.contenido_registros += 'n'+str(self.contador_registros - 3)+' -> n'+str(self.contador_registros)+';\n'
        self.contador_registros += 1
        self.Match(Token.COMA)
        self.valor_registro()
```

- Método guardar_registro

Este método es usado para guardar un registro en la Clase registro, para ello se evalúa si ya se guardo la dimensión de la lista, luego guarda en una lista temporal el intervalo de datos que se desea almacenar, luego se hace la instancia al método agregar_registro de la clase Registro y se envían los parámetros correspondientes y se hacen las nuevas validaciones.

```
def guardar_registro(self):
    if not self.entro_reg:
        self.tamano_val_registros = len(self.valores_registro)
        self.entro_reg = True
    values = self.valores_registro[self.valor_actual:self.valor_actual + self.tamano_val_registros]
    self.registros.agregar_registro(self.contador_registros, values)
    self.contador_registros += 1
    self.valor_actual += self.tamano_val_registros
```

- Método Comentario

Este método es usado para corroborar la estructura de los comentarios de una línea, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Comentario(self):
    self.Match(Token.COMENTARIO_LINEA)
    if not self.repetido('Arbol Comentario Linea'):
        self.nombres_arboles.append('Arbol Comentario Linea')
```

- Método Comentario_Multilinea

Este método es usado para corroborar la estructura de los comentarios de múltiples líneas, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Comentario_Multilinea(self):
    self.Match(Token.COMENTARIO_MULTILINEA)
    if not self.repetido('Arbol Comentario MultiLinea'):
        self.nombres_arboles.append('Arbol Comentario MultiLinea')
```

- Método Imprimir

Este método es usado para corroborar la estructura de la impresión, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se imprime en la interfaz gráfica el texto obtenido. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Imprimir(self):
    self.Match(Token.IMPRIMIR)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    print_consola = self.lista[self.posicion].lexema_valido
    self.Match(Token.CADENA)
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)

    print_consola = print_consola.replace('"','')
    self.txt_consola.insert(self.tkinter.INSERT, print_consola)
    if not self.repetido('Arbol Imprimir'):
        self.nombres_arboles.append('Arbol Imprimir')
```

- Método ImprimirLn

Este método es usado para corroborar la estructura de la impresión con salto de línea, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se imprime en la interfaz gráfica el texto obtenido. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def ImprimirLn(self):
    self.Match(Token.IMPRIMIRLN)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    print_consola = self.lista[self.posicion].lexema_valido
    self.Match(Token.CADENA)
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)

    print_consola = print_consola.replace('"','')
    self.txt_consola.insert(self.tkinter.INSERT, '\n' + print_consola + '\n')
    if not self.repetido('Arbol ImprimirLn'):
        self.nombres_arboles.append('Arbol ImprimirLn')
```

- Método Conteo

Este método es usado para corroborar la estructura de la función conteo, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se imprime en la interfaz gráfica el valor obtenido. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Conteo(self):
    self.Match(Token.CONTEO)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)

    print_consola = len(self.registros.valores)
    self.txt_consola.insert(self.tkinter.INSERT, '>>> ' + str(print_consola))
    if not self.repetido('Arbol Conteo'):
        self.nombres_arboles.append('Arbol Conteo')
```

- Método Promedio

Este método es usado para corroborar la estructura de la función promedio, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se hace la instancia del método obtener_promedio_suma en el cual se manda como parámetro el valor obtenido con anterioridad. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Promedio(self):
    self.Match(Token.PROMEDIO)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    campo = self.lista[self.posicion].lexema_valido
    self.Match(Token.CADENA)
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)
    campo = campo.replace('"','')
    self.obtener_promedio_suma(campo, True)
    if not self.repetido('Arbol Promedio'):
        self.nombres_arboles.append('Arbol Promedio')
```

- Método obtener_promedio_suma

Este método es usado para encontrar el promedio y la suma de los datos en el campo deseado para ello por medio de un ciclo for que recorre la lista de valores_clave se obtiene el índice del campo deseado, luego se evalúa si el índice no es menos uno, se realiza otro ciclo el cual evalúa los valores de la clase registros y concatena la suma en el índice desea, luego se procede a realizar el promedio e imprimirlo en la interfaz, si lo que se desea es imprimir la suma se realiza el else.

```
def obtener_promedio_suma(self, campo, es_promedio):
    indice = -1
    suma = 0
    for claves in self.valores_clave:
        if claves.get_nombre() == campo:
            indice = claves.get_indice()
            break
    if indice != -1:
        for i in range(len(self.registros.valores)):
            suma += int(self.registros.valores[i].args[0][indice])
    if es_promedio and len(self.registros.valores) != 0:
        promedio = suma/len(self.registros.valores)
        self.txt_consola.insert(self.tkinter.INSERT, '>>> ' + str(promedio))
    else:
        self.txt_consola.insert(self.tkinter.INSERT, '>>> ' + str(suma))
```

- Método ContarSi

Este método es usado para corroborar la estructura de la función contarsi, para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se hace la instancia del método valor_contarSi y obtener_contar_si en el cual se manda como parámetro el valor obtenido con anterioridad. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def ContarSi(self):
    self.Match(Token.CONTARSI)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    campo = self.lista[self.posicion].lexema_valido
    self.Match(Token.CADENA)
    self.Match(Token.COMA)
    valor = self.valor_contarSi()
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)
    campo = campo.replace('"','')
    self.obtener_contar_si(campo, valor)
    if not self.repetido('Arbol ContarSi'):
        self.nombres_arboles.append('Arbol ContarSi')
```

- Método ContarSi

Este método es usado para obtener y corroborar la estructura de la función contarsi, para ello se verifica si el tipo de token siguiente es de tipo Numero, Decimal o Cadena, si lo es ingresa y hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado y retorna el valor obtenido.

```
def valor_contarSi(self):
    if Token.NUMERO == self.preanalisis:
        valor = self.lista[self.posicion].lexema_valido
        valor = int(valor)
        self.valor_contarsi = valor
        self.Match(Token.NUMERO)
        return valor
    elif Token.DECIMAL == self.preanalisis:
        valor = self.lista[self.posicion].lexema_valido
        valor = float(valor)
        self.valor_contarsi = valor
        self.Match(Token.DECIMAL)
        return valor
    elif Token.CADENA == self.preanalisis:
        dato: str = self.lista[self.posicion].lexema_valido
        dato = dato.replace('"',"")
        self.valor_contarsi = dato
        self.Match(Token.CADENA)
        return dato
```

- Método obtener_contar si

Este método es usado para obtener el número de veces que se encuentra repetida el valor pedido, para ello mediante un ciclo el cual recorre la lista de valores_clave y encuentra el índice necesario, luego se recorre los registros y se valida de que tipo es el parámetro y se aumenta un contador, luego se imprime en la interfaz gráfica el valor obtenido por el contador.

```
def obtener_contar_si(self, campo, valor):
    indice = -1
    contador = 0
    for claves in self.valores_clave:
        if claves.get_nombre() == campo:
            indice = claves.get_indice()
            break
    if indice != -1:
        for i in range(len(self.registros.valores)):
            if type(valor) == float:
                if valor == float(self.registros.valores[i].args[0][indice]):
                    contador += 1
            elif type(valor) == int:
                if valor == int(self.registros.valores[i].args[0][indice]):
                    contador += 1
            elif type(valor) == str:
                if valor.replace(" ","").upper() == self.registros.valores[i].args[0][indice].replace(" ","").upper():
                    contador += 1
        self.txt_consola.insert(self.tkinter.INSERT, '>>> ' + str(contador)+'\n')
```

- Método Datos

Este método es usado para corroborar la estructura de la función datos, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se hace la instancia del método obtener_datos en el cual se manda como parámetro falso. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Datos(self):
    self.Match(Token.DATOS)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)
    self.obtener_datos(False)
    if not self.repetido('Arbol Datos'):
        self.nombres_arboles.append('Arbol Datos')
```

- Método obtener_datos

Este método es usado para guardar en una variable la información para realizar el reporte en html de los datos del registro, para ello se realiza un ciclo for el cual recorre los valores_clave, y se guardan como los encabezados, luego se realiza un for anidado en el cual se recorren los registros en el índice correspondiente y se concatena la información, si el parámetro es falso se imprime en consola la información obtenida.

