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 y escritura de archivos y la creación de imágenes jpg, así como la utilización de token para la lectura de los archivos y la manipulación de estos en el lenguaje de programación Python, a su vez 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

    PALABRA_RESERVADA = 1
    LETRA = 2
    NUMERO = 3
    RESTO = 4
    CADENA = 5
    SIGNO = 6
    COLOR = 7
    BOOL = 8
    ERROR = 9

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

- Clase Color

Es una clase en la cual se inicializan los atributos de los colores tales como la posición en "x" y en "y", pintar, el código y el índice de la imagen.

```
class Color():
    def __init__(self, x, y, pintar, codigo, indice_imagen):
        self.x = x
        self.y = y
        self.pintar = pintar
        self.codigo = codigo
        self.indice_imagen = indice_imagen
```

- Clase Imagen

Es una clase en la cual se inicializan los atributos de las imágenes, así como el título, filas, columnas, ancho, alto, los filtros, la cantidad de colores y el índice de la imagen; a su vez se inicializa un arreglo de colores.

```
from Color import Color
class Imagen():
    colores = []
    def __init__(self, titulo, filas, columnas, ancho, alto, mirrorx, mirrory, doublemirror, cantidad_colores, indice_imagen):
        self.titulo = titulo
        self.filas = filas
        self.columnas = columnas
        self.ancho = ancho
        self.alto = alto
        self.mirrorx = mirrorx
        self.mirrory = mirrory
        self.doublemirror = doublemirror
        self.cantidad_colores = cantidad_colores
        self.indice_imagen = indice_imagen
```

- 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

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.PALABRA_RESERVADA:
        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.SIGNO:
        return 'Signo'
    elif self.tipo == self.COLOR:
        return 'Color'
    elif self.tipo == self.BOOL:
        return 'Booleano'
```

- Método get_lexema, get_fila, get_columna, get_id

Este método es utilizado para retornar el lexema, la fila, la columna o el id 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

def get_id(self):
    return self.id
```

- 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. Al finalizar se evalua si el tipo no es igual a nueve si no lo es agrega uno más al id.

```
def agregar_token(self, tipo):
    nuevo_token = Token(self.lexema, tipo, self.fila, self.columna, self.id)
    self.tokens.append(nuevo_token)
    self.lexema = ''
    self.estado = 0
    if tipo != 9:
        self.id += 1
```

- 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, html2image y el importe de las clases Token e Imagen. Luego se declaran las variables necesarias y la instancia de las clases.

```
from Token import Token
from Imagen import Imagen
import re
import webbrowser
from html2image import Html2Image
from PIL import Image, ImageDraw
from os import startfile

class Analizador():
    indice_nombre_imagen = 1
    reporteHTML_token = ''
    reporteHTML_errores = ''
    reporte_imagen = ''
    imagenes = []
    lexema = ''
    estado = 0
    tokens = []
    columna = 1
    fila = 1
    id = 0
    img = Image.new('RGB', (550, 550), color='white')
    draw = ImageDraw.Draw(img)
    tipos = Token("lexema", -1, -1, -1, -1)
    colores = Imagen('', -1, -1, -1, -1, False, False, False, -1, -1)
```

- 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 de los métodos quitar_espacios y 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 es un signo igual, si lo es ingresa cambia el estado a 4, modifica los valores y hace la instancia del método agregar_token, como siguiente evalúa 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 evalua si es una arroba, si lo es cambia el estado a 4, suma una columna y concatena el lexema, se realiza lo mismo con el numeral, luego se evalua 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.

En el estado uno solo se permiten letras, si en dado caso el actual carácter no es una letra verifica en los métodos que tipo de token es y lo guarda. En el estado dos solo son permitidos los dígitos, si en dado caso el carácter actual no es un digito guarda el token. Realiza un proceso similar en todos los otros estados.

```
def analizador_estados(self, entrada):
    self.estado = 0
    self.lexema = ''
    self.tokens = []
    self.fila = 1
    self.columna = 1
    entrada = self.quitar_espacios(entrada)
    entrada = self.separar(entrada)
    entrada += '¬'
    print(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.estado = 4
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.SIGNO)

elif actual == '[' or actual == ']' or actual == '{' or actual == '}' or actual == ',' or actual == ';':
    self.estado = 4
    self.columna += 1
    self.lexema += actual
    self.agregar_token(self.tipos.SIGNO)

elif actual == '@':
    self.estado = 4
    self.columna += 1
    self.lexema += actual

elif actual == '#':
    self.estado = 5
    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:
        print('Análisis terminado')

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

elif self.estado == 1:
    if actual.isalpha():
        self.estado = 1
        self.columna += 1
        self.lexema += actual
    else:
        if self.es_palabra_reserva(self.lexema):
            self.agregar_token(self.tipos.PALABRA_RESERVADA)
        elif self.es_booleano(self.lexema):
            self.agregar_token(self.tipos.BOOL)
        else:
            self.agregar_token(self.tipos.ERROR)
```

```
elif self.estado == 2:
    if actual.isdigit():
        self.estado = 2
        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 = 6
        self.lexema += actual
        self.agregar_token(self.tipos.CADENA)

elif self.estado == 4:
    if actual == '@':
        self.estado = 4
        self.columna += 1
        self.lexema += actual
    else:
        self.agregar_token(self.tipos.SIGNO)
```

```
elif self.estado == 5:
    if actual.isalpha():
        self.estado = 5
        self.columna += 1
        self.lexema += actual

    elif actual.isdigit():
        self.estado = 5
        self.columna += 1
        self.lexema += actual

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

- 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 quitar_espacios

Este método es utilizado para eliminar los espacios entre los signos y los caracteres, para ello se hace uso de la librería re, en ella se declaran lo que se desea reemplazar en este caso un espacio y se retorna la nueva entrada.

