MANUAL TÉCNICO

Proyecto No.2

Por: Mario Ernesto Marroquín Pérez-202110509

INDICE

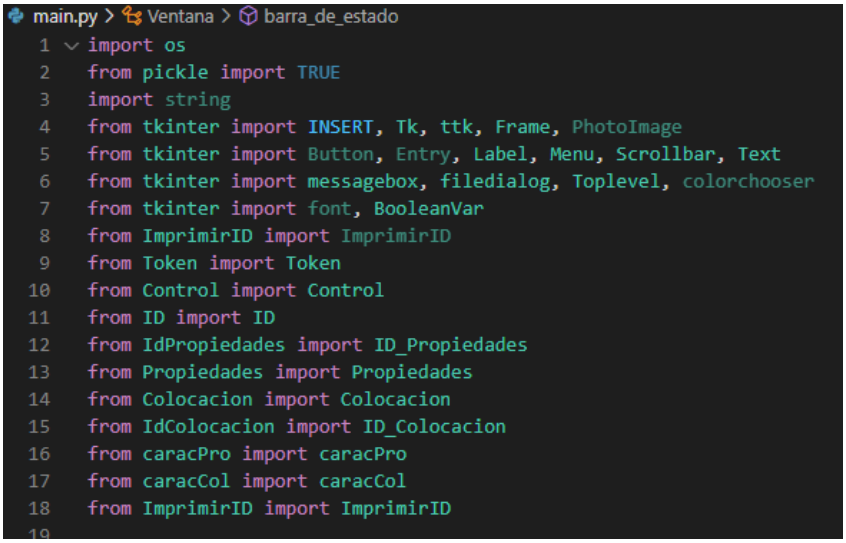
INTRODUCCION

El programa fue desarrollado en el lenguaje Python, haciendo uso de programación orientada a objetos (POO), el software es capaz de identificar un lenguaje dado, identificando los errores léxicos y ejecutando las instrucciones correspondientes, incluyendo una interfaz gráfica agradable e intuitiva para el usuario.

El programa se encuentra organizado y estructurado por clases, funciones de retorno y variables globales.

LIBRERIAS

Dentro del programa se importa una serie de librerías las cuales son utilizadas para la lectura de datos, la creación de la GUI (Interfaz Gráfica de Usuario por sus siglas en inglés). Estas librerías se importan utilizando la palabra “import”

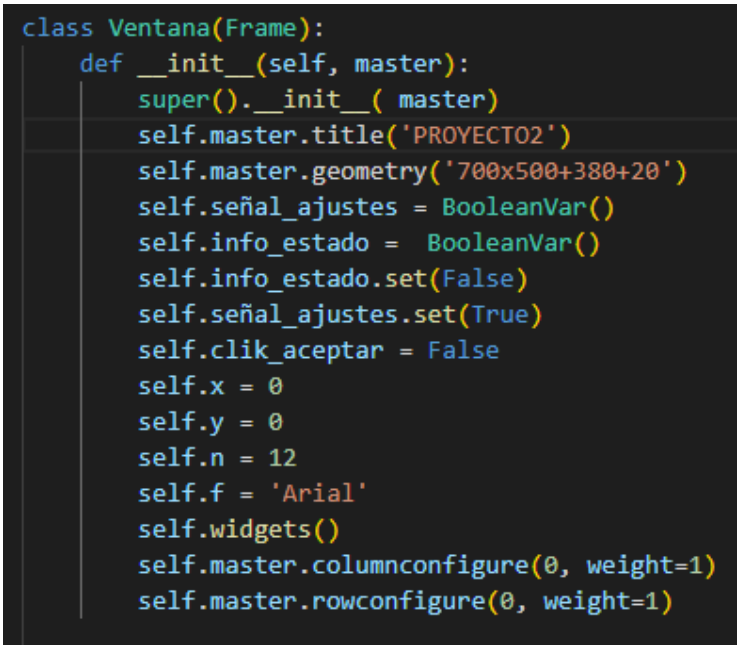


CREACION DE VENTANAS

Para la creación de ventanas se utilizó Programación Orientada a Objetos, utilizando la herencia y colocando un frame detrás del otro al momento de cambiar entre ventanas, mejorando así la experiencia del usuario al utilizar la ventana.

Creación de ventanas:

En la clase “Ventana” se configuran la ventana de forma general y se importan para su creación, dentro de esta ventana se agrega el nombre y las configuraciones.



Pestañas Dentro de la Ventana:

Dentro de la clase “widgets” se configuran las pestañas dentro de la ventana principal, estas pestañas se encuentran en la parte superior de la interfaz gráfica.

```
1272    def widgets(self):
1273        #configuramos el menú superior
1274        menu = Menu(self.master)
1275        self.master.config(menu = menu)
1276
1277        #pestaña de archivo
1278        archivo = Menu(menu, tearoff=0)
1279        archivo.add_command(label="Nuevo", command = self.nueva_ventana)
1280        archivo.add_command(label="Abrir", command = self.abrir_archivo)
1281        archivo.add_command(label="Guardar", command = self.guardar_archivo)
1282        archivo.add_command(label="Guardar Como", command = self.guardar_archivoComo)
1283        archivo.add_separator()
1284        archivo.add_command(label="Salir", command = self.master.quit)
1285
1286        #pestaña de ver
1287        ver = Menu(menu, tearoff=0)
1288        ver.add_checkbutton(label="Barra de estado", variable = self.info_estado, command = self.barra_de_estado)
1289
1290        #pestaña de ayuda
1291        ayuda = Menu(menu, tearoff=0)
1292        ayuda.add_command(label="Acerca de", command= self.acerca_de)
1293        ayuda.add_command(label="Manual de Usuario", command = self.manualUsu)
1294        ayuda.add_command(label="Manual Tecnico", command = self.manualTec)
1295
```

CLASES DENTRO DEL PROGRAMA

Cada clase dentro del programa representa una acción, configuración, estilo o la posición del frame, el contenido dentro de estas clases pueden ser configuraciones adicionales propias de esa ventana, objetos dentro de la ventana, una función de reconocimiento o generación de HTML.

```
31  class Analizar:
32      caracteres = ["a","b","c","d","e","f","g","h","i","j",
33                    "k","l","m","n","o","p","q","r","s","t",
34                    "u","v","w","x","y","z","1","2","3","4",
35                    "5","6","7","8","9","0","A","B","C","D",
36                    "E","F","G","H","I","J","K","L","M","N",
37                    "O","P","Q","R","S","T","U","V","W","X",
38                    "Y","Z",'"']
39      etiquetas = ["Controles","propiedades","Colocacion"]
40      controles = ["Etiqueta","Boton","Check","RadioBoton","Texto","AreaTexto","Clave", "Contenedor", "this"]
41      propiedades = ["setColorLetra","setTexto","setAlineacion","setColorFondo","setMarcada","setGrupo","setAlto","setAn
42      signos = [",",";","=","(",")","{","}","[","]", ":", ".", "<", ">", "+", "-", "*", "/","!","-"]
43      colocacion1 = ["setPosicion","add"]
44      tablaTokens = []
45      tabla_errores = []
46      tablaControles = []
47      tablaPropiedades = []
48      tablaColocacion = []
49      tablaID = []
50      tablaID2=[]
51      tablaIDPropiedades = []
52      tablaIDColocacion = []
53      tablacaracPro=[]
54      tablacaracCol=[]
55
```

Las clases dentro del programa todas son utilizadas de la misma manera, para crear ventanas o guardar los datos de entrada (ingresados a través del archivo), las clases que guardan las

operaciones se encuentran en un archivo de Python diferente al principal, siempre dentro de la misma carpeta.

Excepcion.py > ...

```python
class Excepcion:

    def __init__(self, error, tipo, columna, fila):
        self.error = error
        self.tipo = tipo
        self.columna = columna
        self.fila = fila

```

Clase Token

Dentro de la clase Token se almacena la columna, fila y el lexema de entrada.

Token.py > Token > __init__

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

FUNCIONES DENTRO DEL PROGRAMA

Funciones de Guardar Datos

Estas funciones se encargan de almacenar los diferentes datos ingresados, se almacenan las configuraciones respectivas, los colores, la posición, el ID entre otro tipo de información necesaria para el análisis.

```python
def guardar_IDs(self, lexema):
    #nuevo_ID = ImprimirID(lexema)
    self.tablaImprimirID.append(lexema)

def guardar_IDs2(self, lexema):
    self.tablaID2.append(lexema)

def guardarPalabras(self, lexema):
    self.tablaPalabras.append(lexema)


def guardar_token(self, lexema):
    nuevo_token = Token(self.fila, self.columna, lexema)
    self.tablaTokens.append(nuevo_token)

------------------------------------------------
def guardarControles(self, control):
    nuevo_control = Control(control)
    self.tablaControles.append(nuevo_control)
```

Función Guardar Tokens

Dentro de esta función guardamos los tokens recuperados de nuestro autómata, así como la fila, columna y el lexema recuperado, estos tokens se guardan en un arreglo.

```
def guardar_token(self, lexema):
    nuevo_token = Token(self.fila, self.columna, lexema)
    #self.tabla_tokens.clear()
    self.tabla_tokens.append(nuevo_token)
```