```
def obtener_datos(self, es_reporte):
    self.reporteHTML_registro += '<tr>'
    contenido = ''
    contador = 0
    for claves in self.valores_clave:
        contenido += claves.get_nombre().replace('_',' ')
        self.reporteHTML_registro += '<td align=center><font color=\"#000000\" f
        contador += 1
        if contador != len(self.valores_clave):
            contenido += ' - '
    contenido += '\n'
    contador = 0
    self.reporteHTML_registro += '</tr>'

    for i in range(len(self.registros.valores)):
        self.reporteHTML_registro += '<tr>'
        for claves in self.valores_clave:
            contenido += self.registros.valores[i].args[0][claves.get_indice()]
            self.reporteHTML_registro += '<td align=center><font color=\"#000000
            contador += 1
            if contador != len(self.valores_clave):
                contenido += ' - '
            else:
                contador = 0
        self.reporteHTML_registro += '</tr>'
        contenido += '\n'

    if not es_reporte:
        self.txt_consola.insert(self.tkinter.INSERT, contenido+'\n')
        self.reporteHTML_registro = ''
```

- Método Sumar

Este método es usado para corroborar la estructura de la función sumar, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se hace la instancia del método obtener_promedio_suma en el cual se manda como parámetro el valor obtenido y falso. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Sumar(self):
    self.Match(Token.SUMAR)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    campo = self.lista[self.posicion].lexema_valido
    self.Match(Token.CADENA)
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)
    campo = campo.replace('"','')
    self.obtener_promedio_suma(campo, False)
    if not self.repetido('Arbol Sumar'):
        self.nombres_arboles.append('Arbol Sumar')
```

- Método Max

Este método es usado para corroborar la estructura de la función max, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se hace la instancia del método obtener_max_min en el cual se manda como parámetro el valor obtenido y falso. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Max(self):
    self.Match(Token.MAX)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    campo = self.lista[self.posicion].lexema_valido
    self.Match(Token.CADENA)
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)
    campo = campo.replace('"','')
    self.obtener_max_min(campo, True)
    if not self.repetido('Arbol Max'):
        self.nombres_arboles.append('Arbol Max')
```

- Método obtener_max_min

Este método es usado para obtener el valor máximo o mínimo del campo deseado, para ello se recorre la lista de valores_clalve y se encuentra el índice del campo en cuestión. Luego se recorren los registros y mediante un temporal se valida si el nuevo dato es mayor o menor al anterior y así se obtiene el máximo o el mínimo. Al finalizar se imprime en la interfaz el valor obtenido.

```
def obtener_max_min(self, campo, es_max):
    indice = -1
    maximo = 0
    minimo = 100000
    tmp = 0
    for claves in self.valores_clave:
        if claves.get_nombre() == campo:
            indice = claves.get_indice()
            break
    if indice != -1:
        for i in range(len(self.registros.valores)):
            tmp = float(self.registros.valores[i].args[0][indice])
            if tmp > maximo:
                maximo = tmp
            if tmp < minimo:
                minimo = tmp
    if es_max:
        self.txt_consola.insert(self.tkinter.INSERT, '>>> ' + str(maximo))
    else:
        self.txt_consola.insert(self.tkinter.INSERT, '>>> ' + str(minimo))
```

- Método Min

Este método es usado para corroborar la estructura de la función min, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se hace la instancia del método obtener_max_min en el cual se manda como parámetro el valor obtenido y falso. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Min(self):
    self.Match(Token.MIN)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    campo = self.lista[self.posicion].lexema_valido
    self.Match(Token.CADENA)
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)
    campo = campo.replace('"','')
    self.obtener_max_min(campo, False)
    if not self.repetido('Arbol Min'):
        self.nombres_arboles.append('Arbol Min')
```

- Método Exportar_Reporte

Este método es usado para corroborar la estructura de la función exportarReporte, para para ello se hace la instancia del método Match en el cual se le manda de parámetro el tipo de Token que se espera encontrar y en el orden deseado. Luego se hace la instancia del método crear_reporte_registro en el cual se manda como parámetro el valor obtenido. Al finalizar se evalúa si el nombre para el combobox ya ha sido guardado.

```
def Exportar_Reporte(self):
    self.Match(Token.EXPORTARREPORTE)
    self.Match(Token.PARENTESIS_IZQUIERDO)
    nombre = self.lista[self.posicion].lexema_valido
    self.Match(Token.CADENA)
    self.Match(Token.PARENTESIS_DERECHO)
    self.Match(Token.PUNTO_Y_COMA)
    nombre = nombre.replace('"','')
    self.crear_reporte_registro(nombre)
    if not self.repetido('Arbol Exportar Reporte'):
        self.nombres_arboles.append('Arbol Exportar Reporte')
```

- Método reiniciar

Este método es usado para borrar el contenido de los arreglos y de los registros.

```
def reniciar(self):
    self.lista.clear()
    self.valores_registro.clear()
    self.registros.reiniciar_registro()
    self.valores_clave.clear()
    self.nombres_arboles.clear()
```

- Método opciones_reporte_arbol

Este método es utilizado para guardar las opciones de los nombres de los árboles en el combobox de la interfaz.

```
def opciones_reporte_arbol(self, combo_reportes):
    combo_reportes["values"] = ["Seleccione el Reporte", "Reporte Tokens", "Reporte Errores Léxico", "Reporte Error Sintáctico"]
    values = list(combo_reportes["values"])
    combo_reportes["values"] = values + self.nombres_arboles
```

- Método repetido

Este método es utilizado para saber si un nombre ya ha sido guardado en la lista nombres_arboles, para ello se recorre el arreglo y si hace una validación si es cierta retorna verdadero, si no al final retorna falso.

```
def repetido(self, nombre_entrada):
    for nombres in self.nombres_arboles:
        if nombres == nombre_entrada:
            return True
    return False
```

- Método Repetir

Este método es utilizado como un menú para validar el token siguiente para ello dentro de unas sentencias if y elif se evalua cual es el token en cuestión, al encontrarlo se hace la instancia del método correspondiente y este mismo método. A su vez se concatenan los valores para poder realizar el árbol de derivación.

```
def Repetir(self):
    if Token.CLAVES == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
        self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<CLAVES>"];\n'
        self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Claves()
        self.Repetir()
    elif Token.REGISTROS == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
        self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REGISTROS>"];\n'
        self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Registros()
        self.Repetir()
    elif Token.COMENTARIO_LINEA == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
        self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<C0MENTARIO_LINEA>"];\n'
        self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Comentario()
        self.Repetir()
    elif Token.COMENTARIO_MULTILINEA == self.preanalisis:
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
        self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<COMENTARIO_MULTILINEA>"];\n'
        self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
        self.contador_inicio += 1
        self.Comentario_Multilinea()
        self.Repetir()
```

```
elif Token.IMPRIMIR == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<IMPRIMIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Imprimir()
    self.Repetir()
elif Token.IMPRIMIRLN == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<IMPRIMIRLN>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.ImprimirLn()
    self.Repetir()
elif Token.CONTEO == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<CONTEO>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Conteo()
    self.Repetir()
elif Token.PROMEDIO == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<PROMEDIO>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Promedio()
    self.Repetir()
```

```
elif Token.CONTARSI == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<CONTARSI>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.ContarSi()
    self.Repetir()
elif Token.DATOS == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<DATOS>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Datos()
    self.Repetir()
elif Token.SUMAR == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<SUMAR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Sumar()
    self.Repetir()
elif Token.MAX == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<MAX>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Max()
    self.Repetir()
```

```
elif Token.MIN == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<MIN>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Min()
    self.Repetir()
elif Token.EXPORTARREPORTE == self.preanalisis:
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<REPETIR>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 2)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.contenido_inicio += 'n'+str(self.contador_inicio)+'[label = "<EXTRAERREPORTE>"];\n'
    self.contenido_inicio += 'n'+str(self.contador_inicio - 1)+' -> n'+str(self.contador_inicio)+';\n'
    self.contador_inicio += 1
    self.Exportar_Reporte()
    self.Repetir()
else:
    if not self.repetido('Arbol Inicio'):
        self.nombres_arboles.append('Arbol Inicio')
```