```
def quitar_espacios(self, entrada):
    patron = ' +'
    return re.sub(patron, '', 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()
    palabras_reservadas = ['TITULO', 'ANCHO', 'ALTO', 'FILAS', 'COLUMNAS', 'CELDAS', 'FILTROS', 'MIRRORX', 'MIRRORY', 'DOUBLEMIRROR']
    if entrada in palabras_reservadas:
        return True
    return False
```

- Método es_booleano

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

```
def es_booleano(self, entrada):
    entrada = entrada.upper()
    booleanos = ['TRUE', 'FALSE']
    if entrada in booleanos:
        return True
    return False
```

- Método guardar_imagen

Este método se usa para guardar los tokens en la clase Imagen y Color para ello se declaran las variables necesarias. Luego mediante un ciclo for que recorre la totalidad de la lista de tokens, se evalua si el lexema del token actual es igual a la cadena correspondiente si es así entra y guarda el id del token en una variable auxiliar, luego evalua si el tipo token coincide con el actual y el id menos dos es igual al id auxiliar si es así ingresa y guarda en la variable correspondiente el valor y reinicia la variable auxiliar, este proceso lo realiza con cada una de las palabras reservadas. Seguido evalua si acabo con una línea de colores mediante el id. Evalúa si es lexema actual es Celdas y la posición del id cumple, si es así ingresa y guarda las coordenadas realiza lo mismo para los de tipo booleano. Luego evalúa los filtros si es así guarda en una variable el auxiliar que si se requiere el filtro en cuestión. Para finalizar verifica si el lexema actual es igual a cuatro arrobas o el contador es igual al tamaño de la lista menos 1 si es así ingresa y crea la imagen con los datos recopilados con anterioridad y reinicia las variables utilizadas.

```
def guardar_imagen(self):
    titulo = ''
    filas = 0
    columnas = 0
    ancho = 0
    alto = 0
    cantidad_colores = 0
    mirrorx = False
    mirrory = False
    doublemirror = False
    valor = False
    indice_imagen = 0
    tamano = len(self.tokens)
    counter = 0
    id_token = -1
    id_coordenada = -1
    lexema = ''
    entro_primera_coordenada = False
    for token in self.tokens:
        if token.get_lexema() == 'TITULO':
            id_token = token.get_id()

        elif token.tipo == self.tipos.CADENA  and id_token == int(token.get_id()) - 2:
            titulo = token.get_lexema()
            titulo = titulo.replace('"','')
            id_token = -1

        elif token.get_lexema() == 'ANCHO':
            lexema = token.get_lexema()
            id_token = token.get_id()
```

```
elif token.tipo == self.tipos.NUMERO and id_token == int(token.get_id()) - 2 and lexema == 'ANCHO':
    ancho = token.get_lexema()
    id_token = -1

elif token.get_lexema() == 'ALTO':
    lexema = token.get_lexema()
    id_token = token.get_id()

elif token.tipo == self.tipos.NUMERO and id_token == int(token.get_id()) - 2 and lexema == 'ALTO':
    alto = token.get_lexema()
    id_token = -1

elif token.get_lexema() == 'FILAS':
    lexema = token.get_lexema()
    id_token = token.get_id()

elif token.tipo == self.tipos.NUMERO and id_token == int(token.get_id()) - 2 and lexema == 'FILAS':
    filas = token.get_lexema()
    id_token = -1

elif token.get_lexema() == 'COLUMNAS':
    lexema = token.get_lexema()
    id_token = token.get_id()

elif token.tipo == self.tipos.NUMERO and id_token == int(token.get_id()) - 2 and lexema == 'COLUMNAS':
    columnas = token.get_lexema()
    id_token = -1

elif token.get_lexema() == 'CELDAS':
    lexema = token.get_lexema()
    id_token = token.get_id()
```

```
elif token.get_lexema() == 'FILTROS':
    lexema = token.get_lexema()
    id_token = token.get_id()

if id_coordenada == int(token.get_id()) - 9: #saber si ya toca otra linea de colores
    coordenada_x = -1
    coordenada_y = -1

if token.tipo == self.tipos.NUMERO and lexema == 'CELDAS' :
    #LAS PRIMERAS COORDENADAS
    if coordenada_x == -1 and id_token == int(token.get_id()) - 4 and entro_primera_coordenada == False:
        coordenada_x = token.get_lexema()
        id_coordenada = token.get_id()

    elif coordenada_y == -1 and id_token == int(token.get_id()) - 6 and entro_primera_coordenada == False:
        coordenada_y = token.get_lexema()
        entro_primera_coordenada = True

    #SIGUIENTES CELDAS
    if coordenada_x == -1 and id_coordenada == int(token.get_id()) - 10:
        coordenada_x = token.get_lexema()
        id_coordenada = token.get_id()

    elif coordenada_y == -1 and id_coordenada == int(token.get_id()) - 2:
        coordenada_y = token.get_lexema()

elif token.tipo == self.tipos.BOOL and lexema == 'CELDAS' and id_coordenada == int(token.get_id()) - 4:
    if token.get_lexema() == 'TRUE':
        valor = True
    elif token.get_lexema() == 'FALSE':
        valor = False
```

