Universidad De San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas

Lenguajes Formales y de Programación
Sección "B-"

MANUAL TÉCNICO

Carlos Augusto Calderón Estrada

201905515

Objetivos

General:

Brindarle al desarrollador a trabajar en esta aplicación una guía lo más completa y sencilla posible para el mejor entendimiento de lo que se lleva desarrollado en la aplicación y así evitar errores posibles por el uso incorrecto de los algoritmos y/o estructuras.

Específicos:

- Que el desarrollador trabaje en la aplicación como si fuera creada por el/ella.
- Proporcionarle al lector una explicación sencilla y técnica de los procesos algorítmicos y las relaciones de los métodos, funciones y atributos que son esenciales en la aplicación.

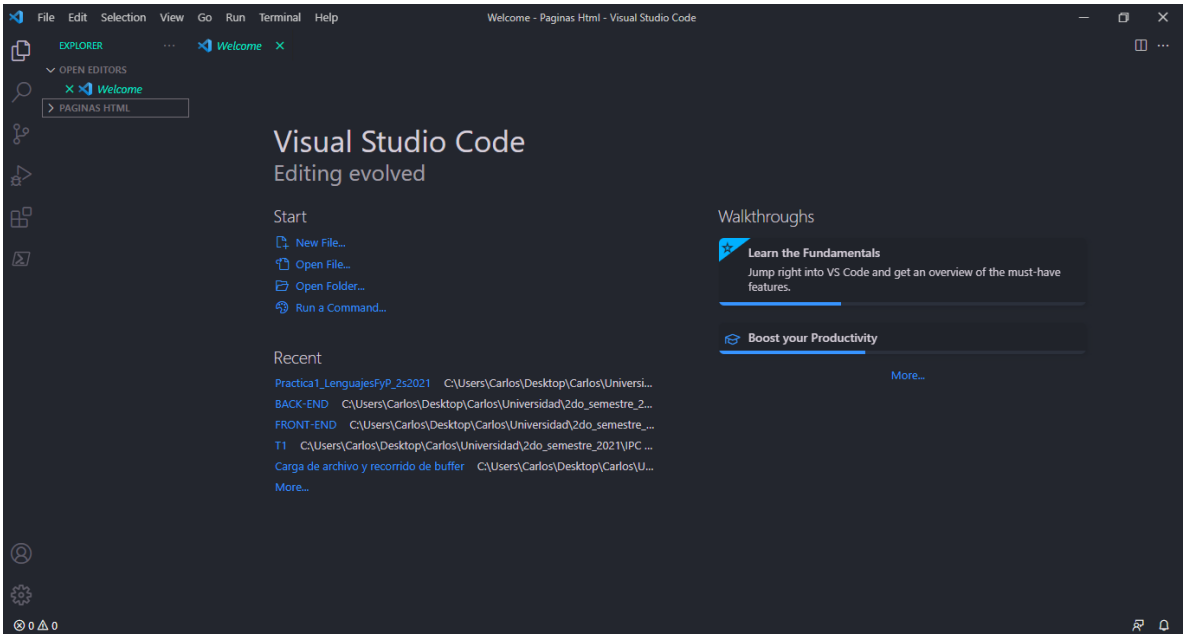
Introducción

Este manual técnico tiene como fin dar a conocer al desarrollador las mejores recomendaciones, requerimientos para un mejor entendimiento al momento de querer realizar modificaciones, optimización de procesos o métodos, indicando el editor de texto que se utilizó para el desarrollo de dicha aplicación.

La aplicación se basa en Analizar léxicamente un archivo por medio de un autómata finito creado por el desarrollador, en el que se cargara por medio de una ventana emergente y este carga el archivo al programa(todo esto a través de una interfaz gráfica de Python) y se procede a colocar los datos dentro de la caja de texto de entrada llamada **código,** al momento de hacer click en la pestaña analizar por medio de métodos recursivos y sentencias de control se realiza el análisis léxico y sintáctico, en el que cualquier desarrollador con un poco de experiencia en programación y un poco de conocimiento sobre el concepto de recursividad conoce la forma en que este método trabaja, se basa el autómata leyendo los tokens y almacenándolas en una sola lista que será la compartida entre clases por medio de retornos, que serán utilizadas como para el análisis sintáctico y son indispensables para ir concatenando en la variable a retornar, luego de realizar el análisis, se desplazara en la consola de la aplicación (no del interprete) las respuestas de los comandos, a su vez se al momento de obtener un error léxico o sintáctico el programa no se termina sino que ejecuta el reporte de errores y estos mismos indican donde se realizó. Con respecto a los reportes, se programaron por medio de sentencias de control y ciclos creando una página agradable que muestre la tabla correspondiente, a diferencia del árbol de derivación que se programó en Graphviz y se traduce a un archivo pdf que se abre después terminar esa función.