- Método crear_reporte_registro, crear_reporte_errores_sintactico

Estos métodos son usados para crear los reportes en HTML para ello se accede al método en cuestión. Dentro del mismo se hace la instancia del método para obtener la información de los tokens y se crea una carpeta donde se guardarán los reportes. Luego dentro de un try – except, se crea una variable file, se abre el archivo con el método open, luego con otras auxiliares, se crea la estructura del HTML y se concatena la variable en cuestión. Luego a través del método write se escribe el archivo, al finalizar se cierra el archivo y se hace uso del método implementado por webbrowser open_new_tab para abrir el reporte después de generarlo.

```
def crear_reporte_registro(self, nombre):
    makedirs('Reportes', exist_ok = True)
    self.obtener_datos(True)
    try:
        file = open('Reportes/' + nombre + '.html','w')
        head = '<head><title>Reporte Registro</title></head>\n'
        body = '''<body bgcolor=\"#B6F49D\">
                <table width=\"600\" bgcolor=#B6F49D align=left> <tr> <td><font color=\"black\" FACE=\"Courier\">
                <p align=\"left\">Arnoldo Luis Antonio González Camey  –  Carné: 201701548</p></font>
                </td> </tr></table></br></br>'''
        body += '<h2 align=\"center\"><font color=\"black\" FACE=\"Courier\">'+nombre+'</h2>'
        body += '<table width=\"1000\" bgcolor=#CDF9BA align=center style="border:5px dashed brown">'
        body += self.reporteHTML_registro +'</table></body>'
        html = '<html>\n' + head + body + '</html>'
        file.write(html)
        print('Reporte de Registro generado exitosamente')
    except OSError:
        print("Error al crear el Reporte de Registro")
    finally:
        file.close()
        webbrowser.open_new_tab('Reportes\\' + nombre + '.html')
```

```
def crear_reporte_errores_sintactico(self):
    makedirs('Reportes', exist_ok = True)
    try:
        file = open('Reportes/Reporte_Errores_Sintactico.html','w')
        head = '<head><title>Reporte Errores Sintácticos</title></head>\n'
        body = '''<body bgcolor=\"#B6F49D\">
                <table width=\"600\" bgcolor=#B6F49D align=left> <tr> <td><font color=\"black\" FACE=\"Courier\">
                <p align=\"left\">Arnoldo Luis Antonio González Camey  –  Carné: 201701548</p></font>
                </td> </tr></table></br></br>
                <h2 align=\"center\"><font color=\"black\" FACE=\"Courier\">Tabla Errores Sintáctico</h2>
                <table width=\"1250\" bgcolor=#CDF9BA align=center style="border:5px dashed brown">
                <tr>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Error Lexema</strong></td>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Error</strong></td>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Fila</strong></td>
                    <td align=center><font color=\"#000000\" face=\"Courier\"><strong>Columna</strong></td>
                </tr>'''
        body += self.reporte_error_sintac +'</table></body>'
        html = '<html>\n' + head + body + '</html>'
        file.write(html)
        print('Reporte de errores sintácticos generado exitosamente')
    except OSError:
        print("Error al crear el Reporte de errores sintácticos")
    finally:
        file.close()
        webbrowser.open_new_tab('Reportes\Reporte_Errores_Sintactico.html')
```

- Métodos árbol_...

Estos métodos son usados para obtener la base para generar el árbol de derivación de cada una de las funciones, para ello se crear una variable donde se almacena la información y se retorna la misma.

```
def arbol_inicio(self):
    return self.contenido_inicio
```

```
def arbol_registros(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Registros"];
    \r\t\tn2[label = "Tk_Igual"];
    \r\t\tn3[label = "Tk_Crch_Izq"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;'''
    contenido += self.contenido_registros
    return contenido
```

```
def arbol_claves(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Claves"];
    \r\t\tn2[label = "Tk_Igual"];
    \r\t\tn3[label = "Tk_Crch_Iz"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;'''
    contenido += self.contenido_claves
    return contenido
```

```
def arbol_contarsi(self):
    tipo = ''
    if type(self.valor_contarsi) == float:
        tipo = 'Tk_Decimal'
    elif type(self.valor_contarsi) == int:
        tipo = 'Tk_Numero'
    elif type(self.valor_contarsi) == str:
        tipo = 'Tk_Cadena'
    contenido = '''
    \r\t\tn1[label = "Tk_ContarSi"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Cadena"];
    \r\t\tn4[label = "Tk_Coma"];
    \r\t\tn5[label = "<VALOR_CONTARSI>"];
    \r\t\tn6[label = "'''+tipo+'''"];
    \r\t\tn7[label = "Tk_Paren_Der"];
    \r\t\tn8[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;
    \r\t\traiz -> n5;
    \r\t\tn5 -> n6;
    \r\t\traiz -> n7;
    \r\t\traiz -> n8;'''
    return contenido
```

```
def arbol_comentario_linea(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Comentario_Linea"];
    \r\t\traiz -> n1;'''
    return contenido
```

```
def arbol_comentario_multilinea(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Comentario_Multilinea"];
    \r\t\traiz -> n1;'''
    return contenido
```

```
def arbol_imprimir(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Imprimir"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Cadena"];
    \r\t\tn4[label = "Tk_Paren_Der"];
    \r\t\tn5[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;
    \r\t\traiz -> n5;'''
    return contenido
```

```
def arbol_imprimirln(self):
    contenido = '''
    \r\t\tn1[label = "Tk_ImprimirLn"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Cadena"];
    \r\t\tn4[label = "Tk_Paren_Der"];
    \r\t\tn5[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;
    \r\t\traiz -> n5;'''
    return contenido
```

```
def arbol_conteo(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Conteo"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Paren_Der"];
    \r\t\tn4[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;'''
    return contenido
```

```
def arbol_promedio(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Promedio"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Cadena"];
    \r\t\tn4[label = "Tk_Paren_Der"];
    \r\t\tn5[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;
    \r\t\traiz -> n5;'''
    return contenido
```

```
def arbol_datos(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Datos"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Paren_Der"];
    \r\t\tn4[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;'''
    return contenido
```

```
def arbol_sumar(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Sumar"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Cadena"];
    \r\t\tn4[label = "Tk_Paren_Der"];
    \r\t\tn5[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;
    \r\t\traiz -> n5;'''
    return contenido
```

```
def arbol_max(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Max"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Cadena"];
    \r\t\tn4[label = "Tk_Paren_Der"];
    \r\t\tn5[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;
    \r\t\traiz -> n5;'''
    return contenido
```

```
def arbol_min(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Min"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Cadena"];
    \r\t\tn4[label = "Tk_Paren_Der"];
    \r\t\tn5[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;
    \r\t\traiz -> n5;'''
    return contenido
```

```
def arbol_exportar_reporte(self):
    contenido = '''
    \r\t\tn1[label = "Tk_Exp_Repo"];
    \r\t\tn2[label = "Tk_Paren_Izq"];
    \r\t\tn3[label = "Tk_Cadena"];
    \r\t\tn4[label = "Tk_Paren_Der"];
    \r\t\tn5[label = "Tk_Punto_Coma"];
    \r\t\traiz -> n1;
    \r\t\traiz -> n2;
    \r\t\traiz -> n3;
    \r\t\traiz -> n4;
    \r\t\traiz -> n5;'''
    return contenido
```