```
elif token.tipo == self.tipos.COLOR and lexema == 'CELDAS' and id_coordenada == int(token.get_id()) - 6:
    codigo_color = token.get_lexema()
    self.colores.agregar_color(coordenada_x, coordenada_y, valor, codigo_color, indice_imagen)
    cantidad_colores += 1
    #print(coordenada_x, coordenada_y,valor,codigo_color)

if token.get_lexema() == 'MIRRORX':
    mirrorx = True
elif token.get_lexema() == 'MIRRORY':
    mirrory = True
elif token.get_lexema() == 'DOUBLEMIRROR':
    doublemirror = True

if token.get_lexema() == '@@@@' or counter == tamano - 1 :
    nueva_imagen = Imagen(titulo, filas, columnas, ancho, alto, mirrorx, mirrory, doublemirror ,cantidad_colores, indice_imagen)
    self.imagenes.append(nueva_imagen)
    indice_imagen += 1
    #REINICIO DE LAS VARIABLES
    id_token = -1
    id_coordenada = -1
    lexema = ''
    entro_primera_coordenada = False
    valor = False
    cantidad_colores = 0
    #print(titulo, ancho, alto, filas, columnas)

counter += 1
```

- Método graficar_imagen

Este método se usa para graficar la imagen en la interfaz, para ello mediante un ciclo for que recorre la totalidad de imágenes se inicia un ciclo while el cual continua a través de la cantidad de colores que existan, se evalúa si la cantidad de colores es igual al contador y se rompe el ciclo, luego se evalúa si el titulo de la imagen es igual al nombre del parámetro. Luego evalúa si los parámetros mirror_x, mirror_y o double_mirror son verdades si lo son verifica si los atributos mirror_x, mirror_y o double_mirror son falsos, si esto se cumple se muestran en pantalla un mensaje y se retorna. Seguido de esto se guardan los valores en unas variables auxiliares y se verifica si el índice de la imagen es igual al índice del color, si esto se cumple guarda realiza ciertas validaciones y operaciones para conocer las coordenadas donde se debe graficar. Al finalizar se evalúa si el atributo pintar es verdadero si lo es pinta en la interfaz en las coordenadas correspondientes.

```
def graficar_imagen(self, lienzo, ancho, alto, mirror_x, mirror_y, double_mirror, nombre, tkinter):
    contador = 0
    self.reporteHTML_imagen = ''
    for image in self.imagenes:
        while contador <= len(image.colores) - 1:
            if image.cantidad_colores == contador:
                break
            if image.titulo == nombre:
                if mirror_x:
                    if image.mirrorx is False:
                        tkinter.messagebox.showinfo(message = "El Filtro MIRRORX no se puede mostrar", title = "Alerta")
                        return
                if mirror_y:
                    if image.mirrory is False:
                        tkinter.messagebox.showinfo(message = "El Filtro MIRRORY no se puede mostrar", title = "Alerta")
                        return
                if double_mirror:
                    if image.doublemirror is False:
                        tkinter.messagebox.showinfo(message = "El Filtro DOUBLEMIRROR no se puede mostrar", title = "Alerta")
                        return
                filas = int(image.filas)
                columnas = int(image.columnas)
                factor_x = ancho//filas
                factor_y = alto//columnas
```

```
    if image.indice_imagen == image.colores[contador].indice_imagen:
        aux_x = int(image.colores[contador].x)
        aux_y = int(image.colores[contador].y)
        if mirror_x:
            aux_x = int(filas) - int(aux_x) + 1
        if mirror_y:
            aux_y = int(columnas) - int(aux_y) + 1
        if double_mirror:
            aux_x = int(filas) - int(aux_x) + 1
            aux_y = int(columnas) - int(aux_y) + 1
        coordenada_x = aux_x * factor_x
        coordenada_y = aux_y * factor_y

        y_arriba = coordenada_y + factor_y
        x_abajo = coordenada_x + factor_x
        if image.colores[contador].pintar:
            self.reporteHTML_imagen += 'contenido.fillStyle = \"'+image.colores[contador].codigo+'\";'
            self.reporteHTML_imagen += 'contenido.fillRect('+str(coordenada_x)+','+str(coordenada_y)+','+str(factor_x)+','+str(factor_y)+');\n'
            #print(coordenada_x, y_arriba, x_abajo, coordenada_y, image.colores[contador].codigo)
            lienzo.create_rectangle(coordenada_x, y_arriba, x_abajo, coordenada_y, width = 0, fill = image.colores[contador].codigo)
contador += 1
```

- Método opciones_imagenes

Este método se usa para conocer los títulos de las imágenes para poder colocarlas en el combobox de la interfaz para ellos se declara una lista de nombres y recorriendo la lista de imágenes se guardan en la lista de nombres los títulos, para al final concatenar ambas listas.