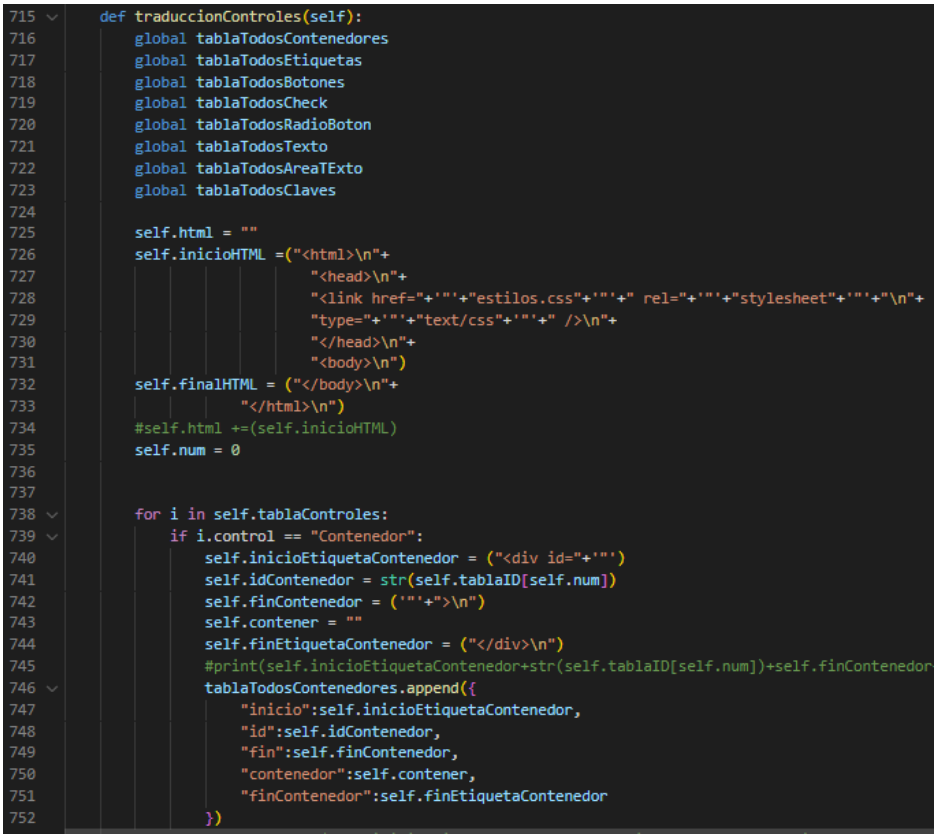
Función Imprimir_tokens

Esta función es la encargada de separar los tokens según su tipo, pueden ser Numero, ID, etc. Esta función guarda los tokens ya separados según su tipo y los almacena en un arreglo para posteriormente utilizar dicho arreglo en la tabla de tokens.

```
651     def imprimir_tokens(self):
652         #print('-'*31)
653         #print ("| {:<10} | {:<10} |{:<4} | {:<7} | {:<10} |"
654         #print('-'*45)
655         self.correlativo = 1
656         self.TipoTokens = ""
657         for token in self.tablaTokens:
658             if token.lexema in self.controles:
659                 self.TipoTokens = "Control"
660             elif token.lexema in self.etiquetas:
661                 self.TipoTokens = "Etiqueta"
662             elif token.lexema in self.signos:
663                 self.TipoTokens = "Signo"
664             elif token.lexema in self.propiedades:
665                 self.TipoTokens = "Propiedades"
666             elif token.lexema in self.colocacion1:
667                 self.TipoTokens = "Colocacion"
668             elif token.lexema.isnumeric() == True:
669                 self.TipoTokens = "Numero"
670             elif token.lexema in self.tablaImprimirID:
671                 self.TipoTokens = "ID"
672             elif token.lexema in self.tablaPalabras:
673                 self.TipoTokens = "Palabras"
674             else:
675                 self.TipoTokens = ""
676             tablaTodosTokens.append({"co":self.correlativo,
677                                      "Tk":self.TipoTokens,
678                                      "fl":token.fila,
679                                      "cl":token.columna,
680                                      "lx":token.lexema})
681             #print ("| {:<10} | {:<10} | {:<4} | {:<7} | {:<1
682             self.correlativo += 1
683         #print('-'*45)
684         self.correlativo = 1
685         self.tablaTokens.clear()
686         self.tablaImprimirID.clear()
```

Función traduccionControles

Esta función es la encargada de recolectar los datos posteriormente analizados y transformarlos a HTML, así como verificar si alguno de los objetos va dentro de los contenedores.



```
715     def traduccionControles(self):
716         global tablaTodosContenedores
717         global tablaTodosEtiquetas
718         global tablaTodosBotones
719         global tablaTodosCheck
720         global tablaTodosRadioBoton
721         global tablaTodosTexto
722         global tablaTodosAreaTExto
723         global tablaTodosClaves
724 
725         self.html = ""
726         self.inicioHTML =("<html>\n"+
727                           "<head>\n"+
728                           "<link href="+'"'+"estilos.css"+'"'+" rel="+'"'+"stylesheet"+'"'+"\n"+
729                           "type="+'"'+"text/css"+'"'+" />\n"+
730                           "</head>\n"+
731                           "<body>\n")
732         self.finalHTML = ("</body>\n"+
733                           "</html>\n")
734         #self.html +=(self.inicioHTML)
735         self.num = 0
736 
737 
738         for i in self.tablaControles:
739             if i.control == "Contenedor":
740                 self.inicioEtiquetaContenedor = ("<div id="+'"')
741                 self.idContenedor = str(self.tablaID[self.num])
742                 self.finContenedor = ('"'+">\n")
743                 self.contener = ""
744                 self.finEtiquetaContenedor = ("</div>\n")
745                 #print(self.inicioEtiquetaContenedor+str(self.tablaID[self.num])+self.finContenedor
746                 tablaTodosContenedores.append({
747                     "inicio":self.inicioEtiquetaContenedor,
748                     "id":self.idContenedor,
749                     "fin":self.finContenedor,
750                     "contenedor":self.contener,
751                     "finContenedor":self.finEtiquetaContenedor
752                 })
```

Función GuardarCom

Esta función es la encargada de guardar el archivo de la forma “Guardar Como”, utiliza el 1filedialog de Python.

```
def guardarCom(self):
    files = [('Text Document', '*.txt'),
             ('All Files', '*.*'),
             ('Python Files', '*.py')]
    file = filedialog.asksaveasfile(filetypes = files, defaultextension = files)
    with open(direccion, 'r') as f:
        cont2 = f.read()
    file.write(cont2)
    messagebox.showwarning(message="Se ha guardado el Archivo", title="FELICIDADES")
```

Función abrirManualUsuario

Esta función es la encargada de abrir el manual de usuario directamente utilizando un botón dentro de la aplicación, la función obtiene la ruta donde se encuentra el manual y lo abre en el lector de PDF preestablecido por el sistema. Tanto para abrir el manual de usuario como el manual técnico las funciones tienen la misma lógica, solo cambia la ruta del archivo.

```
def manualUsu(self):
    directorio =  os.getcwd()
    os.startfile(directorio+"\Manuales\ManualUsuario.pdf")

def manualTec(self):
    directorio =  os.getcwd()
    os.startfile(directorio+"\Manuales\ManualTecnico.pdf")
```

Función traduccionACss

Esta función es la encargada de recolectar los datos analizados y transformar la configuración correspondiente a CSS

```
def traduccionACss(self):
    self.css = ""
    self.css1 = ""
    self.css2 = ""
    self.inicioCss = "#"
    self.numAncho = 0
    self.numAlto = 0
    self.numColorFondo = 0
    self.numColorLetra = 0
    self.numPos1 = 0
    self.numPos2 = 0
    self.ancho = ""
    self.alto = ""
    self.colorFondo = ""
    self.colorLetra = ""
    self.IDcss = ""
    self.pos1 = ""
    self.pos2 = []
    self.Pos1 = ""
    self.Pos2 = ""
    #necesito el ID
    for a in self.tablaPropiedades:
        if a.propiedades == "setAncho":
            self.IDcss = str(self.tablaIDPropiedades[self.numAncho])
            self.ancho = str(self.tablacaracPro[self.numAncho])
        if a.propiedades == "setAlto":
            self.alto = str(self.tablacaracPro[self.numAlto])
        if a.propiedades == "setColorFondo":
            self.colorFondo = str(self.tablacaracPro[self.numColorFondo])
        if a.propiedades == "setColorLetra":
            self.colorLetra = str(self.tablacaracPro[self.numColorLetra])
        self.numAncho += 1
        self.numAlto += 1
        self.numColorFondo += 1
        self.numColorLetra += 1

        self.css +=(self.inicioCss+self.IDcss+"{"+"position: absolute; \n width:"+self.a
            +self.colorLetra+"); \n font-size: 12px;}\n")
    self.numAncho = 0
```

Función generarHTML

Esta función es la encargada de escribir el HTML generado en un archivo, si este archivo no existe el programa lo crea y escribe las líneas generadas, así mismo dentro de esta función se escribe el archivo CSS y se lanza un aviso cuando este proceso se encuentra terminado.

```
def generarHTML(self):
    self.archivo = open("index.html", "w")
    self.archivo.write(self.traduccionControles())
    self.archivo.close()
    #self.traColocacion()
    self.archivoCss = open("estilos.css", "w")
    self.archivoCss.write(self.traduccionACss())
    self.archivoCss.close()
    messagebox.showwarning(message="Generación Completa!", title="FELICIDADES")
    directorio =  os.getcwd()
    os.startfile(directorio+"\index.html")
```

Función barra_de_estado

Esta función es la encargada de mostrar en pantalla la posición del cursor en tiempo real, en esta función se realizan los respectivos cálculos para obtener dicha posición, se configura el tipo de letra y la posición que ocupara la información en la pantalla principal.

```
    def barra_de_estado(self):
        if self.info_estado.get() == True:
            n = len(self.texto.get('1.0','end'))

            self.barra_estado.grid(column=0, row = 2, sticky='ew')
            #self.barra_estado.config(text = f'Numero de letras: {n}' )

        x = self.barra_estado.after(10, self.barra_de_estado)

        if self.info_estado.get() == False:
            self.barra_estado.after_cancel(x)
            self.barra_estado.grid_forget()
        (self.fila, self.col) = self.texto.index(INSERT).split('.')
        self.colum = int(self.col) + 1
        #print(f'Fila: {self.fila} Columna: {str(self.colum)}')
        self.barra_estado.config(text = f'Fila: {self.fila} Columna: {str(self.colum)}' )
```

Función abrir_archivo

Esta función es la encargada de abrir el archivo seleccionado según la ruta que le indiquemos a nuestra aplicación.

```
def abrir_archivo(self):
    direccion = filedialog.askopenfilename(initialdir ='/',
        title='Archivo', filetype=(('txt files', '*.txt*'),('All files', '*.*')))

    if direccion != '':
        archivo = open(direccion, 'r')
        contenido = archivo.read()
        self.texto.delete('1.0', 'end')
        self.texto.insert('1.0', contenido)
        #self.master.title(direccion)
        self.ruta = direccion
```

Función guardar_archivo

Esta función es la encargada de guardar los cambios realizados en el archivo, para que realice esta acción es necesario pulsar el botón “guardar” dentro de la pestaña “Archivo”.

```
def guardar_archivo(self):
    try:
        archivo = open(self.ruta, 'w')
        archivo.write(self.texto.get('1.0', 'end'))
        archivo.close()
        messagebox.showinfo('Guardar Archivo','Archivo guardado')
    except:
        messagebox.showerror('Guardar Archivo', 'ERROR: Archivo no guardado')
```