- Método generar_graphviz_arbol

Este método se usa para generar el árbol de derivación con Graphviz para ello se crean unas variables en las cuales se guarda la estructura de básica. Luego se crea una carpeta donde se guardará el archivo generado. Se crea el archivo mediante open y por medio de la librería os, se accede al método system en el cual sumado al comando mostrado se convierte un archivo tipo dot en una imagen png y luego mediante el método startfile se inicializa la imagen.

```
def generar_graphviz_arbol(self, nombre_token, contenido, nombre_raiz):
    inicio_graphviz = '''
    \rdigraph L {
    \r\tnode[shape = box fillcolor = "#FFFF00" style = filled]
    \r\tsubgraph cluster_p {
    \r\t\tlabel = "Arbol Derivacion - '''+ nombre_token.replace("_", " ") +''' "
    \r\t\tbgcolor = "#6BD6E9"
    \r\t\t'''+nombre_raiz+'''[label = "<'''+nombre_token.upper()+'''>"]'''
    final_graphviz = '\n\t}\n}'
    graphviz = inicio_graphviz + contenido + final_graphviz
    makedirs('Reportes', exist_ok = True)
    miArchivo = open('Reportes/'+nombre_token+'.dot','w')
    miArchivo.write(graphviz)
    miArchivo.close()
    system('dot -Tpng ' +'Reportes/'+nombre_token+'.dot -o '+'Reportes/'+nombre_token+'.png')
    system('cd ./'+'Reportes/'+nombre_token+'.png')
    startfile('Reportes\\'+nombre_token+'.png')
```

- Clase Clave

Es una clase en la cual se inicializan los atributos de las claves tales como el nombre y el índice.

```
class Clave():
    def __init__(self, indice, nombre):
        self.indice = indice
        self.nombre = nombre
```

- Método get_nombre, get_indice

Este método se usa para retornar el nombre y el índice

```
def get_nombre(self):
    return self.nombre

def get_indice(self):
    return self.indice
```

- Clase Registro

Es una clase en la cual se inicializan los atributos de las claves tales como el índice y el args, el cual pueden ser n cantidad de atributos, a su vez se crea una lista de valores.

```
class Registro():
    valores = []
    def __init__(self, indice, *args):
        self.indice = indice
        self.args = args
```

- Método agregar_registro

Este método se usa agregar un nuevo registro a la lista valores, para ello se crea un nuevo registro con los atributos y se añade a la lista por el método append.

```
def agregar_registro(self, indice, values):
    nuevo = Registro(indice, values)
    self.valores.append(nuevo)
```

- Método reiniciar_registro

Este método se usa para borrar los valores dentro de la lista valores.

```
def reiniciar_registro(self):
    self.valores.clear()
```

- Clase Interfaz

Esta clase es la utilizada para generar la interfaz del programa, para ello se hace uso de la librería tkinter la cual proporciona las herramientas para generar la interfaz, primero se declaran las variables necesarias para iniciar la interfaz y los importes necesarios y se inicializan los métodos en el constructor.

```
from tkinter import *
import tkinter
from tkinter import Frame, ttk, filedialog
from Analizador import Analizador
from Sintactico import Sintactico

WIDTH = 1025
HEIGHT = 550

class Interfaz():

    lexico = Analizador()
    ventana = tkinter.Tk()

    def __init__(self):
        self.configuracion_ventana()
        self.crear_toolbar()
        self.crear_txt()
        self.ventana.mainloop()
```

- Método configuracion_ventana

Este método es utilizado para configurar las dimensiones de la ventana y el título de esta.

```
def configuracion_ventana(self):
    self.ventana.geometry(str(WIDTH)+"x"+str(HEIGHT))
    self.ventana.title('Lenguaje')
```

- Método crear_toolbar

Este método es utilizado para crear una barra de herramientas en la cual se colocan los botones, y el combobox que serán utilizados en la interfaz, en cada botón se agrega la instancia del método que este debe de ejecutar. También se crean las etiquetas y se colocan en la ventana.

```
def crear_toolbar(self):
    toolbar = Frame(self.ventana, bg = 'white')
    boton_abrir = tkinter.Button(toolbar, text = 'Abrir', command = self.leer_archivo, width = 10, height = 2)
    boton_analizar = tkinter.Button(toolbar, text = 'Analizar', command = self.analizar_archivo, width = 10, height = 2)
    boton_reportes = tkinter.Button(toolbar, text = 'Reporte', command = self.crear_reportes,  width = 10, height = 2)
    boton_salir = tkinter.Button(toolbar, text = 'Salir', command = lambda: exit(), width = 10, height = 2)
    label_titulo = tkinter.Label(toolbar, text = "Proyecto 2 - 201701548", font = ('Courier', 11), bg = 'white')
    label_entrada = tkinter.Label(self.ventana, text = "Terminal de Entrada", font = ('Courier', 13))
    label_consola = tkinter.Label(self.ventana, text = "Consola", font = ('Courier', 13))

    label_titulo.pack(side = LEFT, padx = 3, pady = 2)
    boton_salir.pack(side = RIGHT, padx = 3, pady = 2)
    boton_reportes.pack(side = RIGHT, padx = 3, pady = 2)
    self.configuracion_combo(toolbar)
    boton_analizar.pack(side = RIGHT, padx = 3, pady = 2)
    boton_abrir.pack(side = RIGHT, padx = 3, pady = 2)
    toolbar.pack(side = TOP, fill = X)

    label_entrada.place(x = 175, y = 65)
    label_consola.place(x = 725, y = 65)
```

- Método crear_txt

Este método es utilizado para crear y posicionar las cajas de texto, para ello se hace uso de la opción Text de tkinter, a su vez para una visualización más cómoda de la información se crean unas scrollbar para poder subir y bajar dentro de las cajas de texto.

```
def crear_txt(self):
    self.txt_consola = tkinter.Text(self.ventana, height = 27.50, width = 58, font = ('Courier', 9), bg = 'white', state = 'disabled')
    self.txt_entrada = tkinter.Text(self.ventana, height = 26, width = 63, font = ('Courier', 10), bg = 'white')
    self.txt_entrada.place(x = 30, y = 100)
    self.txt_consola.place(x = 565, y = 100)

    scrollbar_entrada = ttk.Scrollbar(self.ventana, orient = "vertical", command = self.txt_entrada.yview)
    scrollbar_entrada.place(x = 538, y = 100, height = 420)
    self.txt_entrada.configure(yscrollcommand = scrollbar_entrada.set)

    scrollbar_consola = ttk.Scrollbar(self.ventana, orient = "vertical", command = self.txt_consola.yview)
    scrollbar_consola.place(x = 975, y = 100, height = 420)
    self.txt_consola.configure(yscrollcommand = scrollbar_consola.set)
```