```
def opciones_imagenes(self, combo):
    nombres = []
    values = list(combo["values"])
    for img in self.imagenes:
        nombres.append(img.titulo)
    combo["values"] = values + nombres
```

- 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>'+ font + x.get_lexema() +
```

- 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(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>'+ font + str(x.get_fila())
```

- Método crear_reporte_token, crear_reporte_errores

Estos métodos son usados para crear los reportes en HTML para ello se accede a la variable en cuestión. 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):
    try:
        file = open('Reporte_Tokens.html','w')
        head = '<head><title>Reporte Token</title></head>\n'
        body = "<body bgcolor=\"#B6F49D\">"
        body += "<table width=\"600\" bgcolor=#B6F49D align=left> <tr> <td><font color=\"black\" FACE=\"Courier\">"
        body += "<p align=\"left\">Arnoldo Luis Antonio González Camey  –  Carné: 201701548</p></font>"
        body += "</td> </tr></table></br></br>"
        body += ''' <h2 align=\"center\"><font color=\"black\" FACE=\"Courier\">Reporte 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('Reporte_Tokens.html')
```

```
def crear_reporte_errores(self):
    try:
        file = open('Reporte_Errores.html','w')
        head = '<head><title>Reporte Errores</title></head>\n'
        body = "<body bgcolor=\"#B6F49D\">"
        body += "<table width=\"600\" bgcolor=#B6F49D align=left> <tr> <td><font color=\"black\" FACE=\"Courier\">"
        body += "<p align=\"left\">Arnoldo Luis Antonio González Camey  –  Carné: 201701548</p></font>"
        body += "</td> </tr></table></br></br>"
        body += ''' <h2 align=\"center\"><font color=\"black\" FACE=\"Courier\">Reporte de Errores</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 generado exitosamente')
    except OSError:
        print("Error al crear el Reporte de Errores")
    finally:
        file.close()
        webbrowser.open_new_tab('Reporte_Errores.html')
```

- Método guardar_imagen_png

Este método se usa para crear la imagen en un archivo png para ello se hace uso del método save de la librería Pillow, se manda como parámetro el nombre de la imagen y luego mediante el método startfile de la librería os se abre la imagen.

```
def guardar_imagen_png(self, nombre):
    self.img.save(nombre+str(self.indice_nombre_imagen)+'.png')
    startfile(nombre+str(self.indice_nombre_imagen)+'.png')
    self.indice_nombre_imagen +=1
```

- 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.

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

WIDTH = 800
HEIGHT = 700

class Interfaz():
    data = ''
    lexico = Analizador()
    ventana = tkinter.Tk()
    ancho = 550
    alto = 550

    def __init__(self):
        self.configuracion_ventana()
        lienzo = Canvas(self.ventana, width = self.ancho, height = self.alto, bg = 'white')
        lienzo.place(x = 150, y = 100)
        combo_imagenes = ttk.Combobox(self.ventana, font = ('Courier', 11), state = "readonly")
        self.crear_botones(lienzo, combo_imagenes)
        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('Bitxelart')
```

- Método crear_botones

Este método es utilizado para crear los botones que serán utilizados en la interfaz y colocarlos en la posición deseada dentro de la ventana, en cada botón se agrega la instancia del método que este debe de ejecutar. Al final del método de crear una label del nombre de la imagen y se posiciona.

```
def crear_botones(self, lienzo, combo_imagenes):
    boton_cargar = tkinter.Button(self.ventana, text = 'Cargar', command = self.leer_archivo, width = 10, height = 2)
    boton_analizar = tkinter.Button(self.ventana, text = 'Analizar', command = lambda: self.analizar_archivo(combo_imagenes), width = 10
    boton_reportes = tkinter.Button(self.ventana, text = 'Reportes', command = self.crear_reportes,  width = 10, height = 2)
    boton_salir = tkinter.Button(self.ventana, text = 'Salir', command = lambda: exit(), width = 10, height = 2)

    boton_original = tkinter.Button(self.ventana, text = 'Original', command = lambda: self.dibujar_imagen(lienzo, combo_imagenes, False
    boton_mirror_x = tkinter.Button(self.ventana, text = 'Mirror X', command = lambda: self.dibujar_imagen(lienzo, combo_imagenes, True,
    boton_mirror_y = tkinter.Button(self.ventana, text = 'Mirror Y', command = lambda: self.dibujar_imagen(lienzo, combo_imagenes, False
    boton_double_mirror = tkinter.Button(self.ventana, text = 'Double Mirror', command = lambda: self.dibujar_imagen(lienzo, combo_image
    boton_guardar_imagen = tkinter.Button(self.ventana, text = 'Guardar Imagen', command = self.guardar_imagen, width = 12, height = 3)

    boton_cargar.place(x = 10, y = 10)
    boton_analizar.place(x = 100, y = 10)
    boton_reportes.place(x = 190, y = 10)
    boton_salir.place(x = 280, y = 10)
    boton_original.place(x = 40, y = 130)
    boton_mirror_x.place(x = 40, y = 200)
    boton_mirror_y.place(x = 40, y = 270)
    boton_double_mirror.place(x = 40, y = 340)
    boton_guardar_imagen.place(x = 40, y = 410)

    label_imagenes = tkinter.Label(self.ventana, text = "Seleccione el nombre de la imagen a procesar:", font = ('Courier', 9))
    label_imagenes.place(x = 370, y = 10)
```

- Método analizar_archivo

Este método es utilizado para realizar la instancia de los métodos analizar_estados, guardar_imagen y configuración_combo

```
def analizar_archivo(self, combo_imagenes):
    self.lexico.analizador_estados(self.data)
    self.lexico.guardar_imagen()
    self.configuracion_combo(combo_imagenes)
```

- Método dibujar_imagen

Este método primero guardar en una variable nombre el texto del combo, luego evalúa si el nombre está vacío, si lo está muestra un mensaje en pantalla, si no lo está ingresa y realiza la instancia del método borrar_imagen y seguido del método graficar imagen para dibujar la imagen en la ventana

```
def dibujar_imagen(self, lienzo, combo_imagenes, mirror_x, mirror_y, double_mirror):
    nombre = combo_imagenes.get()
    if nombre != '':
        self.borrar_imagen(lienzo)
        self.lexico.graficar_imagen(lienzo, 400, 400, mirror_x, mirror_y, double_mirror, nombre, tkinter)
    else:
        messagebox.showinfo(message = "Seleccione una imagen", title = "Alerta")
```

- Método guardar_imagen

Este método es usado para hacer la instancia del método guardar imagen de la clase Analizador si y solo si el nombre del combo box no es vacío, si lo es le muestra un mensaje al usuario.