Editor de texto utilizado

1. El editor de texto utilizado fue Visual Studio Code gracias a la variedad de extensiones que posee y su fácil uso para el manejo de la nube git, que permite agregar la propia terminal y ejecutar los comandos desde el entorno de VS Code.



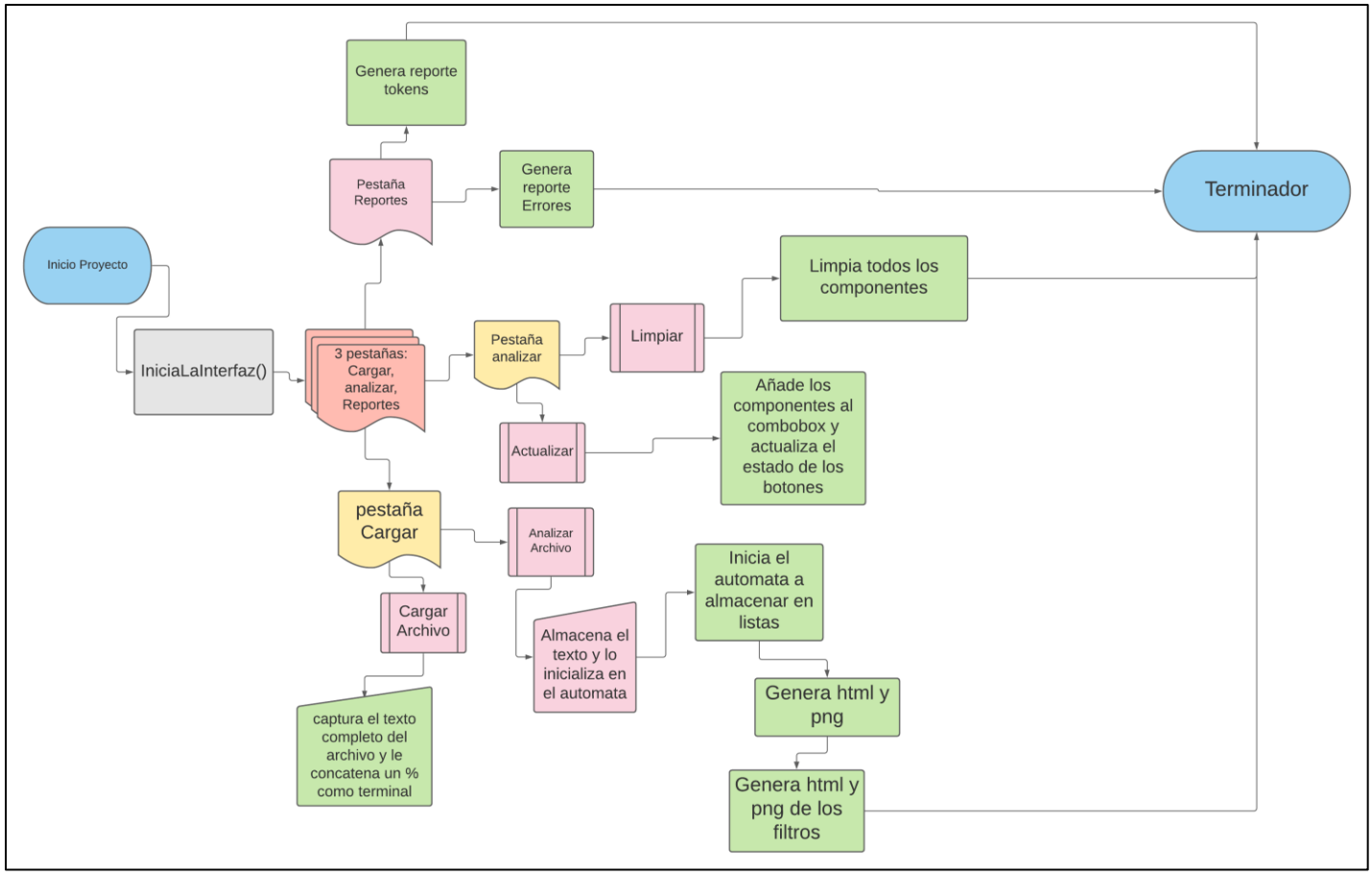
Requerimientos:

- Procesador de 1.6 GHz o más.
- 1 GB de memoria RAM
- Plataformas aprobadas: OS X, Windows 7, Linux (Debian), Linux (Red Hat)
- Microsoft .NET Framework 4.5.2 para VS Code

Sistema Operativo que se llevó a cabo:

Windows 10 de 64 bits

Diagrama de flujo



Lógica del Programa

Main:

La clase principal crea el objeto de la interfaz y la ejecuta para poder trabajar en ella, si se cierra, se cierra el programa

```
from PyQt5 import QtCore, QtGui, QtWidgets
from Interfaz import Ui_MainWindow
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Cargar Archivo:

```
def CargarArchivo(self): ##Inicio del metodo
        #self.Tokens.clear()## al cargar un nuevo archivo se borran los
tokens actuales
        #self.Errores.clear()## al cargar un nuevo archivo se borran los
erroes actuales
        global Text
        ##importamos la opciones para la ventana emergente
        Tk().withdraw()
        filee = filedialog.askopenfile(initialdir="./Archivos Prueba",
        title="Select a file",filetypes=(("Files lfp",".lfp"),
        ("ALL files",".txt")))
        if filee is None:
            print('No file was selected\n')
            return None
        else:
            ##abrimos el archivo y lo leemos
            Text = filee.read()

            ##concatenamos el simbolo terminal
            Text+='\n$'
            filee.close()##Cerramos el archivo
            print("Lectura Exitosa")
            return Text
```

Función isLetra:

```
def isLetter(self,Character):##metodo que retorna si es una letra alfabetica
        if((ord(Character) >= 65 and ord(Character) <= 90)
        or (ord(Character) >= 97 and ord(Character) <= 122)
        or ord(Character) == 164 or ord(Character) == 165):
            return True
        else:
            return False
```

Función isNumero:

```
def isNum(self,Character):##metodo que retorna si es un digito
        if ((ord(Character) >= 48 and ord(Character) <= 57)):
            return True
        else:
            return False
```

Funcion isCharacter:

```
def isCharacter(self, Character):
        if ((ord(Character)>=32 and ord(Character)!=34)):
            return True
        elif (160<=ord(Character)<=165 or ord(Character)==129 or
ord(Character)==130):
            return True
        elif ord(Character)==34:
            return False
```

Función Analizar (class: AutomataLexico):

En esta función se realiza el análisis léxico de la caja de entrada.

```
def analizar(self, texto):  #inserta solamente al final
    #inicializar listas nuevamente
    self.listaTokens = []
    self.listaErrores = []
    estado = 0
    fila =1
    columna=1
    lexema = ''
    i = 0
    #principio de ciclo
    while i <len(texto):
        x=texto[i]
        ## inicia lectura de simbolos
        if estado==0:···
        ## Aqui termina la lectura de simbolos o entradas para la lectura de comandos
        elif estado==1:···
        elif estado==2:···
        elif estado==3:···
        elif estado==4:···
        elif estado==5:···
        elif estado==6:···
        elif estado==7:···
        elif estado==8:···
        elif estado==9:···
        elif estado==10:···
        elif estado==11:···
        i+=1
    return self.listaTokens
```

Inicio del recorrido del archivo

```
 while i <len(texto):#ciclo para recorrer el texto
            x=texto[i]
```

- **Ejemplo de los estados en el automata**

```
if estado==0:
                if x == '=':
                    self.listaTokens.append(Token("=",'igual',fila,columna))
                    columna+=1
                elif x=='[':
                    self.listaTokens.append(Token("[",'corchete1',fila,colum
na))
                    columna+=1
                elif x==']':
                    self.listaTokens.append(Token("]",'corchete2',fila,colum
na))
                    columna+=1
                elif x=='{':
                    self.listaTokens.append(Token("{",'llave1',fila,columna)
)
                    columna+=1
                elif x=='}':
                    self.listaTokens.append(Token("}",'llave2',fila,columna)
)
                    columna+=1
                elif x=='(':
                    self.listaTokens.append(Token("(",'parentesis1',fila,col
umna))
                    columna+=1
                elif x==')':
                    self.listaTokens.append(Token(")",'parentesis2',fila,col
umna))
                    columna+=1
                elif x==';':
                    self.listaTokens.append(Token(";",'puntoycoma',fila,colu
mna))
                    columna+=1
                elif x==',':
                    self.listaTokens.append(Token(",",'coma',fila,columna))
                    columna+=1
                elif x=='"':
```