Función nueva_ventana

Esta función es la encargada de borrar los datos de la ventana principal para así comenzar un documento nuevo, antes de realizar esta función la aplicación le pregunta al usuario si desea guardar el documento.

```
def nueva_ventana(self):
    if self.texto.get !=" ":
        valor = messagebox.askyesno('Proyecto2', '¿Desea guardar el archivo?',parent= self.master)
        if valor == True:
            self.guardar_archivoComo()
            self.ruta = ''
            self.texto.delete('1.0', 'end')
        else:
            self.texto.delete('1.0', 'end')
            self.ruta = ''
    else:
        self.texto.delete('1.0', 'end')
        self.ruta = ''
```

Función acerca_de

Esta función es la encargada de crear una ventana nueva donde se presentan los datos del desarrollador de la aplicación.

```
def acerca_de(self):
    vent_info = Toplevel(self.master)
    vent_info.config( bg='white')
    vent_info.title('Información')
    vent_info.resizable(0,0)
    #vent_info.iconbitmap('icono.ico')
    vent_info.geometry('290x100+200+200')
    Label(vent_info, bg='white',
        text= 'Curso: Lab. Lenguajes Formales y de Programación \n Nombre: Mario Ernesto Marroquín Pérez \n Carné: 202110509').pack(expand=True)
```

Función Volver

Esta función es la encargada de regresar a la ventana principal de la aplicación.

```
def volver():
    vent_info.destroy()
```

Función Ver_tokens

En esta función se crea la tabla donde se visualizan los tokens reconocidos por la aplicación, esta tabla se agrega a una ventana de tkinter y se agregan los datos a dicha tabla.

```
def Ver_tokens(self):
    vent_info = Toplevel(self.master)
    vent_info.config( bg='white')
    vent_info.title('Tabla de Tokens')
    #vent_info.resizable(0,0)
    vent_info.geometry('500x260+200+200')

    def volver():
        vent_info.destroy()


    regresar =ttk.Button(vent_info,text="Regresar",command=volver)
    self.tablaVerTokens = ttk.Treeview(vent_info, columns = ('#1', '#2', '#3', '#4'))

    self.tablaVerTokens.column('#0', width = 100)
    self.tablaVerTokens.column('#1', width = 100)
    self.tablaVerTokens.column('#2', width = 100)
    self.tablaVerTokens.column('#3', width = 100)
    self.tablaVerTokens.column('#4', width = 100)

    self.tablaVerTokens.heading('#0', text = 'Correlativo')
    self.tablaVerTokens.heading('#1', text = 'Tipo de Token')
    self.tablaVerTokens.heading('#2', text = 'Fila')
    self.tablaVerTokens.heading('#3', text = 'Columna')
    self.tablaVerTokens.heading('#4', text = 'Lexema')

    self.tablaVerTokens.grid(column = 0, row = 0, sticky = 'nsew')
    regresar.grid(column = 0, row = 1, sticky = 'nsew')

    tokens = tablaTodosTokens
    for e in reversed(tokens):
        co = e["co"]
        Tk = e["Tk"]
        fl = e["fl"]
        cl = e["cl"]
        lx = e["lx"]
        self.tablaVerTokens.insert('', '0', text = (co), values = (Tk, fl, cl, lx))
```

TABLA DE TOKENS

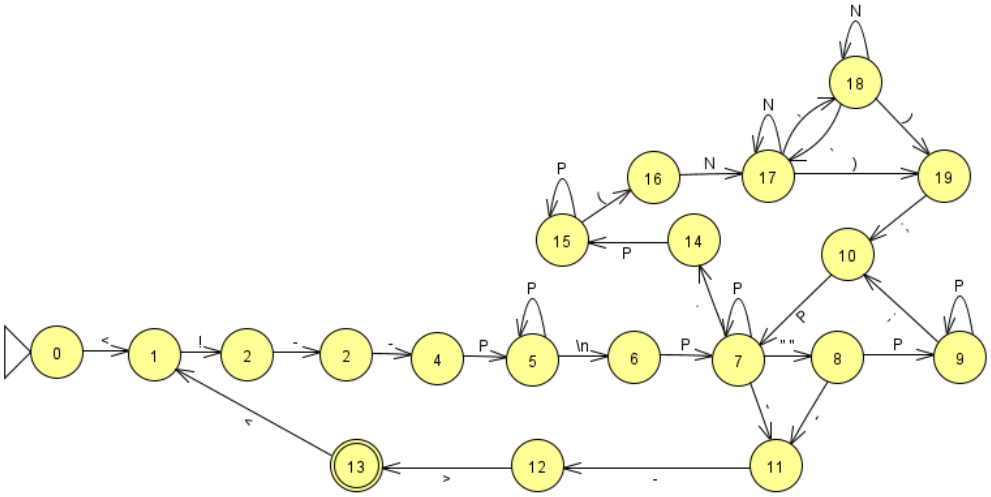
{P} = [a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z]

{N} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

TOKEN	PATRÓN
<	<
!	!
-	-
{P}	Letra seguida de letras
{N}	Números seguidos de números
>	>
.	.
;	;
,	,
(	(
)	)

AUTOMATA

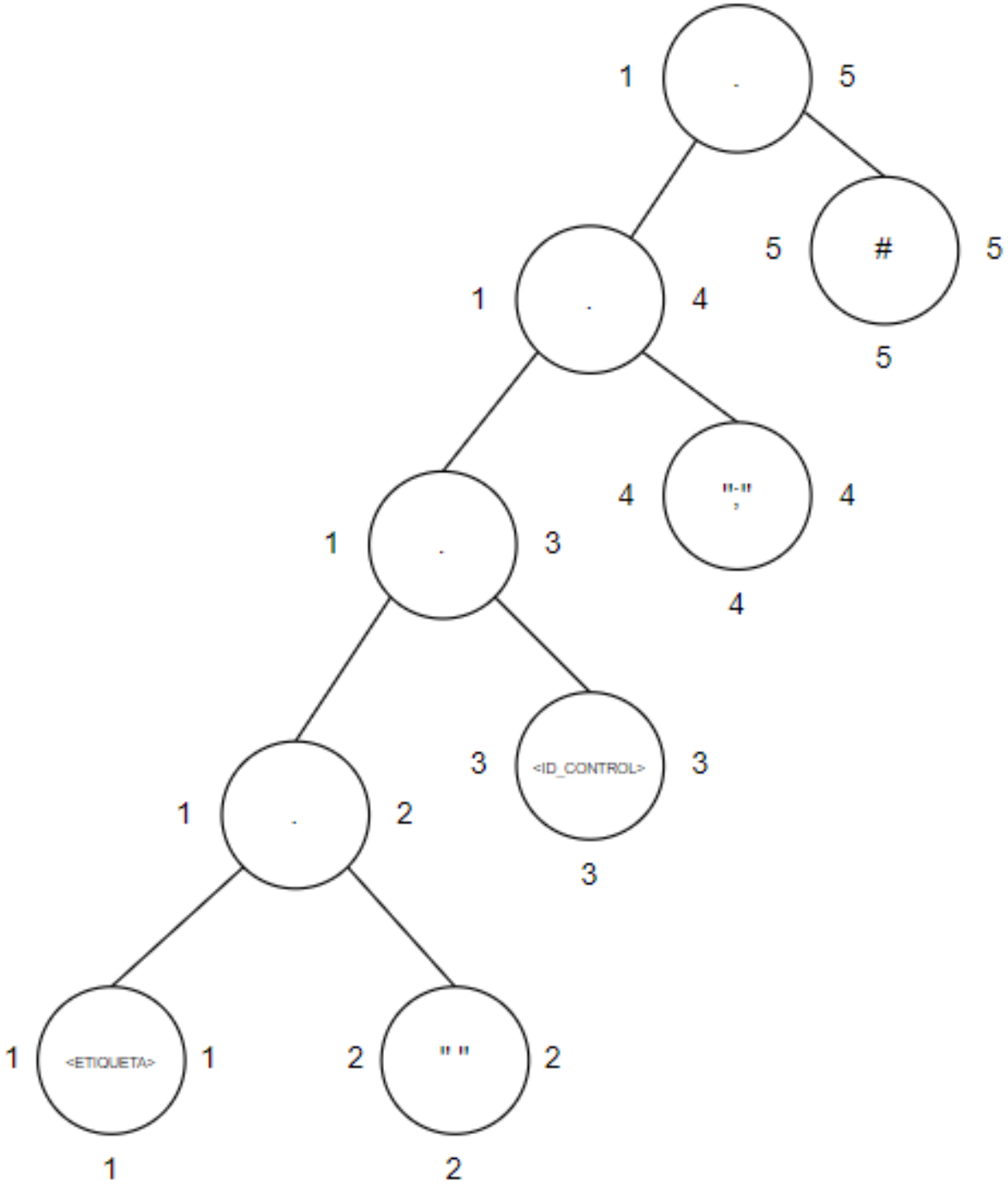
El Autómata Finito Determinista (AFD por sus siglas), diseñado para esta aplicación es el siguiente:



MÉTODO DEL ÁRBOL

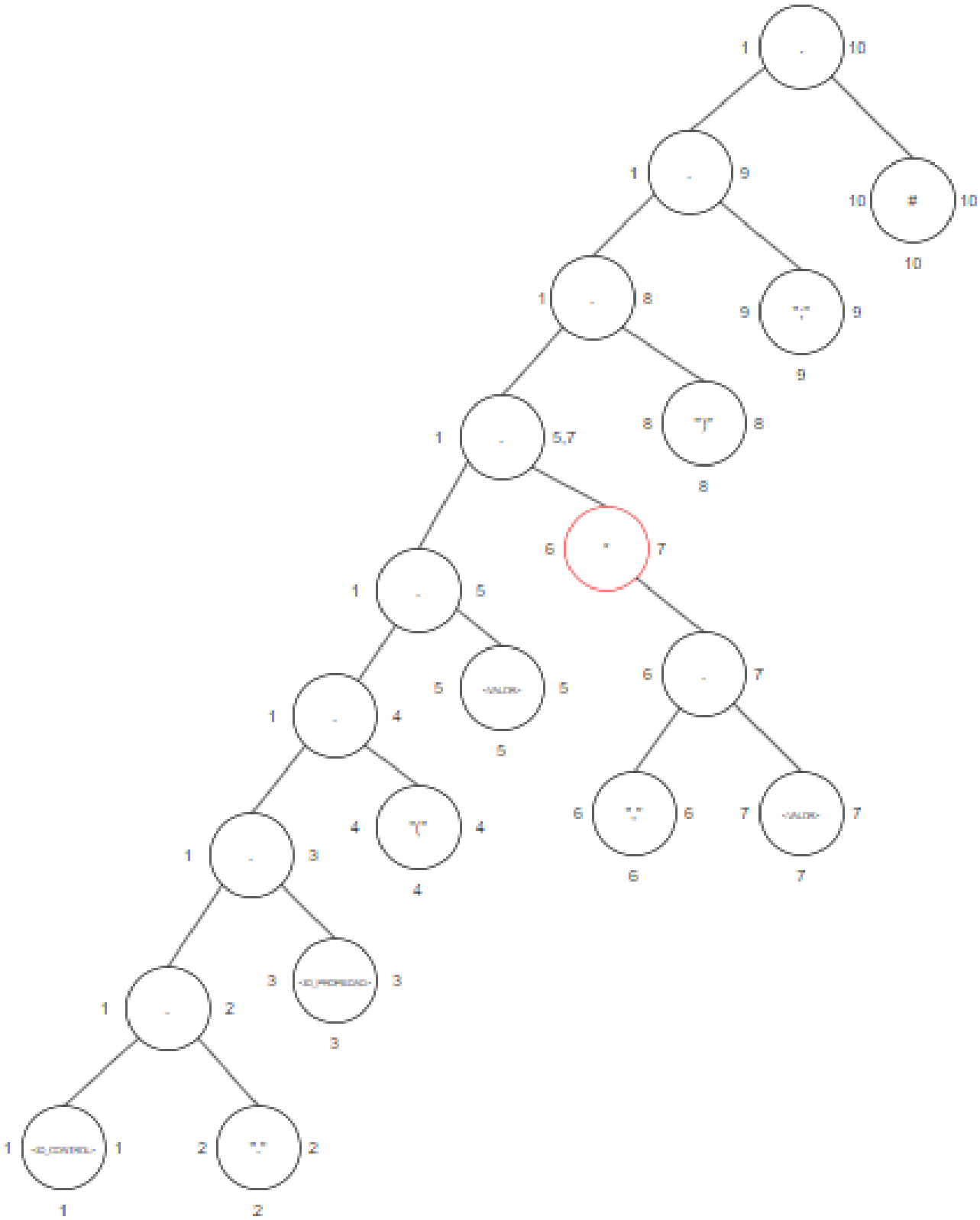
Etiqueta Control;

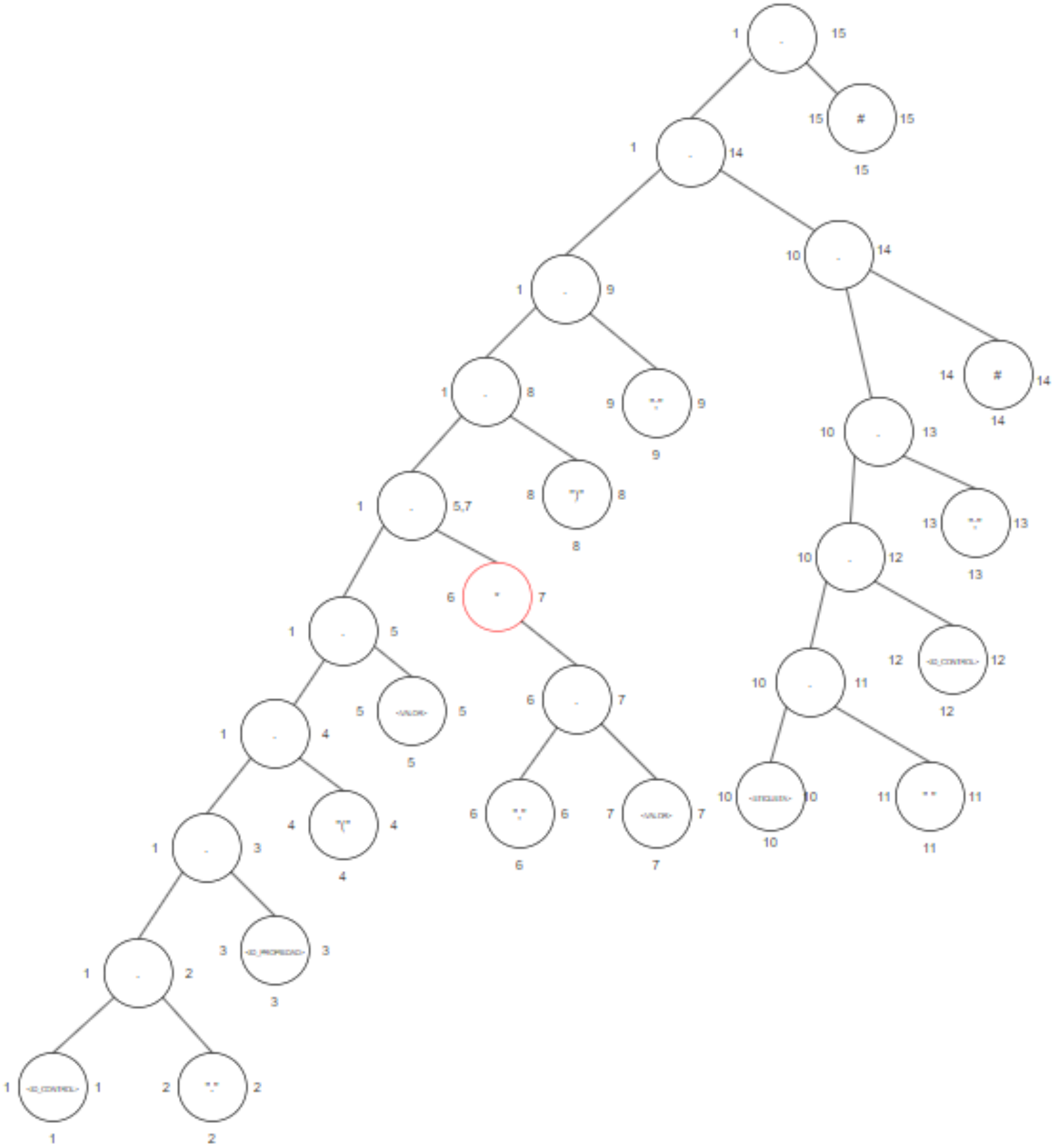
<ETIQUETA>" "<ID_CONTROL>";"#



Control.propiedad(valor[,valor]");

<ID_CONTROL>"."<ID_PROPIEDAD>"("<VALOR>(","<VALOR>)""")";"#




.
#
"."
"("
")"
*
<ID_CONTROL>
<ID_PROPIEDAD>
<VALOR>
<ETIQUETA>
" "

