Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Departamento de Ciencias y Sistemas

“CÓDIGO FUENTE”

Audrie Annelisse del Cid Ochoa
Carné: 201801263
Guatemala, julio de 2012.

Índice

CÓDIGO FUENTE

Interfaz

```
from tkinter import *
from tkinter import ttk
from tkinter import Tk, Menu, messagebox, filedialog, ttk, Label, scrolledtext, INSERT, END,
Button, Scrollbar,BOTTOM, RIGHT, Y, Frame, Canvas, HORIZONTAL, VERTICAL,
simpledialog,X,Text
from AnalizadorL_JS import AnalizadorL_JS
from AnalizadorL_CSS import AnalizadorL_CSS
from colorama import *
from AnalizadorSintactico import Analizadorsintactico
from AnalizadorL_HTML import AnalizadorL_HTML

class ML_WEB(AnalizadorL_JS,AnalizadorL_CSS, AnalizadorL_HTML):
def __init__ (self, window):
self.root = window
self.root.title("ML WEB")
self.Entrada=""
self.extension="Audrie8a"

frame = Frame(root, bg="dark slate gray")
frame = Frame(root, bg="dark slate gray")

canvas = Canvas(frame, bg="dark slate gray")
scroll = Frame(canvas, bg="dark slate gray")
self.editor = scrolledtext.ScrolledText(scroll, undo = True, width = 50, height = 20, font =
("Arial", 15), background = 'dark slate gray', foreground = "dark slate gray")
scrollbar = Scrollbar(frame, orient=VERTICAL, command=canvas.yview)


MenuOpciones = Menu(root)
root.config(menu = MenuOpciones, width = 1000, height = 600)

archivoMenu = Menu(MenuOpciones, tearoff=0)

MenuOpciones.add_cascade(label = "Archivo", menu = archivoMenu)
archivoMenu.add_command(label = "Abrir", command = self.abrir)
archivoMenu.add_command(label = "Guardar", command= self.guardar)
archivoMenu.add_command(label = "Guardar Como",command= self.guardarComo)
MenuOpciones.add_command(label = "Analizar", command = self.analizar)
MenuOpciones.add_command(label = "Salir", command = self.salir)
```

```
scroll.bind("<Configure>",lambda e: canvas.configure(scrollregion=canvas.bbox("all")))

canvas.create_window((0, 0), window=scroll, anchor="nw")

canvas.configure(yscrollcommand=scrollbar.set, width = 1280, height = 800)

ttk.Label(scroll, text = "Editor", font = ("Arial", 20), background='dark slate gray', foreground
= "pale green").grid(column = 1, row = 0)

self.editor = scrolledtext.ScrolledText(scroll, undo = True, width = 50, height = 20, font =
("Arial", 15), background = 'pale green', foreground = "black")

self.editor.grid(column = 1, row = 1, pady = 25, padx = 25)

ttk.Label(scroll, text = "Consola", font = ("Arial", 20), background='dark slate gray',
foreground = "pale green").grid(column = 2, row = 0)

self.consola = scrolledtext.ScrolledText(scroll, undo = True, width = 50, height = 20, font =
("Arial", 15), background = 'pale green', foreground = "black")

self.consola.grid(column = 2, row = 1, pady = 10, padx = 10)


frame.grid(sticky='news')
canvas.grid(row=0,column=1)
scrollbar.grid(row=0, column=2, sticky='ns')

self.editor.focus()
self.consola.focus()

#END

#FUNCIONES-----------------------------------------------------------------------------------------------

def salir(self):
value = messagebox.askokcancel("Salir", "Está seguro que desea salir?")
if value :
root.destroy()
#END

def abrir(self):
global archivo
archivo = filedialog.askopenfilename(title = "Abrir", initialdir = "C:/")
entrada = open(archivo)
contenido = entrada.read()
strArchivo=str(archivo)
Direccion=strArchivo.split('.')
```

```
self.extension=Direccion[-1]
print(Direccion[-1])
self.editor.delete(1.0, END)
self.editor.insert(INSERT, contenido)
#print(Fore.BLUE+"Abierto")
entrada.close()
#END

def guardarComo(self):
global archivo
guardar = filedialog.asksaveasfilename(title = "Guardar Como", initialdir = "C:/")
fileguardar = open(guardar, "w+")
fileguardar.write(self.editor.get(1.0, END))
fileguardar.close()
archivo = guardar
#END

def guardar(self):
global archivo
if archivo == "":
self.guardarComo()
else:
guardarc = open(archivo, "w")
guardarc.write(self.editor.get(1.0, END))
guardarc.close()
#END

def obtenerTexto(self, Texto):
return Texto

def analizar(self):
try:
self.Entrada=self.editor.get("1.0",END)
AuxEntrada= self.Entrada.strip()
if(len(AuxEntrada) != 0):
if(self.extension=="Audrie8a"):
pregunta=messagebox.askquestion("Pregunta", "Desea utilizar el Analizador de JS?")
if(pregunta=='yes'):
Errores= app.funcMain(self.Entrada)
else:
pregunta2=messagebox.askquestion("Pregunta","Desea utilizar el Analizador de CSS?")
if(pregunta2=='yes'):
Errores =app.funcMainCSS(self.Entrada)
else:
pregunta3=messagebox.askquestion("Pregunta","Desea utilizar el Analizador de HTML?")
if(pregunta3=='yes'):
Errores=app.funcMainHTML(self.Entrada)
```

```
else:
pregunta4=messagebox.askquestion("Pregunta","Desea utilizar el Analizador Sintactico?")
if(pregunta4=='yes'):
Texto=self.Entrada.strip().split("\n")
Respuesta=""
for lexema in Texto:
clase=Analizadorsintactico(lexema)
Respuesta+=str(clase.Resultado)+"\n"
Resultados=Respuesta.split("\n")
contador=0
lista=[]
for lex in Texto:
lista.append([lex,Resultados[contador]])
contador+=1
clase.generarReporte(lista)
else:
messagebox.showinfo("Respuesta", "Lo sentimos, no contamos con más analizadores. :(")
else:
if(self.extension.lower()=='js'.lower()):
Errores= app.funcMain(self.Entrada)
self.extension="Audrie8a"
elif (self.extension.lower()=='css'.lower()):
Errores =app.funcMainCSS(self.Entrada)
self.extension="Audrie8a"
elif (self.extension.lower()=='html'.lower()):
Errores=app.funcMainHTML(self.Entrada)
self.extension="Audrie8a"
elif(self.extension.lower()=='rmt'.lower()):
Texto=self.Entrada.strip().split("\n")
Respuesta=""
for lexema in Texto:
clase=Analizadorsintactico(lexema)
Respuesta+=str(clase.Resultado)+"\n"
Resultados=Respuesta.split("\n")
contador=0
lista=[]
for lex in Texto:
lista.append([lex,Resultados[contador]])
contador+=1
clase.generarReporte(lista)
self.extension="Audrie8a"

self.consola.delete(1.0,END)
if (len(Errores)!=0):
self.consola.insert(INSERT, Errores)
messagebox.showinfo("Respuesta","El Análisis finalizado")
else:
```

```
messagebox.showinfo("Respuesta","No se ha ingresado ningún texto a Analizar")
except Exception as e:
print (e)
messagebox.showinfo("Respuesta","Ocurrió un error al Analizar el archivo")
#END


if __name__ == '__main__':
root= Tk()
app= ML_WEB(root)
root.mainloop()
```