- Método analizar_archivo

Este método es utilizado primero para reinciar el txt_consola y hacer la instancia del método reiniciar_tokens, y analizar_estados. Luego de hace la instancia de la clase Sintactico y de los demás métodos necesarios.

```
def analizar_archivo(self):
    self.txt_consola.configure(state = 'normal')
    self.txt_consola.delete('1.0', END)
    contenido_Text = self.txt_entrada.get("1.0", tkinter.END)
    self.lexico.reiniciar_tokens()
    self.lexico.analizador_estados(contenido_Text)

    self.sintactico = Sintactico(tkinter, self.lexico.tokens, self.txt_consola)
    self.txt_consola.configure(state = 'disabled')
    self.sintactico.opciones_reporte_arbol(self.combo_reportes)
    self.lexico.obtener_tokens()
    self.sintactico.reniciar()
```

- Método configuracion_combo

Este método se usado para crear un nuevo combobox y se agregan los ítems básicos que tendrá el mismo, luego se indica que el primer item en mostrar sea el de la posición cero y se ubica en el toolbar.

```
def configuracion_combo(self, toolbar):
    self.combo_reportes = ttk.Combobox(toolbar, font = ('Courier', 9), width = 25,  state = "readonly")
    self.combo_reportes["values"] = ["Seleccione el Reporte", "Reporte Tokens", "Reporte Errores Léxico", "Reporte Error Sintáctico"]
    self.combo_reportes.current(0)
    self.combo_reportes.pack(side = RIGHT, padx = 3, pady = 2)
```

- Método crear_reportes

Este método se usa para crear el reporte seleccionado en el combobox, para ello se guarda tanto la posición como el nombre del combobox en unas variables, luego mediante una sentencia if – elif se valida el indice o el nombre para saber en cual opción debe ingresar, al ingresar se hace la instancia del método en cuestión.

```
def crear_reportes(self):
    indice = self.combo_reportes.current()
    nombre = self.combo_reportes.get()
    if indice == 1:
        self.lexico.crear_reporte_token()
    elif indice == 2:
        self.lexico.crear_reporte_errores()
    elif indice == 3:
        self.sintactico.crear_reporte_errores_sintactico()
    elif nombre == 'Arbol Inicio':
        self.sintactico.arboles.generar_graphviz_arbol('Inicio',self.sintactico.arbol_inicio(), 'n0')

    elif nombre == 'Arbol Claves':
        self.sintactico.arboles.generar_graphviz_arbol('Claves',self.sintactico.arbol_claves(), 'raiz')

    elif nombre == 'Arbol Registros':
        self.sintactico.arboles.generar_graphviz_arbol('Registros',self.sintactico.arbol_registros(), 'raiz')

    elif nombre == 'Arbol ContarSi':
        self.sintactico.arboles.generar_graphviz_arbol('ContarSi',self.sintactico.arbol_contarsi(), 'raiz')

    elif nombre == 'Arbol Imprimir':
        self.sintactico.arboles.generar_graphviz_arbol('Imprimir',self.sintactico.arboles.arbol_imprimir(), 'raiz')

    elif nombre == 'Arbol ImprimirLn':
        self.sintactico.arboles.generar_graphviz_arbol('ImprimirLn',self.sintactico.arboles.arbol_imprimirln(), 'raiz')

    elif nombre == 'Arbol Comentario Linea':
        self.sintactico.arboles.generar_graphviz_arbol('Comentario_Linea',self.sintactico.arboles.arbol_comentario_linea(), 'raiz')
```

```
elif nombre == 'Arbol Comentario MultiLinea':
    self.sintactico.arboles.generar_graphviz_arbol('Comentario_MultiLinea',self.sintactico.arboles.arbol_comentario_multilinea(), 'raiz')

elif nombre == 'Arbol Conteo':
    self.sintactico.arboles.generar_graphviz_arbol('Conteo',self.sintactico.arboles.arbol_conteo(), 'raiz')

elif nombre == 'Arbol Promedio':
    self.sintactico.arboles.generar_graphviz_arbol('Promedio',self.sintactico.arboles.arbol_promedio(), 'raiz')

elif nombre == 'Arbol Datos':
    self.sintactico.arboles.generar_graphviz_arbol('Datos',self.sintactico.arboles.arbol_datos(), 'raiz')

elif nombre == 'Arbol Sumar':
    self.sintactico.arboles.generar_graphviz_arbol('Sumar',self.sintactico.arboles.arbol_sumar(), 'raiz')

elif nombre == 'Arbol Max':
    self.sintactico.arboles.generar_graphviz_arbol('Max',self.sintactico.arboles.arbol_max(), 'raiz')

elif nombre == 'Arbol Min':
    self.sintactico.arboles.generar_graphviz_arbol('Min',self.sintactico.arboles.arbol_min(), 'raiz')

elif nombre == 'Arbol Exportar Reporte':
    self.sintactico.arboles.generar_graphviz_arbol('Exportar_Reporte',self.sintactico.arboles.arbol_exportar_reporte(), 'raiz')
```

- Método leer_archivo

Este método es utilizado para leer el archivo para realizar la carga de las imagenes. Para ello dentro de un try – except se crea una variable fichero en la cual por medio del método askopenfilename de la librería Tkinter se guarda el path del archivo, luego a través del método open se guarda el contenido del archivo en una variable data para luego ser utilzada.

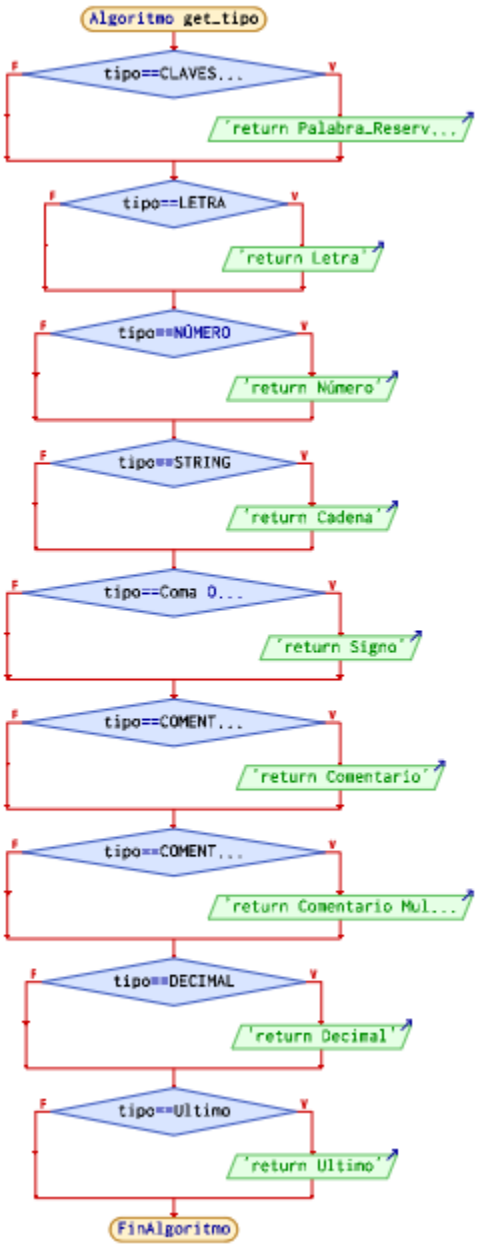
```
def leer_archivo(self):
    try:
        ruta = filedialog.askopenfilename(title = "Abrir un archivo")
        with open(ruta, 'rt', encoding = 'utf-8') as f:
            print('Archivo cargado con éxito')
            self.data = f.read()
            self.txt_entrada.insert(tkinter.INSERT, self.data)
    except OSError:
        print("<<< No se pudo leer el Archivo", ruta ,'>>>')
        return
```

Esta validación sirve para que Python sepa por cuál método debe empezar a ejecutar el programa