```
                    self.listaTokens.append(Token('"',
'ComillaDoble',fila,columna))
                    columna+=1
                    estado=7
                elif x=="#":
                    self.listaTokens.append(Token('#',
'CommentSimple',fila,columna))
                    columna+=1
                    estado=1
                elif x=='.':
                    self.listaTokens.append(Token('.',
'punto',fila,columna))
                    columna+=1
                elif x=="'":
                    self.listaTokens.append(Token("'",
'ComillaSimple',fila,columna))
                    columna+=1
                    estado=2
                elif x=='\n':
                    fila+=1
                    columna=1
                elif self.isNum(x)==True:
                    lexema+=x
                    columna+=1
                    estado=9
                elif self.isLetter(x)==True:
                    lexema+=x
                    columna+=1
                    estado=11
                elif x=='$':
                    self.listaTokens.append(Token('$','Fin',fila,columna))
                    print('fin de lectura')
                else:
                    e='Error lexico en fila '+str(fila)+' y columna
'+str(columna)+" revisar"
                    self.listaErrores.append(Errores(fila, columna, e,
"Simbolo, letra o digito"))
            ## Aqui termina la lectura de simbolos o entradas para la
lectura de comandos
```

- Análisis sintáctico:

Este método lee y ejecuta los comandos ya almacenados en la lista de tokens:

```
def analizar(self, listaTokens, listaErrores):
        global i, ReturnText,NodeData, NodosCrear
        ReturnText=NodeData=NodosCrear=''
        i=0
        self.InicializarContadores()
        self.listaTokens = listaTokens
        self.listaErrores = listaErrores
        self.listClaves=[]
        self.listRegisters=[]
        self.iniciar()
        return ReturnText
def iniciar(self):
        global NodeData,CountInstrucciones,NodosCrear
        NodosCrear+='Instrucciones'+str(CountInstrucciones)+'[label="Instruc
ciones"]\n'
        NodeData+='inicio->Instrucciones'+str(CountInstrucciones)+'\n'
        self.lista_Instrucciones()
def lista_Instrucciones(self):
        global i, NodeData, CountInstrucciones, NodosCrear,
CountInstruccion, CountRegistro,CountListRegistro
        global CountProceso

        if self.listaTokens[i].lexema == 'registros':
            CountInstrucciones+=1
            CountInstruccion+=1
            NodosCrear+='Instrucciones'+str(CountInstrucciones)+'[label="Ins
trucciones"]\n'
            NodosCrear+='Instruccion'+str(CountInstruccion)+'[label="Instruc
cion"]\n'
            NodeData+='Instrucciones'+str(CountInstrucciones-1)+'->
Instruccion'+str(CountInstruccion)+'\n'
            NodeData+='Instruccion'+str(CountInstruccion)+'-
>Proceso_Registros\n'
            self.instruccion_Registros()
            NodeData+='Instrucciones'+str(CountInstrucciones-1)+'->
Instrucciones'+str(CountInstrucciones)+'\n'
            self.lista_Instrucciones()
```

En este ejemplo de análisis sintáctico se puede ver la forma recursiva en que despues de la ejecución se vuelve a llamar a sí misma la clase

- Ejemplo de programación de un comando: en este ejemplo se muestra como fueron programándose cada comando en base a la lista de tokens.