Analizador Léxico JS

```
import re
from graphviz import Digraph
from Rutas import Rutas
from ReporteErrores import ReporteErrores
import metodos
from AFN import AFN
import os
import pydotplus
from sklearn import tree
from sklearn.tree import export_graphviz
from PIL import Image


signos=['\(', '\)' , '\{', '\}', '\;', '\,', '\.', '\:']
operadores=['\=', '\*', '\>','\+', '\-','\!', '\<'] #Math.pow, &&, ||
linea = 0
columna = 0
contador = 0
Recuperacion=""
ABC=['A','B','C','D','E','F','G','H']
Errores = []
palabrasReservadas = ['var','if','else','console.log', 'for','while','do','true','false','return',
'function', 'constructor', 'class','this','return', "math.pow"]

bComentario=False
bIdentificadores=False
bNumeros=False
bCadena= False
bCaracter=False
bSimbolo=False
bDecimal = False
class AnalizadorL_JS(ReporteErrores, Rutas):

def funcMain(self, Entrada):
global contador, Errores,Recuperacion
Salida=""
clase = AnalizadorL_JS()
contador=0
Errores=[]
tokens = clase.Analizador(Entrada+"#")
PalabrasReservadas(tokens)
for token in tokens:
print(token)
#Salida+=listToString(token)+"\n"
```

```
print("---------ERRORES:--------")
Salida+="ERRORES:"+"\n"
if(len(Errores)!=0):
clase.GenerarReporte(Errores,"Reporte Errores Analizador JS")
for err in Errores:
print (err)
Salida+=listToString(err)+"\n"
if(len(tokens)!=0 and len(Recuperacion)!=0):
clase.CrearRuta(tokens, Recuperacion,"js")
Recuperacion=""
reporteGrafico(clase.getRuta())
return Salida
#END

def Analizador(self, Entrada):
global linea, columna, contador,
Errores,Recuperacion,bCadena,bCaracter,bComentario,bDecimal,bIdentificadores,bNumeros,b
Simbolo
linea = 1
columna = 1
listaTokens = []

while contador < len(Entrada)-1:
if Entrada[contador]=="/": #COMENTARIOS
aux=""
if Entrada[contador+1]=="*" and (contador+1)<len(Entrada): #multilínea
contador+=2
columna+=2
aux="/*"

while(Entrada[contador]!="\*" and Entrada[contador+1]!="/" and
(contador+1)<len(Entrada)-1):
aux+= Entrada[contador]
if(Entrada[contador]=="\n"):
linea+=1
#
contador+=1
columna+=1
if(contador+2)<len(Entrada):
contador+=2
columna+=2
aux+="*/"
listaTokens.append([linea,columna,'Comentario',aux])
Recuperacion+=aux
bComentario=True
aux=""
```

```
elif Entrada[contador+1]=="/" and (contador+1)<len(Entrada): #unilinea
aux+="/"
contador+=1
columna+=1
while(Entrada[contador]!="\n" and (contador)<len(Entrada)):
aux+= Entrada[contador]
contador+=1
columna+=1

listaTokens.append([linea,columna,'Comentario',aux])
bComentario=True
Recuperacion+=aux
aux=""
else: #signo aritmético
listaTokens.append([linea,columna,'Operador',Entrada[contador]])
Recuperacion+=str(Entrada[contador])
contador+=1
bSimbolo=True

elif re.search(r"[A-Za-z]", Entrada[contador]): #IDENTIFICADOR
listaTokens.append(EstadoIdentificador(linea, columna, Entrada, Entrada[contador]))
bIdentificadores=True
elif re.search(r"[\n]", Entrada[contador]): #SALTO DE LINEA
Recuperacion+=str(Entrada[contador])
contador += 1
linea += 1
columna = 1
elif re.search(r"[ \t]", Entrada[contador]):#ESPACIOS Y TABULACIONES
Recuperacion+=str(Entrada[contador])
contador += 1
columna += 1
elif re.search(r"[0-9]", Entrada[contador]): #NUMERO
listaTokens.append(EstadoNumero(linea, columna, Entrada, Entrada[contador]))
bSimbolo=True
elif Entrada[contador] == '"':
auxCadena=Entrada[contador]
contador+=1
columna +=1
while Entrada[contador] != '"'and (contador)<len(Entrada)-1:
auxCadena+=Entrada[contador]
contador+=1
columna+=1
listaTokens.append([linea, columna, "Cadena", auxCadena+Entrada[contador]])
Recuperacion+=auxCadena+Entrada[contador]
bCadena=True
contador +=1
columna+=1
```

```
elif Entrada[contador]=="'":
auxCaracter=Entrada[contador]
contador+=1
columna+=1
while(Entrada[contador]!="'"and (contador)<len(Entrada)-1):
auxCaracter+=Entrada[contador]
contador+=1
columna+=1
auxCaracter+=Entrada[contador]
listaTokens.append([linea, columna, 'Caracter', auxCaracter])
Recuperacion+=auxCaracter
bCaracter=True
contador+=1
columna+=1
else:
isSign = False
isOper= False
for sign in signos:
palabra = r"^" + sign + "$"
if re.match(palabra,Entrada[contador], re.IGNORECASE):
listaTokens.append([linea, columna, 'Signo', Entrada[contador]])
Recuperacion+=str(Entrada[contador])
contador += 1
columna += 1
isSign = True
break

if(isSign==False):
for operador in operadores:
oper=r"^" + operador + "$"
if re.match(oper, Entrada[contador], re.IGNORECASE):
listaTokens.append([linea, columna, 'Operador', Entrada[contador]])
Recuperacion+=str(Entrada[contador])
bSimbolo=True
contador += 1
columna += 1