```
def guardar_imagen(self):
    #self.lexico.crear_reporte_imagen()
    if self.nombre != '':
        self.lexico.guardar_imagen_png(self.nombre)
    else:
        messagebox.showinfo(message = "Seleccione una imagen", title = "Alerta")
```

- Método borrar_imagen

Este método se usa para borrar el contenido dentro del lienzo para ello se usa el método delete y se le dice a tkinter que borre todo.

```
def borrar_imagen(self, lienzo):
    lienzo.delete(tkinter.ALL)
```

- Método crear_reportes

Este método hace la instancia de los métodos para crear reportes

```
def crear_reportes(self):
    self.lexico.obtener_tokens()
    self.lexico.obtener_errores()
    self.lexico.crear_reporte_token()
    self.lexico.crear_reporte_errores()
```

- 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.

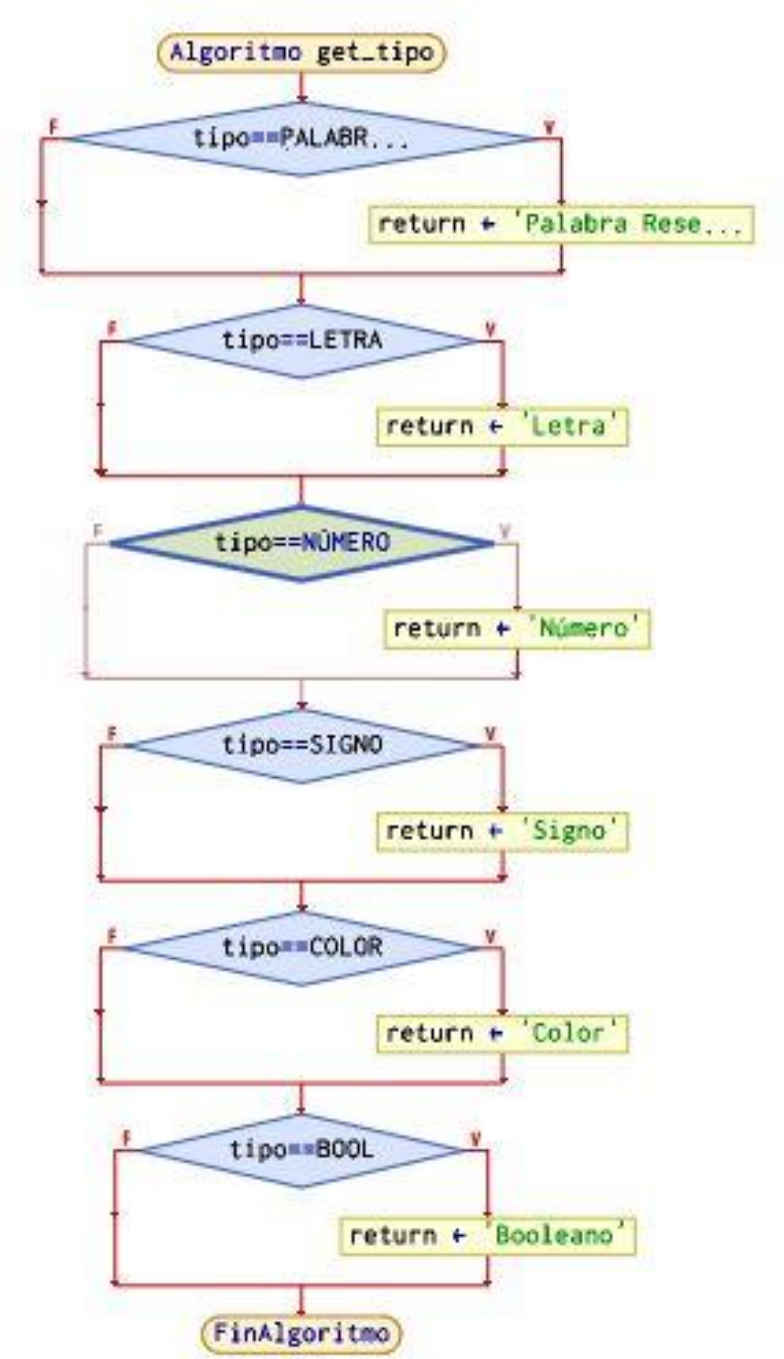
```
def leer_archivo(self):
    try:
        ruta = filedialog.askopenfilename(title = "Abrir un archivo")
        with open(ruta, 'rt', encoding = 'utf-8') as f:
            print('Archivo cargado con éxito')
            self.data = f.read()
    except OSError:
        print("<<< No se pudo leer el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