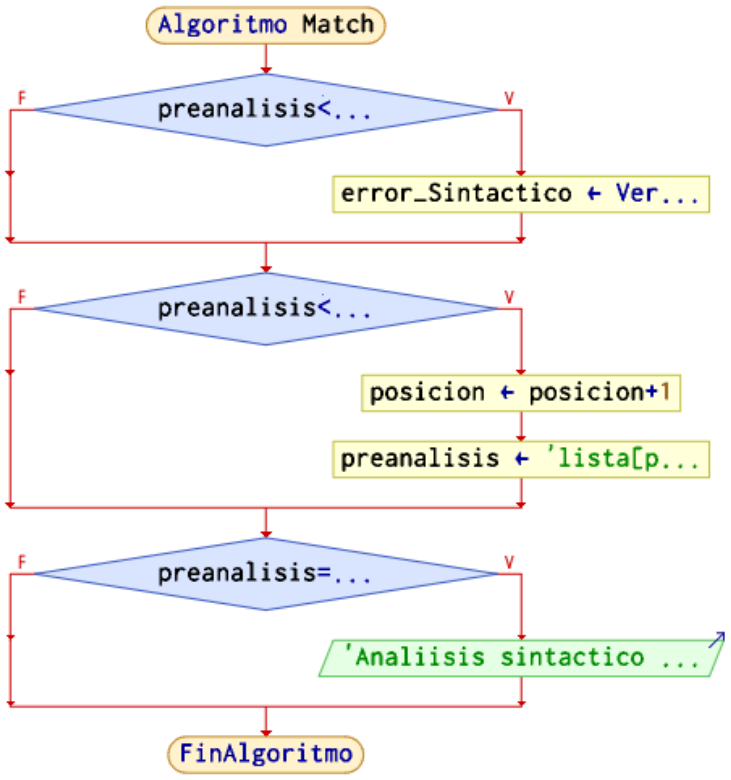
```
if __name__ == '__main__':
    Interfaz()
```

Diagramas de Flujo

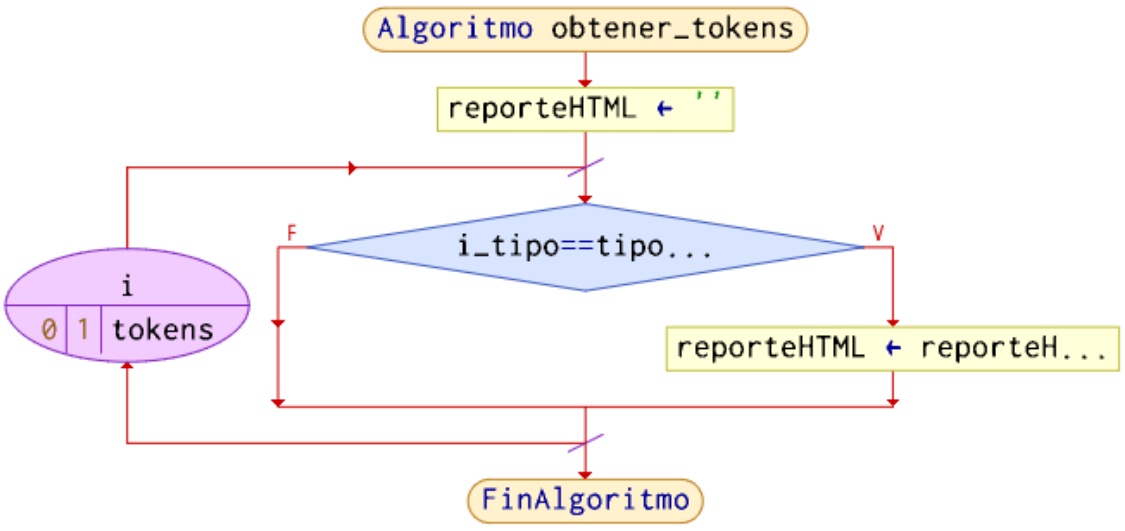
- Método get_tipo



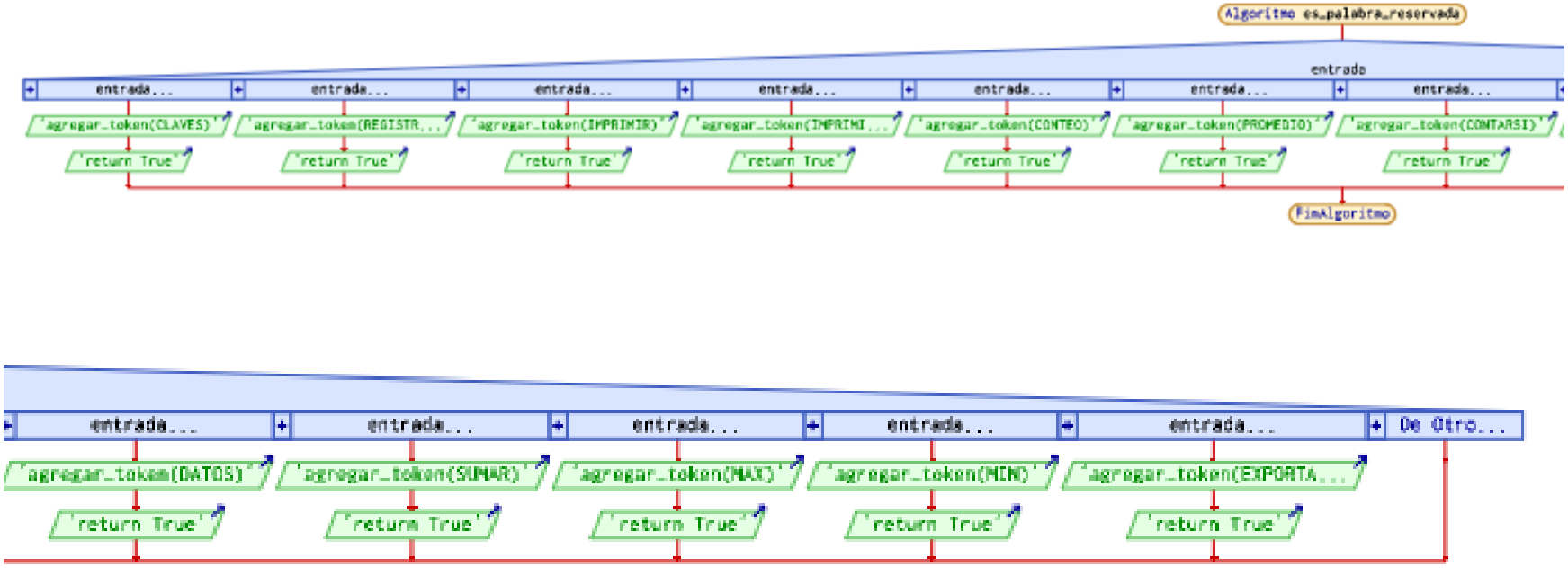
- Método Match



- Método obtener_tokens



- Método es_palabra_reservada



Expresión Regular

PALABRA RESERVADA = { Claves, Registros, imprimir, imprimirln, conteo, promedio, contarsi, datos, sumar, max, min, exportarReporte

LETRA = L = {A − Z} = L^+ → L^+#1

NÚMERO = N = {0 − 9} = N^+ → N#2

RESTO = R = {cualquier símbolo}

CADENAS = "(L|N|R)*" → "(L|N|R)*" #3

SIGNOS = S = {igual, corchete izquierdo, corchete derecho, coma, punto y coma, parentesis izquierdo, parentesis derecho, llave izquierda, llave derecha} → S^+#4

COMENTARIOS_MULTILINEA = CM = '''(L|N|R)*''' → '''(L|N|R)*''' #5

COMENTARIOS_NORMAL = CN = #(L|N|R)* → #(L|N|R)* #6

PUNTO = PT = .