isOper = True
break
if(Entrada[contador]=='&' and Entrada[contador+1]=='&' and (contador+1)<len(Entrada)):
concatenado=Entrada[contador]+Entrada[contador+1]
listaTokens.append([linea,columna,'Operador',concatenado])
Recuperacion+=concatenado
bSimbolo=True
contador+=2
columna +=2
isOper=True
if(Entrada[contador]=='|' and Entrada[contador+1]=='|' and (contador+1)<len(Entrada)):
```

```
concatenado2=Entrada[contador]+Entrada[contador+1]
listaTokens.append([linea,columna,'Operador', concatenado2 ])
Recuperacion+=concatenado2
bSimbolo=True
contador+=2
columna +=2
isOper=True
if (isSign==False and isOper==False):
Errores.append([linea, columna, Entrada[contador]])
contador += 1
columna += 1
#END

return listaTokens
#END

def PalabrasReservadas(lstTokens):
for token in lstTokens:
if token[2] == 'identificador':
for reservada in palabrasReservadas:
palabra = r"^" + reservada + "$"
if re.match(palabra, token[3], re.IGNORECASE):
token[2] = 'reservada'
break
#END

def EstadoIdentificador (linea, column, text, Caracter):
global contador, columna,Recuperacion
contador += 1
columna += 1
if contador < len(text):
if re.search(r"[a-zA-Z_0-9]", text[contador]):#IDENTIFICADOR
return EstadoIdentificador(linea, column, text, Caracter + text[contador])
else:
if(Caracter.lower()== 'math'.lower() and text[contador].lower()==".".lower()):
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
else:
Recuperacion+=Caracter
return [linea, column, 'identificador', Caracter]
#agregar automata de identificador en el arbol, con el valor
else:
Recuperacion+=Caracter
return [linea, column, 'identificador', Caracter]
#END

def EstadoNumero(line, column, text, numero):
global contador, columna,Recuperacion
```

```
contador += 1
columna += 1
if contador < len(text):
if re.search(r"[0-9]", text[contador]):#ENTERO
return EstadoNumero(line, column, text, numero + text[contador])
elif re.search(r"\.", text[contador]):#DECIMAL
return EstadoDecimal(line, column, text, numero + text[contador])
else:
Recuperacion+=str(numero)
return [line, column, 'Número', numero]
#agregar automata de numero en el arbol, con el valor
else:
Recuperacion+=str(numero)
return [line, column, 'Número', numero]
#END

def EstadoDecimal(line, column, text, decimal):
global contador, columna, Recuperacion
contador += 1
columna += 1
if contador < len(text):
if re.search(r"[0-9]", text[contador]):#DECIMAL
return EstadoDecimal(line, column, text, decimal + text[contador])
else:
Recuperacion+=str(decimal)
return [line, column, 'decimal', decimal]
#agregar automata de decimal en el arbol, con el valor
else:
Recuperacion+=str(decimal)
return [line, column, 'decimal', decimal]
#END

def listToString(token):
stri=" "
counter=0
while counter<len(token):
if(counter==0):
stri+="["+str(token[counter])+", "
counter+=1
elif (counter==1):
stri+=str(token[counter])+", "
counter+=1
elif (counter==2):
stri+=str(token[counter])+"] "
counter+=1
```

```
return stri
#END

def subGrafosRG():
global bCadena,bCaracter,bComentario,bDecimal,bIdentificadores,bNumeros,bSimbolo
if(bCadena):
metodos.addER("...\"*C\"")
if (bCaracter):
metodos.addER("...'*C'")
if (bComentario):
metodos.addER(".|..//*C.../A*.*C*E.A/")
if (bDecimal):
metodos.addER("...*NP*N") #DECIMAL ...+NP+N
if (bIdentificadores):
metodos.addER("..*L*||LN_") #IDENTIFICADORES ..+L*||LN_
if (bNumeros):
metodos.addER(".*N") #NUMERO .+N
if(bSimbolo):
metodos.addER(".*S") #Simbolo .+S

#END

def reporteGrafico(Ruta):
global ABC
claseAFN=AFN()
dot = Digraph(name='parent')
dot.attr('node', shape='circle')
subGrafosRG()
listaER=metodos.getListER()
contar=len(listaER)
dot.node("A0","A0")
conteo =0
while contar!=0:
dot.edge("A0","S0"+ABC[conteo])
contar-=1
conteo+=1

contar=0
TextoG=dot.source.replace('}'," ")
TextoSG=""
print (TextoG)
for sg in listaER:
SubGrafo=claseAFN.AFN(sg, ABC[contar])
print("°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°")
print(SubGrafo)
print ("°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°")
```

```
TextoSG+="\n"+SubGrafo
contar+=1
print("-----------------------------------")
Texto=TextoG+"\n"+TextoSG+"\n}"
print(Texto)
arch = open(Ruta+"/ArbolJS.dot", "w")
arch.write(Texto)
arch.close()
os.environ["PATH"]+=os.pathsep+Ruta+'/ArbolJS.dot'
os.system('dot -Tpng '+Ruta+'/ArbolJS.dot -o '+Ruta+'/ArbolJS.png')
f= Image.open(Ruta+"ArbolJS.png")
f.show()
#END
```

```
EntradaTexto= open('entrada.olc1')
contenido = EntradaTexto.read()
```

```
if __name__ == "__main__":
clase = AnalizadorL_JS()
clase.funcMain(contenido)
```