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

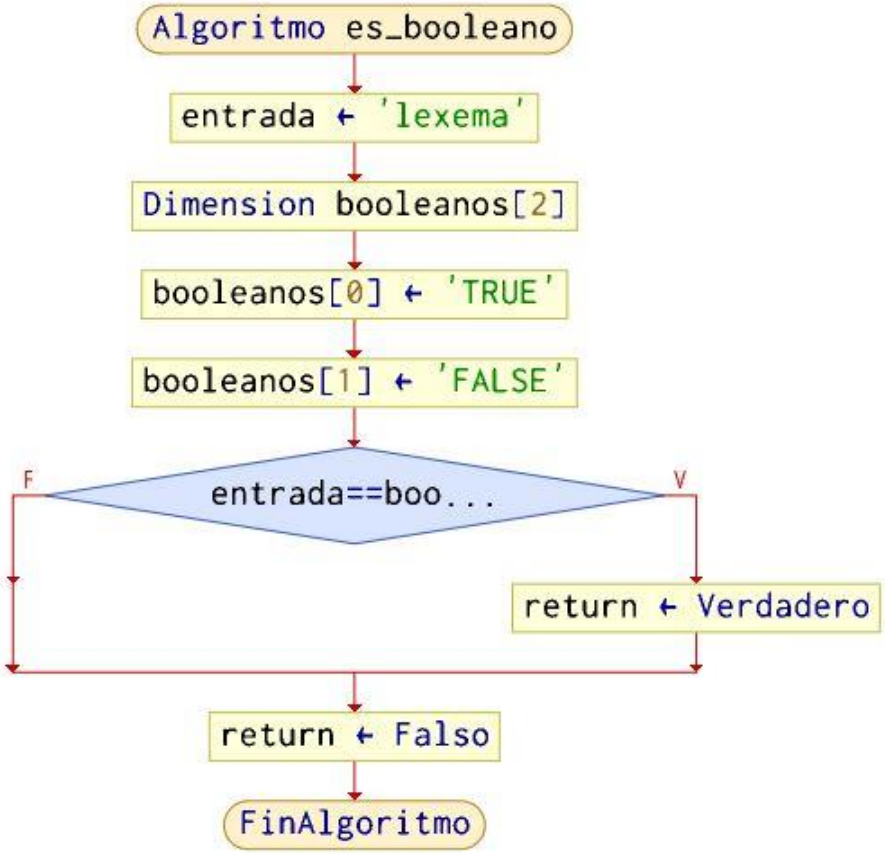
```
if __name__ == '__main__':
    Interfaz()
```