```
## EMPIEZA LA FUNCION IMPRIMIR
def instruccion_Imprimir(self):
    global i, ReturnText, CountSimboloIgual,contadorImprimir, NodeData, NodosCrear, CountProceso,
    ##inicia el comando con el token imprimir
    if self.listaTokens[i].lexema == 'imprimir':
        NodosCrear+='tokenImprimir'+str(countToken)+'[label="tokenImprimir"]\n'
        NodosCrear+='imprimir'+str(countToken)+'[label="imprimir"]\n'
        NodeData+='ProcesoImprimir'+str(CountProceso)+'->tokenImprimir'+str(countToken)+'->imprim
        i+=1
        #el siguiente token es un parentesis ()
        if self.listaTokens[i].lexema == 'parentesis1':
            NodosCrear+='SimboloparentesisA'+str(CountSimboloPa)+'[label="SimboloparentesisA"]\n'
            NodosCrear+='parentesisA'+str(CountSimboloPa)+'[label="("]\n'
            NodeData+='ProcesoImprimir'+str(CountProceso)+'->SimboloparentesisA'+str(CountSimbolo
            CountSimboloPa+=1
            i+=1
```

Y así sucesivamente. Los métodos **Nodos crear se explicarán más adelante.**

- Reporte tokens y errores: Se basa en la escritura de un archivo en HTML, ya que es recorriendo la lista de tokens o errores, pero adicional lo agregamos a una tabla simple y le agregamos el mismo estilo. **Adjunto lógica de reporte de tokens como ejemplo.**

```
-     global listaTokens
-           Archivo = open("./Reportes/Tokens.html",'w')
-           contenidoTabla = ''
-           for i in range(len(listaTokens)):
-               contenidoTabla+='<tr><th scope="row">'+str(i+1)+'</th>\n'
-               contenidoTabla+='<td>'+listaTokens[i].token+'</td>\n'
-               contenidoTabla+='<td>'+listaTokens[i].lexema+'</td>\n'
-               contenidoTabla+='<td>'+str(listaTokens[i].Fila)+'</td>\n'
-               contenidoTabla+='<td>'+str(listaTokens[i].Columna)+'</td>\
    n</tr>'
-           contenidoHTML=(
-                 '<!DOCTYPE html>\n'
-                   ' <html>\n'
-                   '<head> \n'
-                   '<meta charset="utf-8"> \n'
-                   '<link href="assets/css/bootstrap-responsive.css"
    type="text/css" rel="stylesheet">\n'
-                   '<link
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.mi
    n.css" type="text/css" rel="stylesheet">\n'
-                   '<link rel="stylesheet" type="text/css"
    href="./css/bootstrap.css">\n'
```

```
-                   '<link rel="stylesheet"
    type="text/css"  href="css/Style.css">'
-                   '<title>Reporte de Tokens' '</title>\n'
-                   '</head>\n'
-                   '<body>\n'
-                   '<div class="container-fluid welcome-page"
    id="home">\n'
-                   '   <div class="jumbotron">\n'
-                   '       <h1>\n <span>\nTokens\n</span>\n
    </h1>\n<p>Reporte con todos los tokens, lexemas, sus fila y sus
    columna</p>\n'
-                   '</div>'
-                   '</div>'
-                   '<div class="container-fluid " ><div
    class="jumbotron">'
-                   '<table class="table table-responsive">\n'
-                   '   <thead>\n'
-                       '<tr>\n'
-                           '<th scope="col">#</th>\n'
-                           '<th scope="col">Token</th>\n'
-                           '<th scope="col">Lexema</th>\n'
-                           '<th scope="col">Fila</th>\n'
-                           '<th scope="col">Columna</th>\n'
-                       '</tr>\n'
-                   '</thead>\n'
-                   '<tbody>\n'
-                   +contenidoTabla+
-                   '</tbody>\n'
-                   '</table>'
-                   '</div>'
-                   '</div>\n''</body>\n''</html>\n'
-               )
-           Archivo.write(contenidoHTML)
-           Archivo.close()
-           webbrowser.open("file:///"+os.getcwd()+"/Reportes/Tokens.html"
    )
```

Interfaz: Se utilizó PyQt5 como librería para la creación de la ventana y sus componentes