Analizador Léxico CSS

```
import re
from Rutas import Rutas
from ReporteErrores import ReporteErrores
signos=['%','#','\*', '\-' , '\{', '\}', '\;', '\,', '\.', '\:','\(','\)']
linea = 0
columna = 0
contador = 0
Recuperacion=""

Errores = []
Bitacora=[] #[Lexema,Estado, Token, Aceptacion]
palabrasReservadas =
['mm','pt','pc','cm','in','vw','vh','em','px','position','bottom','color','display','top','float','Opacity'
, 'width','right','clear','height','left','text-align','border', 'border-style','font-weight', 'font-
style','font-family','font-size','padding-left','padding-bottom','padding-top','padding-right','line-
height','min-width','min-height','margin','margin-right','margin-bottom','margin-top','margin-
left','max-height','max-width','background-image','background','background-image', ]#text-
align


class AnalizadorL_CSS(ReporteErrores,Rutas):

def funcMainCSS(self, Entrada):
global contador, Errores, Recuperacion, Bitacora
Salida=""
clase = AnalizadorL_CSS()
contador=0
Errores=[]
Bitacora=[]
tokens = clase.Analizador(Entrada+"#")
PalabrasReservadas(tokens)
for token in tokens:
print(token)
#Salida+=listToString(token)+"\n"
print("---------ERRORES:--------")
Salida+="--------------BITÁCORA:--------------"+"\n"
Salida+="\n[Lexema, Estado, Token, Aceptación]\n"
if(len(Errores)!=0):
clase.GenerarReporte(Errores,"Reporte Analizador de CSS")
for err in Errores:
print (err)
#Salida+=listToString(err)+"\n"
if(len(tokens)!=0 and len(Recuperacion)!=0):
clase.CrearRuta(tokens, Recuperacion,"css")
```

```
Recuperacion=""
if(len(Bitacora)!=0):
for tok in Bitacora:
Salida+=printBitacora(tok)+"\n"
return Salida
#END

def Analizador(self, Entrada):
global linea, columna, contador, Errores, Recuperacion,Bitacora
linea = 1
columna = 1
listaTokens = []

while contador < len(Entrada)-1:
if Entrada[contador]=="/": #COMENTARIOS
aux=""
if Entrada[contador+1]=="*" and (contador+1)<len(Entrada): #multilínea
contador+=2
columna+=2
aux="/*"
Bitacora.append([aux,"S0","Comentario","False"])
while(ord(Entrada[contador])!=ord("*") and ord(Entrada[contador+1])!=92 and
(contador+1)<len(Entrada)-1):
aux+= Entrada[contador]
#print(Entrada[contador])
if(Entrada[contador]=="\n"):
linea+=1
#_____
contador+=1
columna+=1
Bitacora.append([aux,"S1","Comentario","False"])
if(contador+2)<len(Entrada):
#print(Entrada[contador])
#print(Entrada[contador+1])
contador+=2
columna+=2
aux+="*/"
Bitacora.append(["*/","S2","Comentario","True"])
listaTokens.append([linea,columna,'Comentario',aux])
Recuperacion+=aux
aux=""
else:
Errores.append([linea, columna, Entrada[contador]])
Bitacora.append([Entrada[contador],"--","ERROR","False"])
contador += 1
columna += 1
```

```
elif re.search(r"[A-Za-z]", Entrada[contador]): #IDENTIFICADOR
Bitacora.append([Entrada[contador],"S0","Identificador","True"])
listaTokens.append(EstadoIdentificador(linea, columna, Entrada, Entrada[contador]))
elif re.search(r"[\n]", Entrada[contador]): #SALTO DE LINEA
Recuperacion+=str(Entrada[contador])
contador += 1
linea += 1
columna = 1
elif re.search(r"[ \t]", Entrada[contador]):#ESPACIOS Y TABULACIONES
Recuperacion+=str(Entrada[contador])
contador += 1
columna += 1
elif re.search(r"[0-9]", Entrada[contador]): #NUMERO
listaTokens.append(EstadoNumero(linea, columna, Entrada, Entrada[contador]))
elif Entrada[contador] == '"':
auxCadena=Entrada[contador]
Bitacora.append([Entrada[contador],"S0","Cadena","False"])
contador+=1
columna +=1
while Entrada[contador] != '"'and (contador)<len(Entrada)-1:
auxCadena+=Entrada[contador]
contador+=1
columna+=1
listaTokens.append([linea, columna, "Cadena", auxCadena+Entrada[contador]])
Bitacora.append([auxCadena,"S1","Cadena","False"])
Bitacora.append([Entrada[contador],"S2","Cadena","True"])
Recuperacion+=auxCadena+Entrada[contador]
contador +=1
columna+=1
else:
isSign = False
for sign in signos:
palabra = r"^" + sign + "$"
if re.match(palabra,Entrada[contador], re.IGNORECASE):
listaTokens.append([linea, columna, 'Signo', Entrada[contador]])
Bitacora.append([Entrada[contador],"S0","Signo","True"])
Recuperacion+=str(Entrada[contador])
contador += 1
columna += 1
isSign = True
break
if (isSign==False ):
Errores.append([linea, columna, Entrada[contador]])
Bitacora.append([Entrada[contador],"--","ERROR","False"])
contador += 1
columna += 1
#END
```

```
return listaTokens
#END

def PalabrasReservadas(lstTokens):
for token in lstTokens:
if token[2] == 'identificador':
for reservada in palabrasReservadas:
palabra = r"^" + reservada + "$"
if re.match(palabra, token[3], re.IGNORECASE):
token[2] = 'reservada'
break


#END

def EstadoIdentificador (linea, column, text, Caracter):
global contador, columna, Recuperacion, Bitacora
contador += 1
columna += 1
if contador < len(text):
if re.search(r"[a-zA-Z_0-9]", text[contador]):#IDENTIFICADOR
Bitacora.append([Caracter+text[contador],"S0","Identificador","True"])
return EstadoIdentificador(linea, column, text, Caracter + text[contador])
else:
if(Caracter.lower()== 'border'.lower() and text[contador].lower()=="-".lower()):
Bitacora.append([Caracter+text[contador],"S1","PR","False"])
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
elif(Caracter.lower()== 'font'.lower() and text[contador].lower()=="-".lower()):
Bitacora.append([Caracter+text[contador],"S1","PR","False"])
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
elif(Caracter.lower()== 'padding'.lower() and text[contador].lower()=="-".lower()):
Bitacora.append([Caracter+text[contador],"S1","PR","False"])
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
elif(Caracter.lower()== 'line'.lower() and text[contador].lower()=="-".lower()):
Bitacora.append([Caracter+text[contador],"S1","PR","False"])
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
elif(Caracter.lower()== 'min'.lower() and text[contador].lower()=="-".lower()):
Bitacora.append([Caracter+text[contador],"S1","PR","False"])
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
elif(Caracter.lower()== 'margin'.lower() and text[contador].lower()=="-".lower()):
Bitacora.append([Caracter+text[contador],"S1","PR","False"])
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
elif (Caracter.lower()== 'max'.lower() and text[contador].lower()=="-".lower()):
Bitacora.append([Caracter+text[contador],"S1","PR","False"])
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
elif(Caracter.lower()== 'background'.lower() and text[contador].lower()=="-".lower()):
```

```
Bitacora.append([Caracter+text[contador],"S1","PR","False"])
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
elif(Caracter.lower()== 'text'.lower() and text[contador].lower()=="-".lower()):
Bitacora.append([Caracter+text[contador],"S1","PR","False"])
return EstadoIdentificador(linea, columna, text, Caracter + text[contador])
else:
Recuperacion+=Caracter
Bitacora.append([Caracter,"S0","Identificador","True"])
return [linea, column, 'identificador', Caracter]
else:
Recuperacion+=Caracter
Bitacora.append([Caracter+text[contador],"S0","Identificador","True"])
return [linea, column, 'identificador', Caracter]
#END

def EstadoNumero(line, column, text, numero):
global contador, columna, Recuperacion
contador += 1
columna += 1
if contador < len(text):
if re.search(r"[0-9]", text[contador]):#ENTERO
Bitacora.append([numero+text[contador],"S0","Numero","True"])
return EstadoNumero(line, column, text, numero + text[contador])
elif re.search(r"\.", text[contador]):#DECIMAL
Bitacora.append([numero+text[contador],"S1","Numero","True"])
return EstadoDecimal(line, column, text, numero + text[contador])
else:
Recuperacion+=str(numero)
Bitacora.append([numero,"S0","Numero","True"])
return [line, column, 'Número', numero]
#agregar automata de numero en el arbol, con el valor
else:
Recuperacion+=str(numero)
Bitacora.append([numero,"S0","Numero","True"])
return [line, column, 'Número', numero]
#END

def EstadoDecimal(line, column, text, decimal):
global contador, columna, Recuperacion
contador += 1
columna += 1
if contador < len(text):
if re.search(r"[0-9]", text[contador]):#DECIMAL
Bitacora.append([decimal+text[contador],"S0","Numero","False"])
return EstadoDecimal(line, column, text, decimal + text[contador])
else:
Recuperacion+=str(decimal)
```

```
Bitacora.append([decimal,"S0","Numero","True"])
return [line, column, 'decimal', decimal]
else:
Recuperacion+=str(decimal)
Bitacora.append([decimal,"S0","Numero","True"])
return [line, column, 'decimal', decimal]
#END

def listToString(token):
stri=" "
counter=0
while counter<len(token):
if(counter==0):
stri+="["+str(token[counter])+", "
counter+=1
elif (counter==1):
stri+=str(token[counter])+", "
counter+=1
elif (counter==2):
stri+=str(token[counter])+"] "
counter+=1

return stri
#END

def printBitacora(token):
stri=" "
counter=0
while counter<len(token):
if(counter==0):
stri+="["+str(token[counter])+"-->"
counter+=1
elif (counter==1):
stri+=str(token[counter])+"--> "
counter+=1
elif (counter==2):
stri+=str(token[counter])+"--> "
counter+=1
elif (counter==3):
stri+=str(token[counter])+"] "
counter+=1

return stri
#END
```

```
EntradaTexto= open('entrada.olc1')
contenido = EntradaTexto.read()


if __name__ == "__main__":
clase = AnalizadorL_CSS()
clase.funcMainCSS(contenido)
```