Diagramas de Flujo

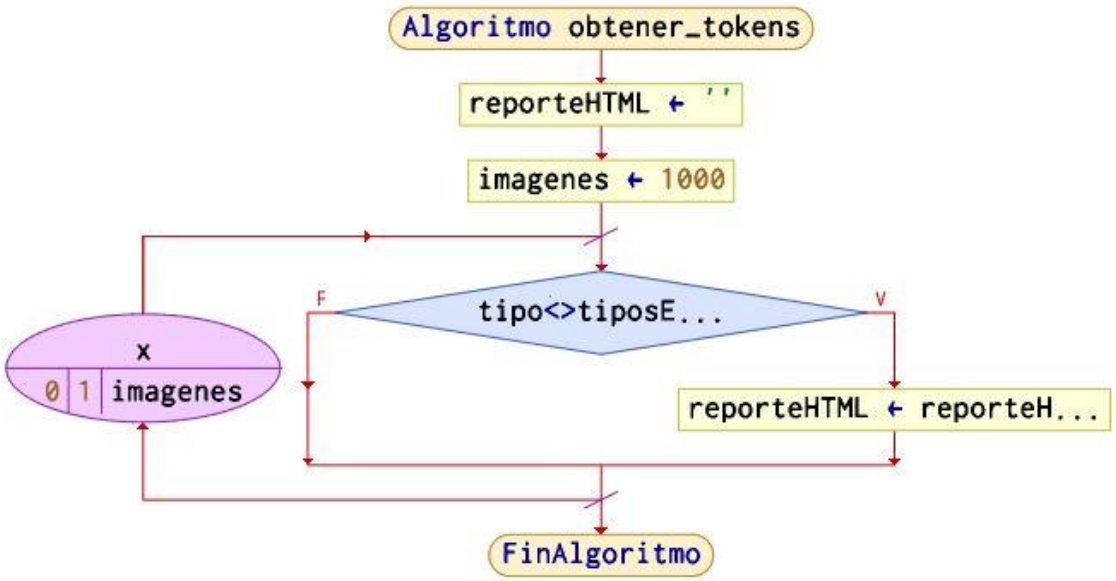
- Método get_tipo



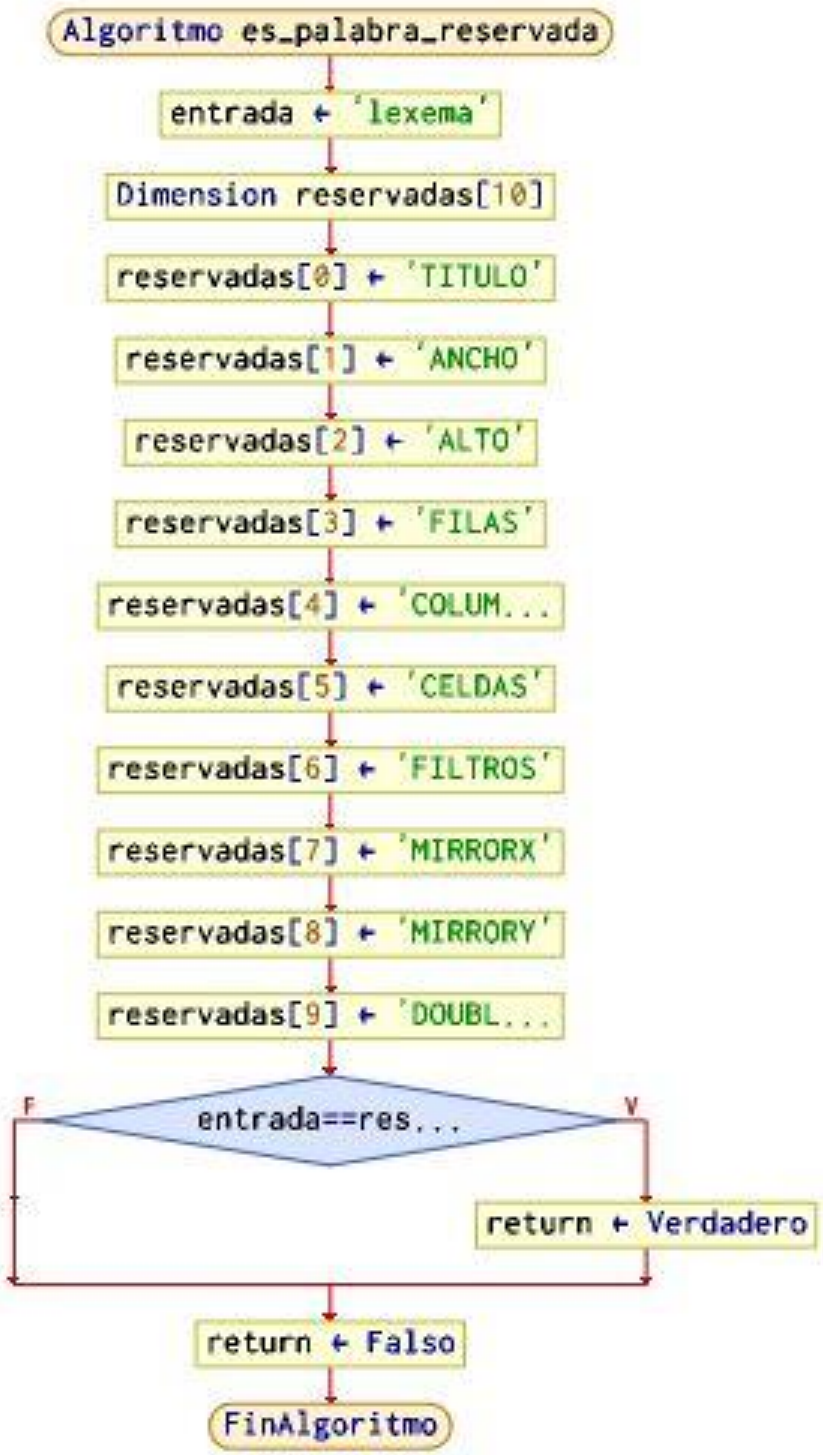
- Método es_booleano



- Método obtener_tokens



- Método es_palabra_reservada



Expresión Regular

PALABRA RESERVADA = {TITULOS, ANCHO, ALTO, FILAS, COLUMNAS, CELDAS, TRUE, FALSE, FILTROS, MIRRROX, MIRRORY, DOUBLEMIRROR}

$$\text{LETRA} = L = \{A - Z\} = L^{+} \quad \rightarrow \quad L^{+}\#1$$

$$\text{NÚMERO} = D = \{0 - 9\} = D^{+} \quad \rightarrow \quad D^{+}\#2$$

$$\text{RESTO} = \ R = \{\text{cualquier símbolo}\}$$

$$\text{CADENAS} = "(L|D|R)^{*}" \quad \rightarrow \quad "(L|D|R)^{*}"\ \#3$$

$$\text{SIGNOS} = S$$

{corchete izquierdo, corchete derecho, llave izquierda, llave derecha, igual, coma, punto y coma, arroba}

$$\rightarrow \quad S^{+}\#4$$

$$\text{NUMERAL} = NM = \#$$

$$\text{COLORES} = NM(D|L)^{+} \quad \rightarrow \quad NM(D|L)^{+}\#5$$

$$\text{EXPRESIÓN REGULAR} = \ [\ L^{+} \mid D^{+} \mid "(L|D|R)^{*}" \mid S^{+} \mid NM(D|L)^{+}]$$

Token	Expresión Regular
LETRA L	L^{+}
NÚMERO D	D^{+}
CADENAS	$"(L\|D\|R)^{*}"$
SIGNOS S	S^{+}
COLORES	$NM(D\|L)^{+}$
	$[\ L^{+} \mid D^{+} \mid "(L\|D\|R)^{*}" \mid S^{+} \mid NM(D\|L)^{+}]$