```
-   class Ui_MainWindow(object):
-       def setupUi(self, MainWindow):
-           MainWindow.setObjectName("MainWindow")
-           MainWindow.resize(1318, 700)
-           self.centralwidget = QtWidgets.QWidget(MainWindow)
-           self.centralwidget.setObjectName("centralwidget")
-           self.txtCodigo = QtWidgets.QPlainTextEdit(self.centralwidget)
-           self.txtCodigo.setGeometry(QtCore.QRect(20, 40, 551, 611))
-           self.txtCodigo.setObjectName("txtCodigo")
-           self.plainTextEdit_2 =
    QtWidgets.QPlainTextEdit(self.centralwidget)
-           self.plainTextEdit_2.setGeometry(QtCore.QRect(590, 40, 711,
    611))
-           font = QtGui.QFont()
-           font.setFamily('MingLiU-ExtB')
-           font.setPointSize(11)
-           self.plainTextEdit_2.setFont(font)
-           self.plainTextEdit_2.setObjectName("plainTextEdit_2")
-           self.plainTextEdit_2.setReadOnly(True)
-           self.lblCodigo = QtWidgets.QLabel(self.centralwidget)
-           self.lblCodigo.setGeometry(QtCore.QRect(260, 10, 71, 31))
-           font = QtGui.QFont()
-           font.setFamily("Calibri")
-           font.setPointSize(16)
-           font.setBold(True)
-           font.setWeight(75)
-           self.lblCodigo.setFont(font)
-           self.lblCodigo.setObjectName("lblCodigo")
-           self.lblSalida = QtWidgets.QLabel(self.centralwidget)
-           self.lblSalida.setGeometry(QtCore.QRect(980, 10, 61, 16))
-           font = QtGui.QFont()
-           font.setFamily("Calibri")
-           font.setPointSize(16)
-           font.setBold(True)
-           font.setWeight(75)
-           self.lblSalida.setFont(font)
-           self.lblSalida.setObjectName("lblSalida")
-           MainWindow.setCentralWidget(self.centralwidget)
-           self.menubar = QtWidgets.QMenuBar(MainWindow)
-           self.menubar.setGeometry(QtCore.QRect(0, 0, 1418, 21))
-           self.menubar.setObjectName("menubar")
-           self.menuCargar_Archivo = QtWidgets.QMenu(self.menubar)
-           self.menuCargar_Archivo.setObjectName("menuCargar_Archivo")
```

```
-           self.menuAnalizar = QtWidgets.QMenu(self.menubar)
-           self.menuAnalizar.setObjectName("menuAnalizar")
-           self.menuReportes = QtWidgets.QMenu(self.menubar)
-           self.menuReportes.setObjectName("menuReportes")
-           MainWindow.setMenuBar(self.menubar)
-           self.statusbar = QtWidgets.QStatusBar(MainWindow)
-           self.statusbar.setObjectName("statusbar")
-           MainWindow.setStatusBar(self.statusbar)
-           self.actionTokens = QtWidgets.QAction(MainWindow)
-           self.actionTokens.setObjectName("actionTokens")
-           self.actionErrores = QtWidgets.QAction(MainWindow)
-           self.actionErrores.setObjectName("actionErrores")
-           self.actionArbol_de_derivacion = QtWidgets.QAction(MainWindow)
-           self.actionArbol_de_derivacion.setObjectName("actionArbol_de_d
    erivacion")
-           self.actionCargar = QtWidgets.QAction(MainWindow)
-           self.actionCargar.setObjectName("actionCargar")
-           self.actionAnalizar = QtWidgets.QAction(MainWindow)
-           self.actionAnalizar.setObjectName("actionAnalizar")
-           self.menuCargar_Archivo.addAction(self.actionCargar)
-           self.menuAnalizar.addAction(self.actionAnalizar)
-           self.menuReportes.addAction(self.actionTokens)
-           self.menuReportes.addAction(self.actionErrores)
-           self.menuReportes.addAction(self.actionArbol_de_derivacion)
-           self.menubar.addAction(self.menuCargar_Archivo.menuAction())
-           self.menubar.addAction(self.menuAnalizar.menuAction())
-           self.menubar.addAction(self.menuReportes.menuAction())
-           self.actionCargar.triggered.connect(self.addFile)
-           self.actionAnalizar.triggered.connect(self.Analizar)
-           self.actionTokens.triggered.connect(self.ReporteTokens)
-           self.actionArbol_de_derivacion.triggered.connect(self.ReporteA
    rbol)
-           self.actionErrores.triggered.connect(self.ReporteErrores)
-           self.retranslateUi(MainWindow)
-           QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