Analizador Léxico HTML

```
import re
from Rutas import Rutas
from ReporteErrores import ReporteErrores
signos=['=']
linea = 0
columna = 0
contador = 0
Recuperacion=""
bPath1=False
bPath2=False

Errores = []
palabrasReservadas =
['html','li','head','title','body','h1','h2','h3','h4','h5','h6','p','br','img','href','a','o','u','style','table'
,'th','tr','td','caption','colgroup','col','thead','tbody','tfoot']


class AnalizadorL_HTML(ReporteErrores,Rutas):

def funcMainHTML(self, Entrada):
global contador, Errores, Recuperacion
Salida=""
clase = AnalizadorL_HTML()
contador=0
Errores=[]
tokens = clase.Analizador(Entrada+"#")
PalabrasReservadas(tokens)
for token in tokens:
print(token)
#Salida+=listToString(token)+"\n"
print("---------ERRORES:--------")
Salida+="--------------ERRORES:--------------"+"\n"
if(len(Errores)!=0):
clase.GenerarReporte(Errores,"Reporte Analizador de HTML")
for err in Errores:
print (err)
#Salida+=listToString(err)+"\n"
if(len(tokens)!=0 and len(Recuperacion)!=0):
clase.CrearRuta(tokens, Recuperacion,"html")
Recuperacion=""
clase.CrearRuta(tokens, Recuperacion,"html")
return Salida
#END
```

```
def Analizador(self, Entrada):
global linea, columna, contador, Errores, Recuperacion,bPath1, bPath2
linea = 1
columna = 1
listaTokens = []

while contador < len(Entrada)-1:
if(Entrada[contador]=='<'):
contador+=1
if bPath1==False:
if(Entrada[contador]=='!'and Entrada[contador+1]=='-'and Entrada[contador+2]=='-'):
auxEntrada=Entrada[contador]+Entrada[contador+1]
contador+=3
while Entrada[contador]!='-'and Entrada[contador+1]!='-'and Entrada[contador+1]!='>':
auxEntrada+=Entrada[contador]
contador+=1
auxEntrada+=Entrada[contador]+Entrada[contador+1]+Entrada[contador+2]
contador+=5
listaTokens.append([linea,columna,"Comentario",auxEntrada])
#listaTokens.append(EstadoComentario(linea,columna, Entrada,Entrada[contador]))
bPath1=True
if bPath2==False:
if(Entrada[contador]=='<' and Entrada[contador+1]=='!'and Entrada[contador+2]=='-'and
Entrada[contador+3]=='-'):
auxEntrada2=Entrada[contador]+Entrada[contador+1]+Entrada[contador+2]
contador+=4
while Entrada[contador]!='-'and Entrada[contador+1]!='-'and Entrada[contador+1]!='>':
auxEntrada2+=Entrada[contador]
contador+=1
auxEntrada2+=Entrada[contador]+Entrada[contador+1]+Entrada[contador+2]
contador+=5
listaTokens.append([linea,columna,"Comentario",auxEntrada2])
#listaTokens.append(EstadoComentario(linea,columna, Entrada,Entrada[contador]))
bPath2=True
elif re.search(r"[A-Za-z]", Entrada[contador]): #IDENTIFICADOR
listaTokens.append(EstadoIdentificador(linea, columna, Entrada, Entrada[contador]))
elif Entrada[contador] == '"':
auxCadena=Entrada[contador]
contador+=1
columna +=1
while Entrada[contador] != '"'and (contador)<len(Entrada)-1:
auxCadena+=Entrada[contador]
contador+=1
columna+=1
listaTokens.append([linea, columna, "Cadena", auxCadena+Entrada[contador]])
Recuperacion+=auxCadena+Entrada[contador]
contador +=1
```

```
columna+=1
elif Entrada[contador]=="'":
auxCaracter=Entrada[contador]
contador+=1
columna+=1
while(Entrada[contador]!="'"and (contador)<len(Entrada)-1):
auxCaracter+=Entrada[contador]
contador+=1
columna+=1
auxCaracter+=Entrada[contador]
listaTokens.append([linea, columna, 'Caracter', auxCaracter])
Recuperacion+=auxCaracter
contador+=1
columna+=1
elif Entrada[contador]=='=':
listaTokens.append([linea, columna, "Signo", Entrada[contador]])
Recuperacion+=Entrada[contador]
contador +=1
columna+=1
elif Entrada[contador]=='/':
listaTokens.append([linea, columna, "Signo", Entrada[contador]])
Recuperacion+=Entrada[contador]
contador +=1
columna+=1
elif Entrada[contador]=='>':
listaTokens.append([linea, columna, "Signo", Entrada[contador]])
Recuperacion+=Entrada[contador]
contador +=1
columna+=1

elif Entrada[contador]=='/':
listaTokens.append([linea, columna, "Signo", Entrada[contador]])
Recuperacion+=Entrada[contador]
contador +=1
columna+=1
elif Entrada[contador]=='>':
listaTokens.append([linea, columna, "Signo", Entrada[contador]])
Recuperacion+=Entrada[contador]
contador +=1
columna+=1
else:
while(Entrada[contador]!='<' and contador<len(Entrada)-1):
if re.search(r"[\n]", Entrada[contador]): #SALTO DE LINEA
Recuperacion+=str(Entrada[contador])
contador += 1
linea += 1
columna = 1
```

```
elif re.search(r"[ \t]", Entrada[contador]):#ESPACIOS Y TABULACIONES
Recuperacion+=str(Entrada[contador])
contador += 1
columna += 1
elif re.search(r"[0-9]", Entrada[contador]): #NUMERO
listaTokens.append(EstadoNumeroTXT(linea, columna, Entrada, Entrada[contador]))
elif re.search(r"[A-Za-z]", Entrada[contador]): #IDENTIFICADOR
listaTokens.append(EstadoIdentificadorTXT(linea, columna, Entrada, Entrada[contador]))
else:
if Entrada[contador]!='<':
listaTokens.append([linea, columna,'Texto',Entrada[contador]])
contador += 1
columna += 1
return listaTokens

#END


def PalabrasReservadas(lstTokens):
for token in lstTokens:
if token[2] == 'identificador':
for reservada in palabrasReservadas:
palabra = r"^" + reservada + "$"
if re.match(palabra, token[3], re.IGNORECASE):
token[2] = 'reservada'
break
#END

def EstadoIdentificadorTXT (linea, column, text, Caracter):
global contador, columna, Recuperacion
contador += 1
columna += 1
if contador < len(text):
if re.search(r"[a-zA-Z_0-9]", text[contador]):#IDENTIFICADOR
return EstadoIdentificadorTXT(linea, column, text, Caracter + text[contador])
else:
Recuperacion+=Caracter
return [linea, column, 'Texto', Caracter]
else:
Recuperacion+=Caracter
return [linea, column, 'Texto', Caracter]
#END

def EstadoIdentificador (linea, column, text, Caracter):
global contador, columna, Recuperacion
contador += 1
columna += 1
```

```
if contador < len(text):
if re.search(r"[a-zA-Z_0-9]", text[contador]):#IDENTIFICADOR
return EstadoIdentificador(linea, column, text, Caracter + text[contador])
else:
Recuperacion+=Caracter
return [linea, column, 'identificador', Caracter]
else:
Recuperacion+=Caracter
return [linea, column, 'identificador', Caracter]
#END

def EstadoNumeroTXT(line, column, text, numero):
global contador, columna, Recuperacion
contador += 1
columna += 1
if contador < len(text):
if re.search(r"[0-9]", text[contador]):#ENTERO
return EstadoNumeroTXT(line, column, text, numero + text[contador])
elif re.search(r"\.", text[contador]):#DECIMAL
return EstadoDecimalTXT(line, column, text, numero + text[contador])
else:
Recuperacion+=str(numero)
return [line, column, 'Texto', numero]
#agregar automata de numero en el arbol, con el valor
else:
Recuperacion+=str(numero)
return [line, column, 'Texto', numero]
#END

def EstadoDecimalTXT(line, column, text, decimal):
global contador, columna, Recuperacion
contador += 1
columna += 1
if contador < len(text):
if re.search(r"[0-9]", text[contador]):#DECIMAL
return EstadoDecimalTXT(line, column, text, decimal + text[contador])
else:
Recuperacion+=str(decimal)
return [line, column, 'decimal', decimal]
else:
Recuperacion+=str(decimal)
return [line, column, 'decimal', decimal]
#END

def listToString(token):
stri=" "
counter=0
```

```
while counter<len(token):
if(counter==0):
stri+="["+str(token[counter])+", "
counter+=1
elif (counter==1):
stri+=str(token[counter])+", "
counter+=1
elif (counter==2):
stri+=str(token[counter])+"] "
counter+=1

return stri
#END
def EstadoComentario(line, column, text, Caracter):
global contador, columna, Recuperacion
if contador < len(text):
if re.search(r"(\<\!\-\-(\s*|.*?)*\-\-\!\>)",Caracter):
Recuperacion+= Caracter
return [line, column, 'Comentario', Caracter]
else:
contador+=1
column+=1
if re.search(r"[\n]", text[contador]): #SALTO DE LINEA
line += 1
Caracter+=text[contador]
EstadoComentario(line,column,text,Caracter)


EntradaTexto= open('entrada.olc1')
contenido = EntradaTexto.read()



if __name__ == "__main__":
clase = AnalizadorL_HTML()
clase.funcMainHTML(contenido)
```