DECIMAL = D = N^+PTN^+ → N^+PTN^+#7

EXPRESIÓN REGULAR = [L^+ | N^+ | "(L|N|R)*" | S^+ | '''(L|N|R)*''' | #(L|N|R)* | N^+PTN^+]

Token	Expresión Regular
LETRA L	L^+
NUMERO N	N^+
CADENAS	"(L|D|R)*"
SIGNOS S	S^+
COMENTARIOS_MULTILINEA	'''(L|N|R)*'''
COMENTARIOS_NORMAL	#(L|N|R)*
DECIMAL D	N^+PTN^+
	[L^+ | N^+ | "(L|N|R)*" | S^+ | '''(L|N|R)*''' | #(L|N|R)* | N^+PTN^+]

Método del Árbol

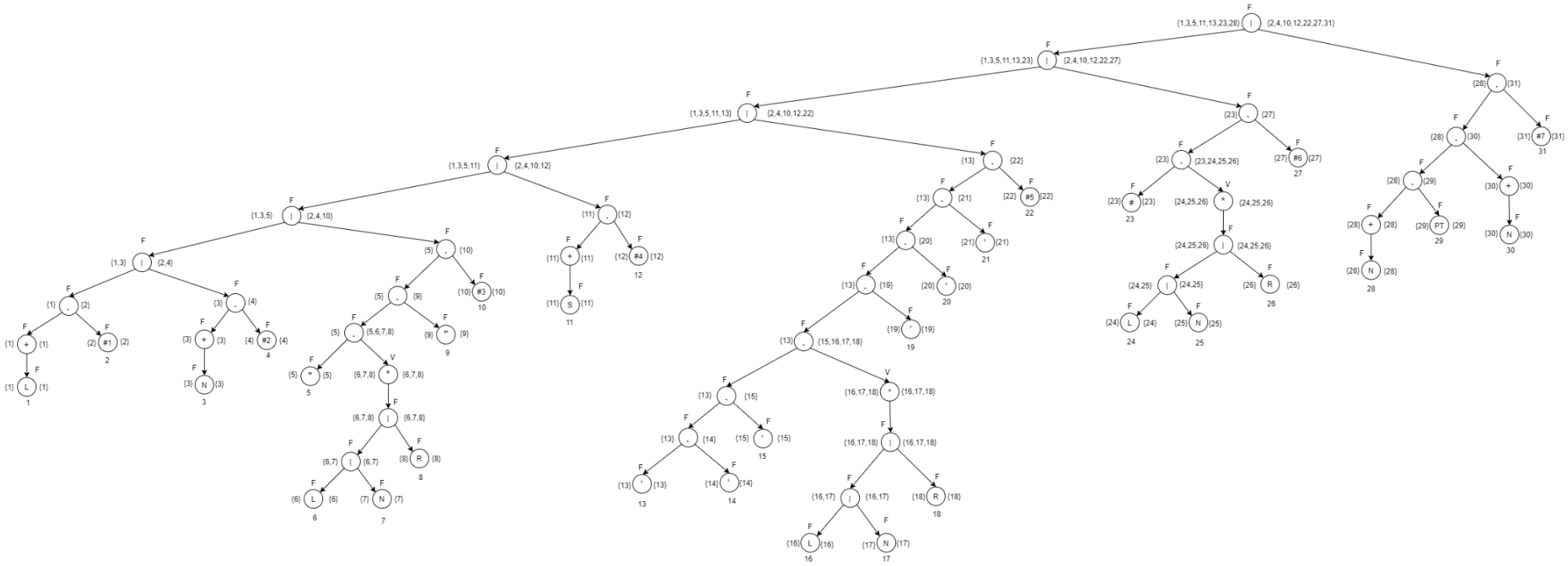


Tabla de Siguientes

i	Terminal	Sig(i)
1	L	1,2
2	#1	
3	N	3,4
4	#2	
5	"	6,7,8,9
6	L	6,7,8,9
7	N	6,7,8,9
8	R	6,7,8,9
9	"	10
10	#3	
11	S	11,12
12	#4	
13	'	14
14	'	15
15	'	16,17,18,19
16	L	16,17,18,19
17	N	16,17,18,19
18	R	16,17,18,19
19	'	20
20	'	21
21	'	22
22	#5	
23	#	24,25,26,27
24	L	24,25,26,27
25	N	24,25,26,27
26	R	24,25,26,27
27	#6	
28	N	28,29
29	PT	30
30	N	30,31
31	#7	