Método del Árbol

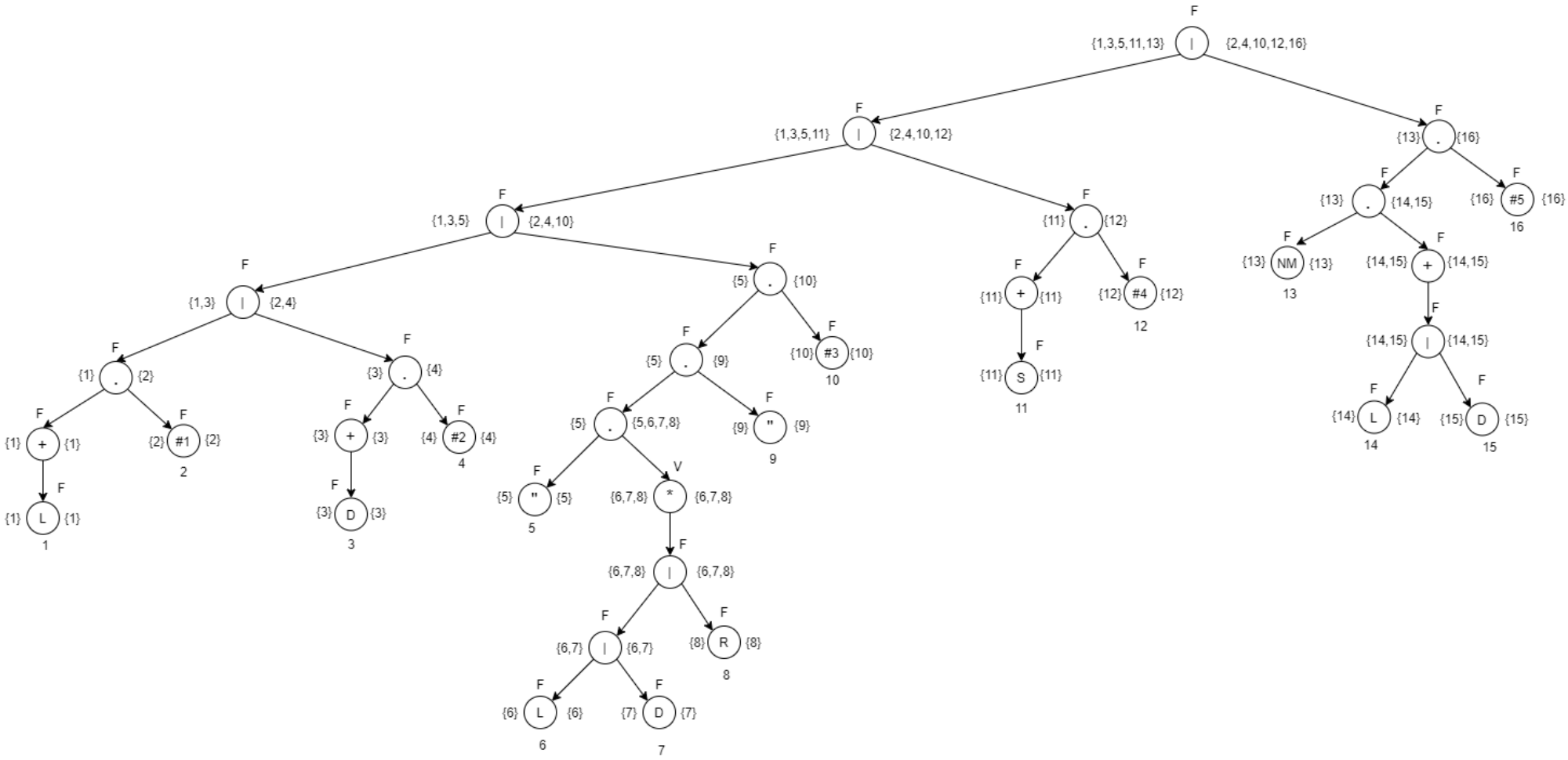


Tabla de Siguientes

i	Terminal	Sig(i)
1	L	1,2
2	#1	
3	D	3,4
4	#2	
5	"	6,7,8,9
6	L	6,7,8,9
7	D	6,7,8,9
8	R	6,7,8,9
9	"	10
10	#3	
11	S	11,12
12	#4	
13	NM	14,15
14	L	14,15,16
15	D	14,15,16
16	#5	

Tabla de Estados

So = {1, 3, 5, 11, 13}
L D " S NM
Sig(L) = Sig(1) = {1, 2} = S1
Sig(D) = Sig(3) = {3, 4} = S2
Sig(") = Sig(5) = {6, 7, 8, 9} = S3
Sig(S) = Sig(11) = {11, 12} = S4
Sig(NM) = Sig(13) = {14, 15} = S5
S1 = {1, 2}
L #1
Sig(L) = Sig(1) = {1, 2} = S1
S2 = {3, 4}
D #2
Sig(D) = Sig(3) = {3, 4} = S2
S3 = {6, 7, 8, 9}
L D R "
Sig(L) = Sig(6) = {6, 7, 8, 9} = S4
Sig(D) = Sig(7) = {6, 7, 8, 9} = S4
Sig(R) = Sig(8) = {6, 7, 8, 9} = S4
Sig(") = Sig(9) = {10} = S6

S4 = {11, 12}
S #4
Sig(S) = Sig(11) = {11, 12] = S4
S5 = {14, 15}
L D
Sig(L) = Sig(14) = {14, 15, 16} = S7
Sig(D) = Sig(15) = {14, 15, 16} = S7
S6 = {10}
#3
S7 = {14, 15, 16}
L D #5
Sig(L) = Sig(14) = {14, 15, 16} = S7
Sig(D) = Sig(15) = {14, 15, 16} = S7

Estado	L	#1	D	#2	R	"	#3	S	#4	NM	#5
S0	S1		S2			S3		S4		S5	
*S1	S1										
*S2			S2								
S3	S4		S4		S4	S6					
*S4								S4			
S5	S7		S7								
*S6											
*S7	S7		S7								

Autómata Finito Determinista