Analizador Sintáctico

```
import re
import webbrowser

class Analizadorsintactico:


def __init__(self, Entrada):
self.Resultado=True
self.contador=0
self.Signos=['\+','\-',' ','\*','/','\(','\)']#Num e identificador
self.Estados=['E','T','G','R','F']

Entrada= Entrada+"#"
self.pila=[]
self.pila.append("#")
self.P(Entrada)
self.pila.clear()

#END

def P(self,Entrada):
self.pila.append('E')
self.q(Entrada)
#END

def q(self, Entrada):
self.contador=0
auxPila=[]
while(len(self.pila)!=0 or auxPila!="#"):
auxPila=self.pila.pop()
CondicionE=self.pruebaEstados(auxPila, Entrada)
if(CondicionE==False):
CondicionEN=self.pruebaEntrada(auxPila,Entrada)
if CondicionEN==False:
self.Resultado=False
break
print(self.imprimirPila())
#END

def imprimirPila(self):
texto=""
for item in reversed(self.pila):
texto+=item
```

```
return texto

def pruebaEstados(self,auxPila, Entrada):
condicionE=False
for state in self.Estados:
palabra = state
if re.match(palabra,auxPila, re.IGNORECASE):
condicionE=True
if(palabra=='E'):
self.pila.append('G')
self.pila.append('T')
elif(palabra=='G'):
if(Entrada[self.contador]=='+'):
self.pila.append('G')
self.pila.append('T')
self.pila.append('+')
elif(Entrada[self.contador]=='-'):
self.pila.append('G')
self.pila.append('T')
self.pila.append('-')
else:
print(" ")
elif(palabra=='T'):
self.pila.append('R')
self.pila.append('F')
elif(palabra=='R'):
if(Entrada[self.contador]=='*'):
self.pila.append('R')
self.pila.append('F')
self.pila.append('*')
elif(Entrada[self.contador]=='/'):
self.pila.append('R')
self.pila.append('F')
self.pila.append('/')
else:
print(" ")
elif(palabra=='F'):
if re.search(r"[A-Za-z]", Entrada[self.contador]): #IDENTIFICADOR
self.pila.append('ID')
#self.EstadoIdentificador(Entrada,Entrada[self.contador],self.contador)
elif (Entrada[self.contador]=='('):
self.pila.append(')')
self.pila.append('E')
self.pila.append('(')
elif re.search(r"[0-9]", Entrada[self.contador]): #NUMERO
self.pila.append('NUM')
return condicionE
```

```
#END

def pruebaEntrada(self, auxPila,Entrada):
condicionEn=False
for indx in self.Signos:
pedazo=str(indx)

if re.match(pedazo,auxPila,re.IGNORECASE):
condicionEn=True
self.contador+=1
elif auxPila=='ID':
if re.search(r"[A-Za-z]", Entrada[self.contador]): #IDENTIFICADOR
self.EstadoIdentificador(Entrada,Entrada[self.contador])
condicionEn=True
else:
self.Resultado=False
return
elif auxPila=='NUM':
if re.search(r"[0-9]", Entrada[self.contador]): #NUMERO
self.EstadoNumero(Entrada, Entrada[self.contador])
condicionEn=True
else:
if auxPila=='#':
return
return condicionEn
#END

def EstadoNumero(self,text, numero):
self.contador += 1
if self.contador < len(text):
if re.search(r"[0-9]", text[self.contador]):#ENTERO
return self.EstadoNumero(text, numero + text[self.contador])
elif re.search(r"\.", text[self.contador]):#DECIMAL
return self.EstadoDecimal(text, numero + text[self.contador])
else:
return numero
#agregar automata de numero en el arbol, con el valor
else:
return numero
#END

def EstadoDecimal(self, text, decimal):

self.contador += 1

if self.contador < len(text):
if re.search(r"[0-9]", text[self.contador]):#DECIMAL
```

```
return self.EstadoDecimal(text, decimal + text[self.contador])
else:
return decimal
#agregar automata de decimal en el arbol, con el valor
else:
return decimal
#END


def EstadoIdentificador (self,text,Caracter):
self.contador += 1
if self.contador < len(text):
if re.search(r"[a-zA-Z_0-9]", text[self.contador]):#IDENTIFICADOR
return self.EstadoIdentificador(text, Caracter + text[self.contador])
else:
return Caracter
#agregar automata de identificador en el arbol, con el valor
else:

return Caracter

def getRespuesta(self):
Respuesta = str(self.Resultado)
return Respuesta

def generarReporte(self,Lista):
f=open('ReporteErrores.html','w')
self.Texto="<h1>"+"Reporte Analizador Sintactico"+"</h1>\n"
self.Texto+="<table border=\"1\">"
self.Texto+=self.Tabla(Lista)

f.write(str(self.Texto))
f.close()

webbrowser.open_new_tab('ReporteErrores.html')
#END

def Tabla(self,Lista):
self.Texto+="<tr>\n"
self.Texto+="<th>Linea</th>\n"
self.Texto+="<th>Expresion</th>\n"
self.Texto+="<th>Evaluacion</th>\n"
self.Texto+="</tr>\n"

contador=0
for token in Lista:
self.Texto+="<tr>\n"
```

```
self.Texto+="<td>"+str(contador)+"</td>\n"
self.Texto+="<td>"+str(token[0])+"</td>\n"
self.Texto+="<td>"+str(token[1])+"</td>\n"
self.Texto+="</tr>\n"
contador+=1

self.Texto+="</table>"

return self.Texto
#END
#END

if __name__ == "__main__":
clase = Analizadorsintactico("4+5-7+(3+x)")#"((4-6*(1/8)/2)+(6-9*(2))-(5)*(3*x)/(var1))")
print(clase.Resultado)
```