CÓDIGO FUENTE

```
import os
from pickle import TRUE
import string
from tkinter import INSERT, Tk, ttk, Frame, PhotoImage
from tkinter import Button, Entry, Label, Menu, Scrollbar, Text
from tkinter import messagebox, filedialog, Toplevel, colorchooser
from tkinter import font, BooleanVar
from ImprimirID import ImprimirID
from Token import Token
from Control import Control
from ID import ID
from IdPropiedades import ID_Propiedades
from Propiedades import Propiedades
from Colocacion import Colocacion
from IdColocacion import ID_Colocacion
from caracPro import caracPro
from caracCol import caracCol
from ImprimirID import ImprimirID
tablaTodosTokens = []
tablaTodosContenedores = []
tablaTodosEtiquetas = []
tablaTodosBotones = []
tablaTodosCheck = []
tablaTodosRadioBoton = []
tablaTodosTexto = []
tablaTodosAreaTExto = []
tablaTodosClaves = []
class Analizar:
 caracteres = ["a","b","c","d","e","f","g","h","i","j",
 "k","l","m","n","o","p","q","r","s","t",
 "u","v","w","x","y","z","1","2","3","4",
 "5","6","7","8","9","0","A","B","C","D",
 "E","F","G","H","I","J","K","L","M","N",
 "O","P","Q","R","S","T","U","V","W","X",
 "Y","Z",""]
 etiquetas = ["Controles","propiedades","Colocacion"]
 controles =
["Etiqueta","Boton","Check","RadioBoton","Texto","AreaTexto","
Clave", "Contenedor", "this"]
 propiedades =
["setColorLetra","setTexto","setAlineacion","setColorFondo","set
Marcada","setGrupo","setAlto","setAncho"]
 signos = [",",";","=","(",")","{","}","[","]", ":", ".", "<", ">", "+", "-
", "*", "/","!","-"]
 colocacion1 = ["setPosicion","add"]
 tablaTokens = []
 tabla_errores = []
 tablaControles = []
 tablaPropiedades = []
 tablaColocacion = []
 tablaID = []
 tablaID2=[]
 tablaIDPropiedades = []
 tablaIDColocacion = []
 tablacaracPro=[]
 tablacaracCol=[]
 tablaImprimirID = []
 tablaImprimirPropiedades = []
 tablaImprimirColocacion = []
 tablaPalabras = []
 dondeViene = 0
 global tablaTodosTokens
 fila = 0
 columna = 0
 cadena = ""
 estado_actual = 0
 estado_anterior = 0
 mas = ""
 num = 0
 numerito1 = 0
 comentarioUnilinea = False;
```

```
 comentarioMultilinea = False;
 etiquetaComentario = " # guardar *
 caracter = "
 def analizar(self, cadena):
 token = ""
 palabra = ""
 caracteristicas = ""
 carac=""
 comentarioUnilinea = False;
 comentarioMultilinea = False;
 etiquetaComentario = " # guardar *
 caracter = "
 while len(cadena)>0:
 char = cadena[0]
# ignorar espacios en blanco o saltos de linea
 if char == '\n':
 self.fila += 1
 self.columna = 0
 #cadena = cadena[1:]
 if char == ' ':
 self.columna += 1
 #cadena = cadena[1:]
if self.estado_actual == 0:
 if char == "<":
 self.guardar_token(char)
 self.estado_actual = 1
 self.estado_anterior = 0
 elif self.estado_actual == 1:
 if char == "!":
 self.guardar_token(char)
 self.estado_actual = 2
 self.estado_anterior = 1
elif self.estado_actual == 2:
 if char == "-":
 self.guardar_token(char)
 self.estado_actual = 3
 self.estado_anterior = 2
 elif self.estado_actual == 3:
 if char == "-":
 self.guardar_token(char)
 self.estado_actual = 4
 self.estado_anterior = 3
elif self.estado_actual == 4:
 if char in self.caracteres:
 token += char
 self.estado_actual = 5
 self.estado_anterior = 4
elif self.estado_actual == 5:
 if char in self.caracteres:
 token += char
 self.estado_actual = 5
 self.estado_anterior = 5
 elif char == "\n":
 self.guardar_token(token)
 #self.guardar_token(char)
 palabra = token
 self.estado_actual = 6
 self.estado_anterior = 5
 token = ""
 elif char == '/':
 if comentarioUnilinea: # se activa cuando se reconoce //
 if char == '\n':
 comentarioUnilinea = False
 continue
 else:
 continue
 # if char == '/*':
 # if comentarioMultilinea: # se activa cuando se reconoce /*
 # if etiquetaComentario == '*/':
 # comentarioMultilinea = False
 # continue
 # else:
```

```
 # if len(etiquetaComentario) == 0:
 # etiquetaComentario = char
 # elif len(etiquetaComentario) == 1:
 # etiquetaComentario += char
 # else:
 # etiquetaComentario = etiquetaComentario[1] + char
 # continue
elif self.estado_actual == 6:
 if char in self.caracteres:
 token += char
 self.estado_actual = 7
 self.estado_anterior = 6
 elif char == '/':
 if comentarioUnilinea: # se activa cuando se reconoce //
 if char == '\n':
 comentarioUnilinea = False
 continue
 else:
 continue
elif self.estado_actual == 7:
 if char in self.caracteres:
 token += char
 self.estado_actual = 7
 self.estado_anterior = 7
 elif char == " ":
 self.guardar_token(token)
 #self.guardar_token(char)
 if palabra == "Controles":
 self.guardarControles(token)
 self.estado_actual = 8
 self.estado_anterior = 7
 token = ""
elif char == "-":
 self.guardar_token(token)
 self.guardar_token(char)
 self.estado_actual = 11
 self.estado_anterior = 7
 token = ""
 palabra = ""
 elif char ==".":
self.guardar_token(token)
self.guardar_token(char)
if palabra == "propiedades":
self.guardarID_Propiedades(token)
if palabra == "Colocacion":
self.guardarID_Colocacion(token)
self.estado_actual = 14
self.estado_anterior = 7
token = ""
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
elif self.estado_actual==8:
if char == "-":
#self.guardar_token(token)
self.guardar_token(char)
self.estado_actual = 11
self.estado_anterior = 8
token = ""
#palabra = ""
elif char in self.caracteres:
token += char
self.estado_actual = 9
self.estado_anterior = 8
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
```

```
continue
else:
continue
elif self.estado_actual==9:
if char in self.caracteres:
token += char
self.estado_actual = 9
self.estado_anterior = 9
elif char == ";":
self.guardar_token(token)
self.guardar_token(char)
if palabra == "Controles":
self.guardarID(token)
self.guardar_IDs(token)
self.guardar_IDs2(token)
# if palabra == "propiedades":
# self.guardarPropiedades(token)
 # if palabra == "Colocacion":
# self.guardarColocacion(token)
self.estado_actual = 10
self.estado_anterior = 9
token = ""
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
elif self.estado_actual==10:
if char in self.caracteres:
token += char
self.estado_actual = 7
self.estado_anterior = 10
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
elif self.estado_actual==11:
if char == "-":
self.guardar_token(char)
self.estado_actual = 12
self.estado_anterior = 11
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
elif self.estado_actual==12:
if char == ">":
self.guardar_token(char)
self.estado_actual = 13
self.estado_anterior = 12
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
elif self.estado_actual==13:
if char == "<":
self.guardar_token(char)
self.estado_actual = 1
self.estado_anterior = 13
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
```

```
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
elif self.estado_actual==14:
if char in self.caracteres:
token += char
self.estado_actual = 15
self.estado_anterior = 14
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
 elif self.estado_actual==15:
if char in self.caracteres:
token += char
self.estado_actual = 15
self.estado_anterior = 15
 elif char == "(":
self.guardar_token(token)
self.guardar_token(char)
if palabra == "propiedades":
self.guardarPropiedades(token)
 if palabra == "Colocacion":
self.guardarColocacion(token)
self.estado_actual = 16
self.estado_anterior = 15
token = ""
 elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
 elif self.estado_actual==16:
if char in self.caracteres:
token += char
caracteristicas+=char
carac += char
self.estado_actual = 17
self.estado_anterior = 16
 elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
 elif self.estado_actual==17:
if char in self.caracteres:
token += char
caracteristicas+=char
carac += char
self.estado_actual = 17
self.estado_anterior = 17
elif char == ",":
 self.guardar_token(token)
 self.guardar_token(char)
 caracteristicas+=char
 carac += char
 self.estado_actual = 18
 self.estado_anterior = 17
 token = ""
elif char == ")":
self.guardar_token(token)
self.guardar_token(char)
self.estado_actual = 19
```

```
self.estado_anterior = 17
token = ""
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
elif self.estado_actual==18:
if char in self.caracteres:
token += char
caracteristicas+=char
carac += char
self.estado_actual = 18
self.estado_anterior = 18
elif char == ",":
self.guardar_token(token)
self.guardar_token(char)
caracteristicas+=char
carac += char
self.estado_actual = 17
self.estado_anterior = 18
token = ""
elif char == ")":
self.guardar_token(token)
self.guardar_token(char)
self.estado_actual = 19
self.estado_anterior = 18
token = ""
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
elif self.estado_actual==19:
if char == ";":
self.guardar_token(char)
self.estado_actual = 10
self.estado_anterior = 19
if palabra == "propiedades":
self.guardarcaracPro(caracteristicas)
self.guardarPalabras(caracteristicas)
 if palabra == "Colocacion":
self.guardarcaracCol(carac)
caracteristicas = ""
carac = ""
elif char == '/':
if comentarioUnilinea: # se activa cuando se reconoce //
if char == '\n':
comentarioUnilinea = False
continue
else:
continue
#print(self.estado_anterior, '->', self.estado_actual)
self.columna += 1
cadena = cadena[1:]
return cadena
def guardar_IDs(self, lexema):
#nuevo_ID = ImprimirID(lexema)
self.tablaImprimirID.append(lexema)
 def guardar_IDs2(self, lexema):
self.tablaID2.append(lexema)
def guardarPalabras(self, lexema):
self.tablaPalabras.append(lexema)
def guardar_token(self, lexema):
nuevo_token = Token(self.fila, self.columna, lexema)
self.tablaTokens.append(nuevo_token)
#-------------------------------------------------
def guardarControles(self, control):
```

```