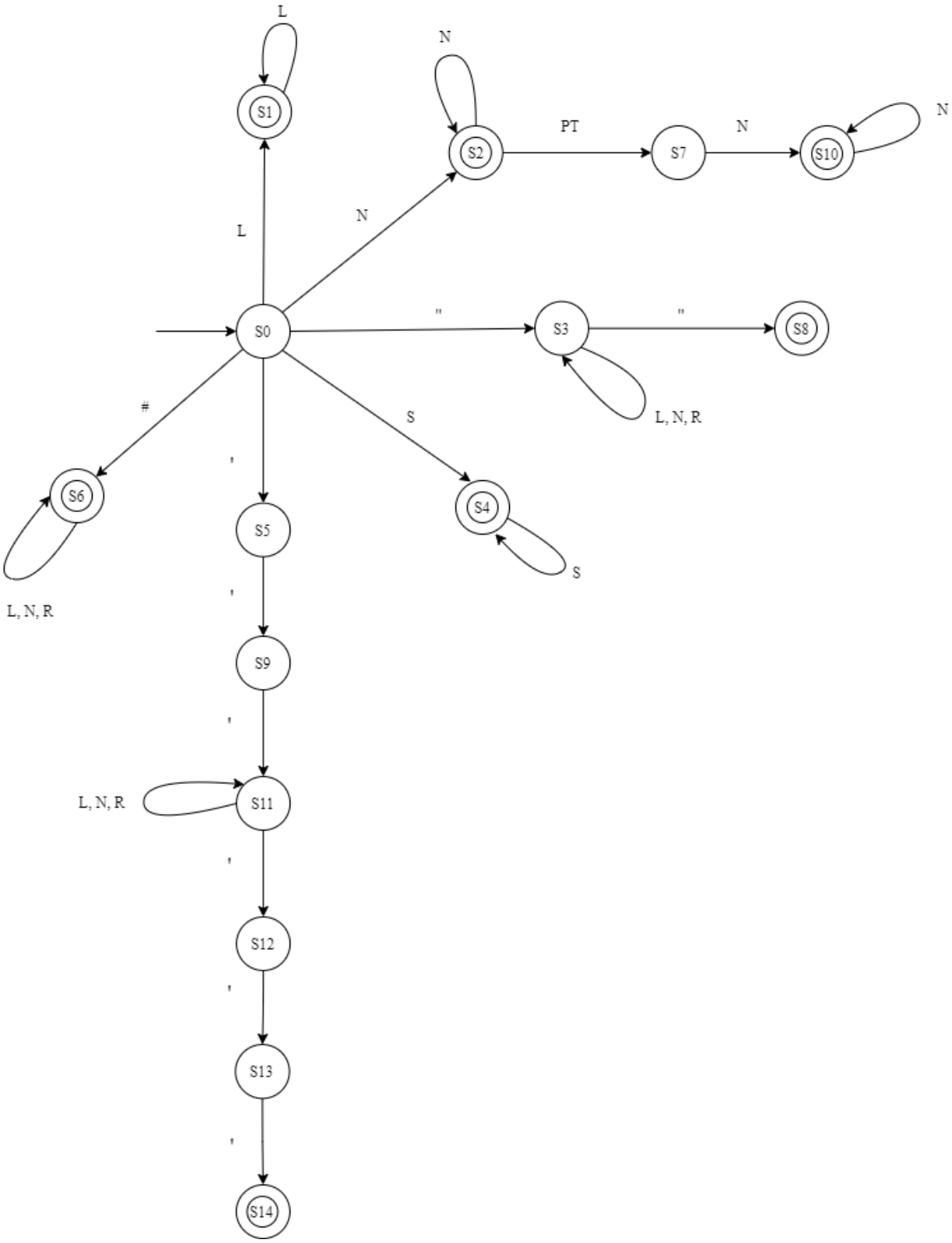
Tabla de Estados

S0 = {1, 3, 5, 11, 13, 23,28}
L N " S ' # N
Sig(L) = Sig(1) = {1, 2} = S1
Sig(N) = Sig(3)USig(28) = {3, 4, 28, 29} = S2
Sig(") = Sig(5) = {6, 7, 8, 9} = S3
Sig(S) = Sig(11) = {11, 12} = S4
Sig(') = Sig(13) = {14} = S5
Sig(#) = Sig(23) = {24, 25, 26, 27} = S6
S1 = {1, 2}
L #1
Sig(L) = Sig(1) = {1, 2} = S1
S2 = {3, 4, 28, 29}
N #2 N PT
Sig(N) = Sig(3)USig(28) = {3, 4, 28, 29} = S2
Sig(PT) = Sig(29) = {30} = S7
S3 = {6, 7, 8, 9}
L N R "
Sig(L) = Sig(6) = {6, 7, 8, 9} = S3
Sig(N) = Sig(7) = {6, 7, 8, 9} = S3
Sig(R) = Sig(8) = {6, 7, 8, 9} = S3
Sig(") = Sig(9) = {10} = S8
S4 = {11, 12}
S #4
Sig(S) = Sig(11) = {11, 12] = S4
S5 = {14}
'
Sig(') = Sig(14) = {15} = S9

S6 = {24, 25, 26, 27}
L N R #6
Sig(L) = Sig(24) = {24, 25, 26, 27} = S6
Sig(N) = Sig(25) = {24, 25, 26, 27} = S6
Sig(R) = Sig(26) = {24, 25, 26, 27} = S6
S7 = {30}
N
Sig(N) = Sig(30) = {30, 31} = S10
S8 = {10} #3
S9 = {15}
'
Sig(') = Sig(15) = {16, 17, 18, 19} = S11
S10 = {30, 31}
N #7
Sig(N) = Sig(30) = {30, 31} = S10
S11 = {16, 17, 18, 19}
L N R '
Sig(L) = Sig(16) = {16, 17, 18, 19} = S11
Sig(N) = Sig(17) = {16, 17, 18, 19} = S11
Sig(R) = Sig(18) = {16, 17, 18, 19} = S11
Sig(') = Sig(19) = {20} = S12
S12 = {20}
'
Sig(') = Sig(20) = {21} = S13
S13 = {21}
'
Sig(') = Sig(21) = {22} = S14
S14 = {22} #5

Estado	L	#1	N	#2	R	"	#3	S	#4	'	#5	#	#6	PT	#7
S0	S1		S2			S3		S4		S5		S6			
*S1	S1														
*S2			S2											S7	
S3	S3		S3		S3	S8									
*S4								S4							
S5										S9					
*S6	S6		S6		S6										
S7			S10												
*S8															
S9										S11					
*S10			S10												
S11	S11		S11		S11					S12					
S12										S13					
S13										S14					
*S14															

Autómata Finito Determinista



Gramática

- Inicio

<INICIO>::= <CLAVES> <REPETIR>
| <REGISTROS> <REPETIR>
| <COMENTARIO_LINEA> <REPETIR>
| <COMENTARIO_MULTILINEA> <REPETIR>
| <IMPRIMIR> <REPETIR>
| <IMPRIMIRLN> <REPETIR>
| <CONTEO> <REPETIR>
| <PROMEDIO> <REPETIR>
| <CONTARSI> <REPETIR>
| <DATOS> <REPETIR>
| <SUMAR> <REPETIR>
| <MAX> <REPETIR>
| <MIN> <REPETIR>
| <EXPORTAR_REPORTE> <REPETIR>

<REPETIR>::= <CLAVES> <REPETIR>
| <REGISTROS> <REPETIR>
| <COMENTARIO_LINEA> <REPETIR>
| <COMENTARIO_MULTILINEA> <REPETIR>
| <IMPRIMIR> <REPETIR>
| <IMPRIMIRLN> <REPETIR>
| <CONTEO> <REPETIR>
| <PROMEDIO> <REPETIR>
| <CONTARSI> <REPETIR>
| <DATOS> <REPETIR>
| <SUMAR> <REPETIR>
| <MAX> <REPETIR>
| <MIN> <REPETIR>
| <EXPORTAR_REPORTE> <REPETIR>
| Epsilon

- Claves

<CLAVES>::= TK_Clave Tk_Igual Tk_Corchete_Izq <BLOQUE_CLAVES> Tk_Corchete_Der

<BLOQUE_CLAVES>::= Tk_Cadena Tk_Coma <BLOQUE_CLAVES>

| Tk_Cadena

- Registros

<REGISTROS>::= Tk_Registros Tk_Igual Tk_Corchete_Izq <BLOQUE_REGISTROS> Tk_Corchete_Der

<BLOQUE_REGISTROS>::= <CUERPO_REGISTROS> <BLOQUE_REGISTROS>

| <CUERPO_REGISTROS>

<CUERPO_REGISTROS>::= Tk_Llave_Izq <VALOR_REGISTRO> Tk_Llave_Der

<VALOR_REGISTRO>::= Tk_Numero

| Tk_Numero Tk_Coma <VALOR_REGISTRO>
| Tk_Cadena
| Tk_Cadena Tk_Coma <VALOR_REGISTRO>
| Tk_Decimal
| Tk_Decimal Tk_Coma <VALOR_REGISTRO>

- Contarsi

<CONTARSI>::= Tk_ContarSi Tk_Paren_Izq Tk_Cadena Tk_Coma <VALOR_CONTARSI> Tk_Paren_Der Tk_Punto_Coma

<VALOR_CONTARSI>::= Tk_Numero

| Tk_Decimal

| Tk_Cadena

- Comentario Línea

<COMENTARIO_LINEA>::= Tk_Comentario_Linea

- Comentario Multilínea

<COMENTARIO_MULTILINEA>::= Tk_Comentario_MultiLinea

- Imprimir

<IMPRIMIR>::= Tk_Imprimir Tk_Paren_Izq Tk_Cadena Tk_Paren_Der Tk_Punto_Coma

- ImprimirIn

<IMPRIMIRLN>::= Tk_ImprimirLn Tk_Paren_Izq Tk_Cadena Tk_Paren_Der Tk_Punto_Coma

- Conteo

<CONTEO>::= Tk_Conteo Tk_Paren_Izq Tk_Paren_Der Tk_Punto_Coma

- Promedio

<PROMEDIO>::= Tk_Promedio Tk_Paren_Izq Tk_Cadena Tk_Paren_Der Tk_Punto_Coma

- Datos

<DATOS>::= Tk_Datos Tk_Paren_Izq Tk_Paren_Der Tk_Punto_Coma

- Sumar

<SUMAR>::= Tk_Sumar Tk_Paren_Izq Tk_Cadena Tk_Paren_Der Tk_Punto_Coma

- Max

<MAX>::= Tk_Max Tk_Paren_Izq Tk_Cadena Tk_Paren_Der Tk_Punto_Coma

- Min

<MIN>::= Tk_Min Tk_Paren_Izq Tk_Cadena Tk_Paren_Der Tk_Punto_Coma

- Exportar Reporte

<EXPORTAR_REPORTE>::= Tk_Exp_Repo Tk_Paren_Izq Tk_Cadena Tk_Paren_Der Tk_Punto_Coma