Método del Árbol- Reporte JS

```
from pilaArbol import pilaArbol
from tablaTran import tablaTran
import metodos
import json
from nodo import nodo

class AFN:
def AFN(self, ER, L):
ER=ER+"#"

ConstruirA= pilaArbol(ER)

Raiz= ConstruirA.getRaiz()


Raiz.getNodo() #Se crea la información del nodo (Anulabilidad, primeros y últimos)
Raiz.siguientes() #Se crea la tabla siguientes con los nodos obtenidos de la ER

Transiciones= tablaTran(Raiz) #Se genera los datos de la tabla de transiciones

Subgrafo = Transiciones.grafo(L)
Subgrafo=Subgrafo.replace('digraph','subgraph')

ConstruirA.limpiarPila()
metodos.limpiarHojas()
metodos.limpiarTabla()
return Subgrafo #Se genera la parte del dot para la ER ingresada

if __name__ == "__main__":
ER = "....ab*b*|ab"

reporte = AFN()
reporte.AFN(ER,"|")
```

Clases ayuda al método:

Nodo

```
from tipo import tipo
import metodos


class nodo:

def __init__(self, lexema, tipo, numero, izq, der):
self.primeros = []
self.ultimos = []
self.anulable = True

self.lexema = lexema
self.tipo = tipo
self.numero = numero

self.aceptacion = False
if lexema == "#":
self.aceptacion = True
self.izq = izq
self.der = der
#END


def getNodo(self):
izq = self.izq.getNodo() if isinstance(self.izq, nodo) else None
der = self.der.getNodo() if isinstance(self.der, nodo) else None


if self.tipo == tipo.HOJA: #TIPO HOJA

self.anulable = False
self.primeros.append(self.numero)
self.ultimos.append(self.numero)

elif self.tipo == tipo.AND: #TIPO CONCATENACION

if ( isinstance(izq, nodo) and isinstance(der, nodo) ):
# Anulable
self.anulable = izq.anulable and der.anulable

# Primeros
```

```
if izq.anulable: #C1 Anulable Se Unen
self.primeros.extend(izq.primeros) #Primeros C1
self.primeros.extend(der.primeros) #Primeros C2
else:
self.primeros.extend(izq.primeros)

# Ultimos
if der.anulable: #C2 Anulable Se Unen
self.ultimos.extend(izq.ultimos) #Ultimos C1
self.ultimos.extend(der.ultimos) #Ultimos C2
else:
self.ultimos.extend(der.ultimos)

elif self.tipo == tipo.OR: #TIPO ALTERNANCIA

if ( isinstance(izq, nodo) and isinstance(der, nodo) ):
# Anulable
self.anulable = izq.anulable or der.anulable #Anulable si cualquier hijo es anulable

# Primeros
self.primeros.extend(izq.primeros) #Siempre se unen los first de los hijos
self.primeros.extend(der.primeros)

# Ultimos
self.ultimos.extend(izq.ultimos) #Siempre se unen los last de los hijos
self.ultimos.extend(der.ultimos)

elif self.tipo == tipo.KLEENE: #TIPO CERRADURA DE KLEENE

if isinstance(izq, nodo):
self.anulable = True #Siempre es anulable
self.primeros.extend(izq.primeros) #Se copial los first y last del hijo
self.ultimos.extend(izq.ultimos)

else:
pass

return self
#END

def siguientes(self):
izq = None if (self.izq == None) else self.izq.siguientes()
der = None if (self.der == None) else self.der.siguientes()


if self.tipo == tipo.AND: #TIPO CONCATENACION
for i in izq.ultimos:
```

```
nodo = metodos.getHoja(i)
metodos.append(nodo.numero, nodo.lexema, der.primeros) #Creo la tabla Siguientes


elif self.tipo == tipo.KLEENE: #TIPO CERRADURA DE KLEENE
for i in izq.ultimos:
nodo = metodos.getHoja(i)
metodos.append(nodo.numero, nodo.lexema, izq.primeros)


else:
pass

return self
#END

def limpiarPrimeros(self):
self.primeros.clear()
#END

def limpiarUltimos(self):
self.ultimos.clear()
#END
```

NumHoja

```
class numHoja:
def __init__(self, Expresion):
self.Expresion = self.limpiar(Expresion) + 1
#END

def limpiar(self, Expresion):
return len(Expresion.replace(".","").replace("|","").replace("*",""))
#END

def getNumHoja(self):
self.Expresion=self.Expresion-1
return self.Expresion
#END
```

constructor Arbol- PilaArbol

```
from numHoja import numHoja
from tipo import tipo
from nodo import nodo
import metodos

class pilaArbol:
n=""
def __init__(self, er):
global n
nh=numHoja(er)
self.pila=[]

for op in reversed(list(er)): #Invierte la cadena

if op == "|":
izq = self.pila.pop(len(self.pila) - 1)
der = self.pila.pop(len(self.pila) - 1)

if (isinstance(izq,nodo) and isinstance(der,nodo)): #Se verifica si son del tipo nodo
n = nodo(op, tipo.OR, 0, izq, der)
self.pila.append(n)

elif op == ".":
izq = self.pila.pop(len(self.pila) - 1)
der = self.pila.pop(len(self.pila) - 1)

if (isinstance(izq,nodo) and isinstance(der,nodo)):
n = nodo(op, tipo.AND, 0, izq, der)
self.pila.append(n)
elif op == "*":
unario = self.pila.pop(len(self.pila) - 1)

if (isinstance(unario,nodo)):
n = nodo(op, tipo.KLEENE, 0, unario, None) #Como es unario, sólo tiene un hijo
self.pila.append(n)
else:
n = nodo(op, tipo.HOJA, nh.getNumHoja(), None, None)
self.pila.append(n)
metodos.addHoja(n) # Se agregan las hojas


self.raiz = self.pila.pop(len(self.pila) - 1)
#END
```

```
def getRaiz(self):
return self.raiz
#END

def limpiarPila(self):
global n
tamanio= len(self.pila)
if tamanio !=0:
self.pila.clear()
n.limpiarPrimeros()
n.limpiarUltimos()
```