nuevo_control = Control(control)
self.tablaControles.append(nuevo_control)
 def imprimirControles(self):
#self.tablaControles.pop()
for control in self.tablaControles:
print("|{:<10} |".format(control.control))
#print(control.control)
self.tablaControles.clear()
def guardarID(self, id):
nuevo_id = ID(id)
self.tablaID.append(nuevo_id)
def imprimirID(self):
#self.tablaID.pop()
print('-'*20)
for id in self.tablaID:
print("|{:<10} |".format(id.id))
#print(control.control)
self.tablaID.clear()
#-------------------------------------------------
def guardarID_Propiedades(self, propiedades):
nuevo_propiedad = ID_Propiedades(propiedades)
self.tablaIDPropiedades.append(nuevo_propiedad)
 def imprimirIDPropiedades(self):
print('-'*20)
print("ID_Propiedades")
print('-'*20)
for propiedad in self.tablaIDPropiedades:
print("|{:<10} |".format(propiedad.Id_propiedades))
#print(control.control)
print('-'*20)
self.tablaIDPropiedades.clear()
def guardarPropiedades(self, propiedad):
nuevo_propiedad = Propiedades(propiedad)
self.tablaPropiedades.append(nuevo_propiedad)
def imprimirPropiedades(self):
#self.tablaID.pop()
print('-'*20)
print("Propiedades")
print('-'*20)
for propiedad in self.tablaPropiedades:
print("|{:<10} |".format(propiedad.propiedades))
#print(control.control)
print('-'*20)
self.tablaPropiedades.clear()
def guardarcaracPro(self, caracteristicas):
nuevo_caracPro = caracPro(caracteristicas)
self.tablacaracPro.append(nuevo_caracPro)
def imprimircaracPro(self):
#self.tablaID.pop()
print('-'*20)
print("caracPro")
print('-'*20)
for caracPro in self.tablacaracPro:
print("|{:<10} |".format(caracPro.caracPro))
#print(control.control)
print('-'*20)
self.tablacaracPro.clear()
#-------------------------------------------------
def guardarID_Colocacion(self, colocacion):
nuevo_colocacion = ID_Colocacion(colocacion)
self.tablaIDColocacion.append(nuevo_colocacion)
 def imprimirIDColocacion(self):
print('-'*20)
print("ID Colocacion")
print('-'*20)
for colocacion in self.tablaIDColocacion:
print("|{:<10} |".format(colocacion.Id_colocacion))
#print(control.control)
print('-'*20)
self.tablaIDColocacion.clear()
def guardarColocacion(self, colocacion):
nuevo_colocacion = Colocacion(colocacion)
```

```
self.tablaColocacion.append(nuevo_colocacion)
def imprimirColocacion(self):
#self.tablaID.pop()
print('-'*20)
print("Colocacion")
print('-'*20)
for colocacion in self.tablaColocacion:
print("|{:<10} |".format(colocacion.colocacion))
#print(control.control)
print('-'*20)
self.tablaColocacion.clear()
def guardarcaracCol(self, carac):
nuevo_caracCol = caracCol(carac)
self.tablacaracCol.append(nuevo_caracCol)
def imprimircaracCol(self):
#self.tablaID.pop()
print('-'*20)
print("caracCol")
print('-'*20)
for caracCol in self.tablacaracCol:
print("|{:<10} |".format(caracCol.caracCol))
#print(control.control)
print('-'*20)
self.tablacaracCol.clear()
#------------------------------------------------------
 def imprimir_tokens(self):
#print('-'*31)
#print ("| {:<10} | {:<10} |{:<4} | {:<7} | {:<10}
|".format('Correlativo','Tipo de Tokens','Fila','Columna','Lexema'))
#print('-'*45)
self.correlativo = 1
self.TipoTokens = ""
for token in self.tablaTokens:
if token.lexema in self.controles:
self.TipoTokens = "Control"
elif token.lexema in self.etiquetas:
self.TipoTokens = "Etiqueta"
elif token.lexema in self.signos:
self.TipoTokens = "Signo"
elif token.lexema in self.propiedades:
self.TipoTokens = "Propiedades"
elif token.lexema in self.colocacion1:
self.TipoTokens = "Colocacion"
elif token.lexema.isnumeric() == True:
self.TipoTokens = "Numero"
elif token.lexema in self.tablaImprimirID:
self.TipoTokens = "ID"
elif token.lexema in self.tablaPalabras:
self.TipoTokens = "Palabras"
else:
self.TipoTokens = ""
tablaTodosTokens.append({"co":self.correlativo,
"Tk":self.TipoTokens,
"fl":token.fila,
"cl":token.columna,
"lx":token.lexema})
#print ("| {:<10} | {:<10} | {:<4} | {:<7} | {:<10}
|".format(self.correlativo,self.TipoTokens,token.fila,
token.columna, token.lexema))
self.correlativo += 1
#print('-'*45)
self.correlativo = 1
self.tablaTokens.clear()
self.tablaImprimirID.clear()
 def traPropiedades(self):
self.numero = 0
for i in self.tablaPropiedades:
if i.propiedades == "setColorLetra":#css
self.tablacaracPro[self.numero]
 elif i.propiedades == "setColorFondo":#css
self.tablacaracPro[self.numero]
 elif i.propiedades == "setTexto": #html
self.tablacaracPro[self.numero]
elif i.propiedades == "setAlineacion": #html
```

```
self.tablacaracPro[self.numero]
elif i.propiedades == "setMarcada":#html
self.tablacaracPro[self.numero]
 elif i.propiedades == "setAlto":#css
self.tablacaracPro[self.numero]
 elif i.propiedades == "setAncho":#css
self.tablacaracPro[self.numero]
self.numero += 1
def traduccionControles(self):
global tablaTodosContenedores
global tablaTodosEtiquetas
global tablaTodosBotones
global tablaTodosCheck
global tablaTodosRadioBoton
global tablaTodosTexto
global tablaTodosAreaTExto
global tablaTodosClaves
self.html = ""
self.inicioHTML =("<html>\n"+
"<head>\n"+
"<link href="+""+"estilos.css"+""+"
rel="+""+"stylesheet"+""+"\n"+
"type="+""+"text/css"+""+" />\n"+
"</head>\n"+
"<body>\n")
self.finalHTML = ("</body>\n"+
"</html>\n")
#self.html +=(self.inicioHTML)
self.num = 0
for i in self.tablaControles:
if i.control == "Contenedor":
self.inicioEtiquetaContenedor = ("<div id="+"")
self.idContenedor = str(self.tablaID[self.num])
self.finContenedor = (""+">\n")
self.contener = ""
self.finEtiquetaContenedor = ("</div>\n")
print(self.inicioEtiquetaContenedor+str(self.tablaID[self.num])+sel
f.finContenedor+self.mas+self.finEtiquetaContenedor)
tablaTodosContenedores.append({
"inicio":self.inicioEtiquetaContenedor,
"id":self.idContenedor,
"fin":self.finContenedor,
"contenedor":self.contener,
"finContenedor":self.finEtiquetaContenedor
})
# self.html
+=(self.inicioEtiquetaContenedor+self.idContenedor+self.finConte
nedor+self.contener+self.finEtiquetaContenedor)
elif i.control == "Etiqueta":
self.numEtiqueta = 0
self.numEtiqueta2=0
self.inicioEtiquetaEtiqueta = ("<label id="+"")
self.idEtiqueta = str(self.tablaID[self.num])
self.finEtiqueta = (""+">\n")
self.etiquetaTexto = ""
self.textEtiqueta = ""
self.textEtiqueta1 = ""
for i in self.tablaPropiedades:
if i.propiedades == "setTexto":
for a in self.tablaIDPropiedades:
if a.Id_propiedades == self.idEtiqueta:
#self.numEtiqueta2 += 1
#if self.numEtiqueta2 == 1:
self.textEtiqueta = str(self.tablacaracPro[self.numEtiqueta])
self.textEtiqueta1 = self.textEtiqueta.replace("", ")
self.numEtiqueta+= 1
self.numEtiqueta = 0
self.numEtiqueta2 = 0
self.finEtiquetaEtiqueta = ("</label>\n")
tablaTodosEtiquetas.append({
"inicio":self.inicioEtiquetaEtiqueta,
"id":self.idEtiqueta,
"fin":self.finEtiqueta,
"texto":self.textEtiqueta1,
```

```
"fin2":self.finEtiquetaEtiqueta

})

# self.html
+=(self.inicioEtiquetaEtiqueta+self.idEtiqueta+self.finEtiqueta+self
.textEtiqueta1+self.finEtiquetaEtiqueta)

elif i.control == "Boton":

self.numerit = 0

self.numerit2 = 0

self.inicioEtiquetaBoton = ("<input
type="+""+"submit"+""+"id="+"")

self.idBoton = str(self.tablaID[self.num])

self.finBoton = ("")

self.finBoton2 = ("value=")

self.textBoton = ("")

self.textoAlineacionBoton = ("left"+"")

self.numerit3 = 0

for i in self.tablaPropiedades:

if i.propiedades == "setTexto":

for a in self.tablaIDPropiedades:

if a.Id_propiedades == self.idBoton:

self.numerit3 += 1

if self.numerit3 == 1:

self.textBoton = str(self.tablacaracPro[self.numerit])

self.numerit+= 1

self.numerit = 0

self.numerit3 = 0

if i.propiedades == "setAlineacion":

for a in self.tablaIDPropiedades:

if a.Id_propiedades == self.idBoton:

self.textoAlineacionBoton = str(self.tablacaracPro[self.numerit2])

if self.textoAlineacionBoton == self.textBoton:

self.textoAlineacionBoton = ("left"+"")

if self.textoAlineacionBoton == (""+"izquierda"+""):

self.textoAlineacionBoton = ("left"+"")

elif self.textoAlineacionBoton == (""+"derecha"+""):

self.textoAlineacionBoton = ("right"+"")

elif self.textoAlineacionBoton == (""+"centro"+""):

self.textoAlineacionBoton = ("center"+"")

self.numerit2 += 1

self.numerit2 = 0

self.finBoton3 = ("style="+""+"text-align:")

self.finBoton4 = ("/>\n")

tablaTodosBotones.append({

"inicio":self.inicioEtiquetaBoton,

"id":self.idBoton,

"fin":self.finBoton,

"fin2":self.finBoton2,

"texto":self.textBoton,

"fin3":self.finBoton3,

"alineacion":self.textoAlineacionBoton,

"fin4":self.finBoton4

})

# self.html
+=(self.inicioEtiquetaBoton+self.idBoton+self.finBoton+self.finBo
ton2+self.textBoton+self.finBoton3+self.textoAlineacionBoton+sel
f.finBoton4)

elif i.control == "Check":

self.numerCheck = 0

self.numerCheck1 = 0

self.numerCheck2 = 0

self.numerCheck3 = 0

self.inicioEtiquetaCheck = ("<input
type="+""+"checkbox"+""+"id="+"")

self.idCheck = str(self.tablaID[self.num])

self.finCheck = ("")

self.marcado = ""

self.finCheck1 = ("/>")

self.textoCheck = ""

self.textoCheck2 = ""

self.finCheck2 = ("<br>\n")

for i in self.tablaPropiedades:

if i.propiedades == "setMarcada":

for a in self.tablaIDPropiedades:

if a.Id_propiedades == self.idCheck:

self.numerCheck1 += 1

if self.numerCheck1 == 1:
```

```
self.marcado = str(self.tablacaracPro[self.numerCheck])
if self.marcado.lower() ==("true"):
self.marcado = ("checked")
self.numerCheck += 1
self.numerCheck = 0
self.numerCheck1 = 0
if i.propiedades == "setTexto":
for a in self.tablaIDPropiedades:
if a.Id_propiedades == self.idCheck:
#self.numerCheck3 += 1
#if self.numerCheck3 == 1:
self.textoCheck = str(self.tablacaracPro[self.numerCheck2])
self.textoCheck2 = self.textoCheck.replace("", '')
self.numerCheck2 += 1
self.numerCheck2 = 0
#self.numerCheck3 = 0
tablaTodosCheck.append({
"inicio":self.inicioEtiquetaCheck,
"id":self.idCheck,
"fin":self.finCheck,
"marcado":self.marcado,
"fin1":self.finCheck1,
"texto":self.textoCheck2,
"fin2":self.finCheck2
})
# self.html
+=(self.inicioEtiquetaCheck+self.idCheck+self.finCheck+self.marc
ado+self.finCheck1+self.textoCheck2+self.finCheck2)
elif i.control == "RadioBoton":
self.numerRadio0 = 0
self.numerRadio00 = 0
self.numerRadio = 0
self.numerRadio1 = 0
self.numerRadio2 = 0
self.numerRadio3 = 0
self.inicioEtiquetaRadio = ("<input
type="+""+"radio"+""+"name=")
self.grupoRadio = ""
self.inicioEtiquetaRadio1 =("id="+"")
self.idRadio = str(self.tablaID[self.num])
self.finRadioID = ("")
self.marcadoRadio = ""
self.finRadio1 = ("/>")
self.textoRadio = ""
self.textoRadio2 = ""
self.finRadio2 = ("<br>\n")
for i in self.tablaPropiedades:
if i.propiedades == "setGrupo":
for a in self.tablaIDPropiedades:
if a.Id_propiedades == self.idRadio:
self.numerRadio00 += 1
if self.numerRadio00 == 1:
self.grupoRadio = str(self.tablacaracPro[self.numerRadio0])
self.numerRadio0 += 1
self.numerRadio0 = 0
self.numerRadio00 = 0
if i.propiedades == "setMarcada":
for a in self.tablaIDPropiedades:
if a.Id_propiedades == self.idRadio:
self.numerRadio1 += 1
if self.numerRadio1 == 2:
self.marcadoRadio = str(self.tablacaracPro[self.numerRadio])
if self.marcadoRadio ==("true"):
self.marcadoRadio = ("checked")
self.numerRadio += 1
self.numerRadio = 0
self.numerRadio1 = 0
if i.propiedades == "setTexto":
for a in self.tablaIDPropiedades:
if a.Id_propiedades == self.idRadio:
self.numerRadio3 += 1
if self.numerRadio3 == 3:
self.textoRadio = str(self.tablacaracPro[self.numerRadio2])
self.textoRadio2 = self.textoRadio.replace("", '')
```

```
self.numerRadio2 += 1
self.numerRadio2 = 0
self.numerRadio3 = 0
tablaTodosRadioBoton.append({
"inicio":self.inicioEtiquetaRadio,
"grupo":self.grupoRadio,
"inicio1":self.inicioEtiquetaRadio1,
"id":self.idRadio,
"fin":self.finRadioID,
"marcado":self.marcadoRadio,
"fin1":self.finRadio1,
"texto":self.textoRadio2,
"fin2":self.finRadio2
})
# self.html
+=(self.inicioEtiquetaRadio+self.grupoRadio+self.inicioEtiquetaRa
dio1+self.idRadio+self.finRadioID+self.marcadoRadio+self.finRad
io1+self.textoRadio2+self.finRadio2)
elif i.control == "Texto":
self.numerito1 = 0
self.numer = 0
self.inicioEtiquetaTexto = ("<input
type="+""+"text"+""+"id="+"")
self.idTexto = str(self.tablaID[self.num])
self.finTexto = ("")
self.finTexto2 = ("value=")
self.texto0 = (""+"")
self.textoAlineacion = ("left"+"")
self.numeri = 0
for i in self.tablaPropiedades:
if i.propiedades == "setTexto":
for a in self.tablaIDPropiedades:
if a.Id_propiedades == self.idTexto:
self.numeri += 1
if self.numeri == 1:
self.texto0 = str(self.tablacaracPro[self.numerito1])
self.numerito1 += 1
self.numerito1 = 0
self.numeri = 0
if i.propiedades == "setAlineacion":
for a in self.tablaIDPropiedades:
if a.Id_propiedades == self.idTexto:
self.textoAlineacion = str(self.tablacaracPro[self.numer])
if self.textoAlineacion == "" or self.textoAlineacion == " ":
self.textoAlineacion = ("left"+"")
if self.textoAlineacion == (""+"izquierda"+""):
self.textoAlineacion = ("left"+"")
elif self.textoAlineacion == (""+"derecha"+""):
self.textoAlineacion = ("right"+"")
elif self.textoAlineacion == (""+"centro"+""):
self.textoAlineacion = ("center"+"")
self.numer += 1
self.numer = 0
#self.numerito1 = 0
self.finTexto3 = ("style="+""+"text-align:")
#self.numerito1 = 0
self.finTexto4 = ("/>\n")
tablaTodosTexto.append({
"inicio":self.inicioEtiquetaTexto,
"id":self.idTexto,
"fin":self.finTexto,
"fin2":self.finTexto2,
"texto":self.texto0,
"fin3":self.finTexto3,
"alineacion":self.textoAlineacion,
"fin4":self.finTexto4
})
# self.html
+=(self.inicioEtiquetaTexto+self.idTexto+self.finTexto+self.finTex
to2+self.texto0+self.finTexto3+self.textoAlineacion+self.finTexto4
)
elif i.control == "AreaTexto":
self.numAreaTexto = 0
self.numAreaTexto2 = 0
self.inicioEtiquetaAreaTexto = ("<TEXTAREA id="+"")
self.idAreaTexto = str(self.tablaID[self.num])
```

```
self.finAreaTexto = (""+">\n")
self.textArea = ""
self.textArea1 = ""
for i in self.tablaPropiedades:
if i.propiedades == "setTexto":
for a in self.tablaIDPropiedades:
if a.Id_propiedades == self.idAreaTexto:
#self.numEtiqueta2 += 1
#if self.numEtiqueta2 == 1:
self.textArea = str(self.tablacaracPro[self.numAreaTexto])
self.textArea1 = self.textArea.replace("", '')
self.numAreaTexto+= 1
self.numAreaTexto = 0
self.numAreaTexto2 = 0
self.finEtiquetaAreaTexto = ("</TEXTAREA>\n")
tablaTodosAreaTExto.append({
"inicio":self.inicioEtiquetaAreaTexto,
"id":self.idAreaTexto,
"fin":self.finAreaTexto,
"texto":self.textArea1,
"fin2":self.finEtiquetaAreaTexto
})
# self.html
+=(self.inicioEtiquetaAreaTexto+self.idAreaTexto+self.finAreaTe
xto+self.textArea1+self.finEtiquetaAreaTexto)
elif i.control == "Clave":
self.numerito1Clave = 0
self.numerClave = 0
self.inicioEtiquetaClave = ("<input type="+""+"password"+""+"
id="+"")
self.idClave = str(self.tablaID[self.num])
self.finClave = ("")
self.finClave2 = (" value=")
self.textoClave = (""+"")
self.textoAlineacionClave = ("left"+"")
self.numeriClave = 0
for i in self.tablaPropiedades:
if i.propiedades == "setTexto":
for a in self.tablaIDPropiedades:
if a.Id_propiedades == self.idClave:
self.numeriClave += 1
if self.numeriClave == 1:
self.textoClave = str(self.tablacaracPro[self.numerito1Clave])
self.numerito1Clave += 1
self.numerito1Clave = 0
self.numeriClave = 0
if i.propiedades == "setAlineacion":
self.textoAlineacionClave =
str(self.tablacaracPro[self.numerClave])
if self.textoAlineacionClave == "" or self.textoAlineacionClave ==
" ":
self.textoAlineacionClave = ("left"+"")
if self.textoAlineacionClave == (""+"izquierda"+""):
self.textoAlineacionClave = ("left"+"")
elif self.textoAlineacionClave == (""+"derecha"+""):
self.textoAlineacionClave = ("right"+"")
elif self.textoAlineacionClave == (""+"centro"+""):
self.textoAlineacionClave = ("center"+"")
self.numerClave += 1
self.numerClave = 0
self.finClave3 = (" style="+""+" text-align:")
self.finClave4 = ("/>\n")
tablaTodosClaves.append({
"inicio":self.inicioEtiquetaClave,
"id":self.idClave,
"fin":self.finClave,
"fin2":self.finClave2,
"texto":self.textoClave,
"fin3":self.finClave3,
"alineacion":self.textoAlineacionClave,
"fin4":self.finClave4
})
# self.html
+=(self.inicioEtiquetaClave+self.idClave+self.finClave+self.finCla
ve2+self.textoClave+self.finClave3+self.textoAlineacionClave+sel
f.finClave4)
```

```
self.num = self.num + 1

self.numerito = 0

self.num = 0

self.numerito = 0

#self.html +=(self.finalHTML)

#self.tablacaracCol.clear()

#self.tablacaracPro.clear()

#self.tablaID.clear()

#self.tablaIDPropiedades.clear()

#self.tablaPropiedades.clear()

#self.tablaColocacion.clear()

#self.tablaIDColocacion.clear()

#self.tablaControles.clear()

self.numeroP = 0

self.Idd = ""

self.agregacion = ""

self.nom = ""

for h in self.tablaColocacion:

if h.colocacion == "add":

self.Idd = str(self.tablaIDColocacion[self.numeroP])

if self.Idd != "this":

for q in tablaTodosContenedores:

if q["id"] == self.Idd:

#print(q["id"])

self.nom = str(self.tablacaracCol[self.numeroP]) #ARREGLAR
ESTO

for i in tablaTodosContenedores:

if i["id"] == self.nom:

self.agregacion +=
(i["inicio"]+i["id"]+i["fin"]+i["contenedor"]+i["finContenedor"])

for i in tablaTodosEtiquetas:

if i["id"] == self.nom:

self.agregacion +=
(i["inicio"]+i["id"]+i["fin"]+i["texto"]+i["fin2"])

for i in tablaTodosBotones:

if i["id"] == self.nom:

self.agregacion +=
(i["inicio"]+i["id"]+i["fin"]+i["fin2"]+i["texto"]+i["fin3"]+i["alinea
cion"]+i["fin4"])

for i in tablaTodosCheck:

if i["id"] == self.nom:

self.agregacion +=
(i["inicio"]+i["id"]+i["fin"]+i["marcado"]+i["fin1"]+i["texto"]+i["fi
n2"])

for i in tablaTodosRadioBoton:

if i["id"] == self.nom:

self.agregacion +=
(i["inicio"]+i["grupo"]+i["inicio1"]+i["id"]+i["fin"]+i["marcado"]+
i["fin1"]+i["texto"]+i["fin2"])

for i in tablaTodosTexto:

if i["id"] == self.nom:

self.agregacion +=
(i["inicio"]+i["id"]+i["fin"]+i["fin2"]+i["texto"]+i["fin3"]+i["alinea
cion"]+i["fin4"])

for i in tablaTodosAreaTExto:

if i["id"] == self.nom:

self.agregacion +=
(i["inicio"]+i["id"]+i["fin"]+i["fin2"]+i["texto"]+i["fin2"])

for i in tablaTodosClaves:

if i["id"] == self.nom:

self.agregacion +=
(i["inicio"]+i["id"]+i["fin"]+i["fin2"]+i["texto"]+i["fin3"]+i["alinea
cion"]+i["fin4"])

q["contenedor"] = self.agregacion

self.numeroP += 1

self.html +=(self.inicioHTML)

for i in tablaTodosContenedores:

self.html
+=(i["inicio"]+i["id"]+i["fin"]+i["contenedor"]+i["finContenedor"]
)

self.html +=(self.finalHTML)

tablaTodosContenedores.clear()

tablaTodosEtiquetas.clear()

tablaTodosBotones.clear()

tablaTodosCheck.clear()

tablaTodosRadioBoton.clear()

tablaTodosTexto.clear()

tablaTodosAreaTExto.clear()
```

```
tablaTodosClaves.clear()
return self.html
def traColocacion(self):
self.numeroColocacion = 0
for u in self.tablaColocacion:
self.nombreColocacion = u.colocacion
if self.nombreColocacion == "add":
print(self.nombreColocacion)
self.numeroColocacion += 1
self.numeroColocacion = 0
def traduccionACss(self):
self.css = ""
self.css1 = ""
self.css2 = ""
self.inicioCss = "#"
self.numAncho = 0
self.numAlto = 0
self.numColorFondo = 0
self.numColorLetra = 0
self.numPos1 = 0
self.numPos2 = 0
self.ancho = ""
self.alto = ""
self.colorFondo = ""
self.colorLetra = ""
self.IDcss = ""
self.pos1 = ""
self.pos2 = []