Metodos

```
#----------------------------------------HOJAS----------------------------------------
lista = []

def addHoja(nodo):
global lista
lista.append(nodo)
#END

def getHoja(numHoja):
global lista

for hoja in lista:
if hoja.numero == numHoja:
return hoja
return None
#END

def aceptacion(numHoja):
global lista

for h in lista:
if h.numero == numHoja:
return h.aceptacion
return False
#END

def limpiarHojas():
global lista
lista.clear()

#----------------------------------TABLA SIGUIENTES----------------------------------

tabla = []

def append(numNodo, lexema, sigLista):
global tabla

for sig in tabla:
if (sig[0] == numNodo) and (sig[1] == lexema):
for new in sigLista:
if not(new in sig[2]):
sig[2].append(new)

return
```

```
tabla.append( [numNodo, lexema, sigLista] )
#END

def getSig(numNodo):
global tabla

for sig in tabla:
if (sig[0] == numNodo):
return sig[1],sig[2]

return "",[]
#END

def limpiarTabla():
global tabla
tabla.clear()

#-----------------------------------Grafo---------------------------------------

listER=[]

def addER(er):
global listER
listER.append(er)

def getListER():
global listER
return listER

def limpiarER():
global listER
listER.clear()
#----------------------------------Analisis Sintactico-----------------------------
pila = []

def addPila(Caracter):
global pila
pila.append(Caracter)

def popPila():
global pila
pila.pop()

def limpiarPila():
global pila
pila.clear()
```

```
def getPila():
global pila
return pila
```

Reporte de Errores

```
import webbrowser

class ReporteErrores:
def GenerarReporte (self, Lista, Titulo):
f=open('ReporteErrores.html','w')
self.Texto="<h1>"+Titulo+"</h1>\n"
self.Texto+="<table border=\"1\">"
self.Texto+=self.Tabla(Lista)
f.write(str(self.Texto))
f.close()

webbrowser.open_new_tab('ReporteErrores.html')
#END

def Tabla(self,Lista):
self.Texto+="<tr>\n"
self.Texto+="<th>Linea</th>\n"
self.Texto+="<th>Columna</th>\n"
self.Texto+="<th>Error</th>\n"
self.Texto+="</tr>\n"

contador=0
for token in Lista:
self.Texto+="<tr>\n"
self.Texto+="<td>"+str(token[0])+"</td>\n"
self.Texto+="<td>"+str(token[1])+"</td>\n"
self.Texto+="<td>"+str(token[2])+"</td>\n"
self.Texto+="</tr>\n"

self.Texto+="</table>"

return self.Texto
#END
```

Generador de Rutas

```
import platform as pl
import os.path as osp
import os
Ruta =""
class Rutas:
def CrearRuta(self,LSTtoken, Texto,Extension):
global Ruta
self.sistema=""
Ruta= self.obtenerPaths(LSTtoken)
existencia=osp.exists(Ruta)

if(existencia==True):
print("Se guardaran archivos aquí"+Ruta)
self.CrearGuardar(Ruta,Texto,Extension)
else: #Se crea ruta si no encuentra la ruta ingresada
if not os.path.isdir(Ruta):
oldmask=os.umask(0)
os.makedirs(Ruta.rstrip(),mode =755)
os.umask(oldmask)
existencia=osp.exists(Ruta)
if(existencia==True):
self.CrearGuardar(Ruta,Texto,Extension)

#END

def CrearGuardar(self,Ruta,Texto,Extension):
if self.sistema.lower()=="linux".lower():
Archivo = open(Ruta+"/Recuperacion."+Extension, "w")
Archivo.write(Texto)
Archivo.close()
elif self.sistema.lower()=="windows".lower():
Archivo = open(Ruta+"\Recuperacion."+Extension, "w")
Archivo.write(Texto)
Archivo.close()
#END

def obtenerPaths(self,listTokens):
PathObtenida=""
Path1=listTokens[0]
Path2=listTokens[1]

Buscando1= str(Path1[3])
Buscando2= str(Path2[3])
```

```
buquedaSegura1=Buscando1.lower()
buquedaSegura2=Buscando2.lower()
palabra1="pathl".lower()
palabra2="pathw".lower()

self.sistema = pl.system()

if(str(self.sistema).lower() =="linux".lower()):
if(buquedaSegura1.find(palabra1)!=-1):
PathObtenida=self.extraerRuta(Buscando1)
return PathObtenida
elif (buquedaSegura2.find(palabra1)!=-1):
PathObtenida=self.extraerRuta(Buscando2)
return PathObtenida
elif (str(self.sistema).lower=="windows".lower()):
if(buquedaSegura1.find(palabra2)!=-1):
PathObtenida=self.extraerRuta(Buscando1)
return PathObtenida
elif (buquedaSegura2.find(palabra2)!=-1):
PathObtenida=self.extraerRuta(Buscando2)
return PathObtenida
else:
PathObtenida="No se encontró"
return PathObtenida
#END

def extraerRuta(self, rut):
contador=0
cadena=""
while(rut[contador]!=":"):
cadena+=rut[contador]
contador+=1
contador+=1
cadena=""
tamnioRuta=len(rut)-1
while(contador<tamnioRuta):
cadena+=rut[contador]
contador+=1
cadena+=str(rut[contador])
return cadena.lstrip()
#END

def setRuta(self,ruta):
global Ruta
Ruta=ruta
#END
```

```
def getRuta(self):
global Ruta
return Ruta
#END


#END
```

Tabla Transiciones

```
import metodos
from transicion import transicion
from graphviz import Digraph
class tablaTran:

def __init__(self, raiz):
self.estados = []
self.cont = 0

# [ nombre, elementos, transiciones, Aceptacion]
self.estados.append( ["S"+str(self.cont), raiz.primeros, [], False] )
self.cont += 1

for estado in self.estados:
elementos = estado[1]

for hoja in elementos:

lexema, siguientes = metodos.getSig(hoja)

estado_existe = False
estado_encontrado = ""
for e in self.estados:
if "".join(str(v) for v in e[1]) == "".join(str(v) for v in siguientes):
estado_existe = True
estado_encontrado = e[0]
break


if not estado_existe:
if metodos.aceptacion(hoja):
estado[3] = True
if lexema == "":
continue

nuevo = ["S"+str(self.cont), siguientes, [], False]
trans = transicion(estado[0], lexema, nuevo[0])
estado[2].append(trans)

self.cont += 1
self.estados.append(nuevo)

else:
if metodos.aceptacion(hoja):
```

```
estado[3] = True
trans_existe = False

for trans in estado[2]:
if trans.comparar(estado[0], lexema):
trans_existe = True
break

if not trans_existe:
trans = transicion(estado[0], lexema, estado_encontrado)
estado[2].append(trans)
#END

def grafo(self,L):
dot = Digraph(name='child')
dot.attr('node', shape='circle')

# Creamos los nodos
for e in self.estados:
dot.node(e[0]+L,e[0]+L)
if e[3]:
dot.node(e[0]+L, shape='doublecircle')

#Creamos las transiciones
for e in self.estados:
for t in e[2]:
dot.edge(t.eIni+L, t.eFin+L, label=t.tran)
texto=dot.source
#dot.render('/home/audrie8a/Escritorio/Estados.gv', view=False)
#print("Grafo de Estados Generado")
return texto
#END
```

Tipo

```
from enum import Enum

class tipo(Enum):
HOJA=1
AND=2
OR=3
KLEENE=4
```

Transiciones

```
class transicion:

def __init__(self, eIni, tran, eFin):
self.eIni = eIni
self.tran = tran
self.eFin = eFin
#END


def comparar(self, eIni, tran):
if (self.eIni == eIni) and (self.tran == tran):
return True

return False
#END


def string(self):
return self.eIni+" "+self.tran+" "+self.eFin
#END
```