self.Pos1 = ""
self.Pos2 = ""
#necesito el ID
for a in self.tablaPropiedades:
if a.propiedades == "setAncho":
self.IDcss = str(self.tablaIDPropiedades[self.numAncho])
self.ancho = str(self.tablacaracPro[self.numAncho])
if a.propiedades == "setAlto":
self.alto = str(self.tablacaracPro[self.numAlto])
if a.propiedades == "setColorFondo":
self.colorFondo = str(self.tablacaracPro[self.numColorFondo])
if a.propiedades == "setColorLetra":
self.colorLetra = str(self.tablacaracPro[self.numColorLetra])
self.numAncho += 1
self.numAlto += 1
self.numColorFondo += 1
self.numColorLetra += 1
self.css +=(self.inicioCss+self.IDcss+"{"+"position: absolute; \n
width:"+self.ancho+"px; \n height:"+self.alto+"px; \n background-
color: rgb("+self.colorFondo+"); \n color: rgb("
+self.colorLetra+"); \n font-size: 12px;}\n")
self.numAncho = 0
self.numAlto = 0
self.numColorFondo = 0
self.numColorLetra = 0
self.IDcss = ""
for b in self.tablaColocacion:
if b.colocacion =="setPosicion":
self.IDcss = str(self.tablaIDColocacion[self.numPos1])
self.pos1 = str(self.tablacaracCol[self.numPos1])
pos3 = self.pos1
self.pos2 = pos3.split(",")
self.Pos1 = self.pos2[0]
self.Pos2 = self.pos2[1]
self.numPos1 += 1
self.css1 +=(self.inicioCss+self.IDcss+"{"+"position: absolute; \n
top:"+self.Pos2+"px; \n left:"+self.Pos1+"px;}\n")
self.numPos1 = 0
self.css2 = self.css + self.css1
self.tablacaracCol.clear()
self.tablacaracPro.clear()
self.tablaID.clear()
self.tablaIDPropiedades.clear()
self.tablaPropiedades.clear()
self.tablaColocacion.clear()
self.tablaIDColocacion.clear()
```

```
 self.tablaControles.clear()
 return self.css2
def generarHTML(self):
 self.archivo = open("index.html", "w")
 self.archivo.write(self.traduccionControles())
 self.archivo.close()
 #self.traColocacion()
 self.archivoCss = open("estilos.css", "w")
 self.archivoCss.write(self.traduccionACss())
 self.archivoCss.close()
 messagebox.showwarning(message="Generación Completa!",
title="FELICIDADES")
 directorio = os.getcwd()
 os.startfile(directorio+"\index.html")
#-------------------------------------------------------------------------------
-----------------------------------
class Ventana(Frame):
 def __init__(self, master):
 super().__init__( master)
 self.master.title('PROYECTO2')
 self.master.geometry('700x500+380+20')
 self.señal_ajustes = BooleanVar()
 self.info_estado = BooleanVar()
 self.info_estado.set(False)
 self.señal_ajustes.set(True)
 self.clik_aceptar = False
 self.x = 0
 self.y = 0
 self.n = 12
 self.f = 'Arial'
 self.widgets()
 self.master.columnconfigure(0, weight=1)
 self.master.rowconfigure(0, weight=1)
 def widgets(self):
 #configuramos el menú superior
 menu = Menu(self.master)
 self.master.config(menu = menu)
 #pestaña de archivo
 archivo = Menu(menu, tearoff=0)
 archivo.add_command(label="Nuevo", command =
self.nueva_ventana)
 archivo.add_command(label="Abrir", command =
self.abrir_archivo)
 archivo.add_command(label="Guardar", command =
self.guardar_archivo)
 archivo.add_command(label="Guardar Como", command =
self.guardar_archivoComo)
 archivo.add_separator()
 archivo.add_command(label="Salir", command = self.master.quit)
 #pestaña de ver
 ver = Menu(menu, tearoff=0)
 ver.add_checkbutton(label="Barra de estado", variable =
self.info_estado, command = self.barra_de_estado)
 #pestaña de ayuda
 ayuda = Menu(menu, tearoff=0)
 ayuda.add_command(label="Acerca de", command=
self.acerca_de)
 ayuda.add_command(label="Manual de Usuario", command =
self.manualUsu)
 ayuda.add_command(label="Manual Tecnico", command =
self.manualTec)
 #pestaña de analizar
 analizar = Menu(menu, tearoff=0)
 analizar.add_command(label="Generar pagina web", command =
self.leerText) #Agregar esta funcion, command = self.analizar)
 #pestaña de tokens
 tokens = Menu(menu, tearoff=0)
 tokens.add_command(label="Ver tokens", command =
self.Ver_tokens) #Agregar esta funcion, command =
self.verTokens)
 #pestaña de errores
 errores = Menu(menu, tearoff=0)
 errores.add_command(label="Ver errores") #Agregar esta funcion,
command = self.verErrores
 #añadimos las pestañas al menú superior
 menu.add_cascade(label="Archivo", menu=archivo)
 menu.add_cascade(label="Analisis",menu=analizar)
 menu.add_cascade(label="Tokens",menu=tokens)
 menu.add_cascade(label="Errores", menu=errores)
```

```
menu.add_cascade(label="Ver", menu=ver)
menu.add_cascade(label="Ayuda", menu=ayuda)
#configuramos la entrada de texto
self.texto = Text(self.master, font= ('Arial', 12),
undo= True, background='black', foreground='Aqua',
insertbackground = "white") #colores letras, fondo, cursor
self.texto.grid(column=0, row=0, sticky='nsew')
self.texto.config(wrap='none')
ladox = Scrollbar(self.master, orient = 'horizontal', command=
self.texto.xview)
ladox.grid(column=0, row = 1, sticky='ew')
ladoy = Scrollbar(self.master, orient ='vertical', command =
self.texto.yview)
ladoy.grid(column = 1, row = 0, sticky='ns')
self.texto.configure(xscrollcommand = ladox.set, yscrollcommand
= ladoy.set)
self.barra_estado = Label(self.master, font = ('Segoe UI Symbol',
10))
#funcion para analizar el texto
def leerText(self):
lectura = self.texto.get('1.0', 'end')
autom = Analizar()
autom.analizar(lectura)
autom.imprimir_tokens()
autom.imprimirControles()
autom.imprimirID()
autom.imprimirIDPropiedades()
autom.imprimirPropiedades()
autom.imprimircaracPro()
autom.imprimirIDColocacion()
autom.imprimirColocacion()
autom.imprimircaracCol()"'
#autom.impresion()
#autom.traduccionControles()
#autom.traPropiedades()
#autom.imprimirControles1()
autom.generarHTML()
def manualUsu(self):
directorio = os.getcwd()
os.startfile(directorio+"\Manuales\ManualUsuario.pdf")
def manualTec(self):
directorio = os.getcwd()
os.startfile(directorio+"\Manuales\ManualTecnico.pdf")
#funcion para ver el numero de letras
def barra_de_estado(self):
if self.info_estado.get() == True:
n = len(self.texto.get('1.0','end'))
self.barra_estado.grid(column=0, row = 2, sticky='ew')
#self.barra_estado.config(text = f'Numero de letras: {n}' )
x = self.barra_estado.after(10, self.barra_de_estado)
if self.info_estado.get() == False:
self.barra_estado.after_cancel(x)
self.barra_estado.grid_forget()
(self.fila, self.col) = self.texto.index(INSERT).split('.')
self.colum = int(self.col) + 1
#print(f'Fila: {self.fila} Columna: {str(self.colum)}')
self.barra_estado.config(text = f'Fila: {self.fila} Columna:
{str(self.colum)}' )
#funcion para abrir un archivo ya listo
def abrir_archivo(self):
direccion = filedialog.askopenfilename(initialdir ='/',
title='Archivo', filetype=(('txt files', '*.txt*'),('All files', '*.*')))
if direccion != '':
archivo = open(direccion, 'r')
contenido = archivo.read()
self.texto.delete('1.0', 'end')
self.texto.insert('1.0', contenido)
#self.master.title(direccion)
self.ruta = direccion
#funcion para guardar un archivo como
def guardar_archivoComo(self):
try:
filename = filedialog.asksaveasfilename(defaultextension='.txt')
archivo = open(filename, 'w')
archivo.write(self.texto.get('1.0', 'end'))
```

```
archivo.close()
messagebox.showinfo('Guardar Archivo','Archivo guardado en: ' + str(filename) )
self.ruta = filename
except:
messagebox.showerror('Guardar Archivo', 'ERROR: Archivo no guardado')
#funcion para guardar una archivo
def guardar_archivo(self):
try:
archivo = open(self.ruta, 'w')
archivo.write(self.texto.get('1.0', 'end'))
archivo.close()
messagebox.showinfo('Guardar Archivo','Archivo guardado')
except:
messagebox.showerror('Guardar Archivo', 'ERROR: Archivo no guardado')
#funcion de documento nuevo (gurada y borra o no guarda y borra)
def nueva_ventana(self):
if self.texto.get !=" ":
valor = messagebox.askyesno('Proyecto2', '¿Desea guardar el archivo?',parent= self.master)
if valor == True:
self.guardar_archivoComo()
self.ruta = ''
self.texto.delete('1.0', 'end')
else:
self.texto.delete('1.0', 'end')
self.ruta = ''
else:
self.texto.delete('1.0', 'end')
self.ruta = ''
#funcion acerca de
def acerca_de(self):
vent_info = Toplevel(self.master)
vent_info.config( bg='white')
vent_info.title('Información')
vent_info.resizable(0,0)
#vent_info.iconbitmap('icono.ico')
vent_info.geometry('290x100+200+200')
Label(vent_info, bg='white',
text= 'Curso: Lab. Lenguajes Formales y de Programación \n Nombre: Mario Ernesto Marroquín Pérez \n Carné: 202110509').pack(expand=True)
def Ver_tokens(self):
vent_info = Toplevel(self.master)
vent_info.config( bg='white')
vent_info.title('Tabla de Tokens')
#vent_info.resizable(0,0)
vent_info.geometry('500x260+200+200')
def volver():
vent_info.destroy()
regresar =ttk.Button(vent_info,text="Regresar",command=volver)
self.tablaVerTokens = ttk.Treeview(vent_info, columns = ('#1', '#2', '#3', '#4'))
self.tablaVerTokens.column('#0', width = 100)
self.tablaVerTokens.column('#1', width = 100)
self.tablaVerTokens.column('#2', width = 100)
self.tablaVerTokens.column('#3', width = 100)
self.tablaVerTokens.column('#4', width = 100)
self.tablaVerTokens.heading('#0', text = 'Correlativo')
self.tablaVerTokens.heading('#1', text = 'Tipo de Token')
self.tablaVerTokens.heading('#2', text = 'Fila')
self.tablaVerTokens.heading('#3', text = 'Columna')
self.tablaVerTokens.heading('#4', text = 'Lexema')
self.tablaVerTokens.grid(column = 0, row = 0, sticky = 'nsew')
regresar.grid(column = 0, row = 1, sticky = 'nsew')
tokens = tablaTodosTokens
for e in reversed(tokens):
co = e["co"]
Tk = e["Tk"]
fl = e["fl"]
cl = e["cl"]
lx = e["lx"]
```

```
 self.tablaVerTokens.insert('', '0', text = (co), values = (Tk, fl, cl,
lx))

if __name__ == "__main__":

 ventana = Tk()

 app = Ventana(ventana)

 app.mainloop()
```