- Método retranslateUi: Establece los nombres visibles de los componentes.

```
-   def retranslateUi(self, MainWindow):
-           _translate = QtCore.QCoreApplication.translate
-           MainWindow.setWindowTitle(_translate("MainWindow", "Proyecto
    2"))
-           self.lblCodigo.setText(_translate("MainWindow", "Codigo"))
-           self.lblSalida.setText(_translate("MainWindow", "Salida"))
-           self.menuCargar_Archivo.setTitle(_translate("MainWindow",
    "Cargar Archivo"))
-           self.menuAnalizar.setTitle(_translate("MainWindow",
    "Analizar"))
-           self.menuReportes.setTitle(_translate("MainWindow",
    "Reportes"))
-           self.actionTokens.setText(_translate("MainWindow", "Tokens"))
-           self.actionErrores.setText(_translate("MainWindow",
    "Errores"))
-           self.actionArbol_de_derivacion.setText(_translate("MainWindow"
    , "Arbol de derivacion"))
-           self.actionCargar.setText(_translate("MainWindow", "Cargar"))
-           self.actionAnalizar.setText(_translate("MainWindow",
    "Analizar"))
-           self.pushButton_2.setText(_translate("Bixelart", "Analizar Arc
    hivo"))
-           self.toolBox.setItemText(self.toolBox.indexOf(self.Cargar), _t
    ranslate("Bixelart", "Cargar Archivo"))
-           self.lblImagen.setText(_translate("Bixelart", ""))
-           self.groupBox.setTitle(_translate("Bixelart", "Botones"))
-           self.btnOriginal.setText(_translate("Bixelart", "Original"))
-           self.btnMiX.setText(_translate("Bixelart", "MirrorX"))
-           self.btnMiY.setText(_translate("Bixelart", "MirrorY"))
-           self.btnDMi.setText(_translate("Bixelart", "DoubleMirror"))
-           self.btnActCombo.setText(_translate("Bixelart", "Actualizar"))
-           self.btnLimpiar.setText(_translate("Bixelart", "Limpiar"))
-           self.toolBox.setItemText(self.toolBox.indexOf(self.Analizar),
    _translate("Bixelart", "Analizar"))
-           self.pushButton_7.setText(_translate("Bixelart", "Reporte De T
    okens"))
-           self.pushButton_8.setText(_translate("Bixelart", "Reporte De E
    rrores"))
-           self.label_2.setText(_translate("Bixelart", "Por Favor presion
    e el boton para Generar el Reporte deseado"))
-           self.toolBox.setItemText(self.toolBox.indexOf(self.GenerarRepo
    rtes), _translate("Bixelart", "Reportes"))
-
```

función addFile: Agrega los datos a la caja de texto Codigo

```
def addFile(self):
        self.plainTextEdit_2.clear()
        self.txtCodigo.clear()
        Text = gestor.CargarArchivo()
        self.txtCodigo.insertPlainText(Text)
```

función Analizar: Ejecuta el análisis y la salida de texto en consola de la aplicación

```
def Analizar(self):
        global listaTokens, listaErrores
        try:
            self.plainTextEdit_2.clear()
            Text = self.txtCodigo.toPlainText()
            listaTokens = Auto.analizar(Text)
            listaE = Auto.listE()
            textoSalida=gestorSintactico.analizar(listaTokens,listaE)
            listaErrores=gestorSintactico.listEr()
            self.plainTextEdit_2.insertPlainText(textoSalida)
        except:
            listaErrores=gestorSintactico.listEr()
            self.ReporteErrores()
            print('Murio xd')
```

Función Reporte Arbol: Esta función por medio de las variables NodosCrear y NodeData se crean archivo **.dot** y se traslada a **.pdf**

```
def ReporteArbol(self):
        gestorSintactico.ArbolDeDerivacion()
def ArbolDeDerivacion(self):
        global NodeData, NodosCrear
        f = open('ArchivosDots/archivoArbolDerivacion.dot', 'w',
encoding='utf-8')
        graph=''
        contenidoDot=(
            'digraph Arbol{\n'
            '{\n''node [margin=0 fontsize=20 width=0.5 style=filleed]\n'
            +NodosCrear+'}'
            +NodeData+'}')
        f.write(contenidoDot)
        f.close()
        os.system("dot -Tpdf ArchivosDots/archivoArbolDerivacion.dot -o
./Reportes/Arbol_De_Derivacion.pdf")
        webbrowser.open("file:///"+os.getcwd()+"/Reportes/Arbol_De_Derivacio
n.pdf")
```

Librerías Utilizadas

Las librerías que se utilizaron para el desarrollo de esta practica fueron:

Imports locales:

- from Token import Token
- from Errores import Errores
- from AutomataLexico import AutomataLexico
- from GestorSintactico import *
- from Gestor import *
- import os, webbrowser

Imports TKINTER: Estas librerías fueron utilizadas para la carga del archivo

- from tkinter import *
- from tkinter import filedialog, Tk
- from tkinter.filedialog import askopenfilename

Imports herramienta grafica

- from graphviz import *
- from prettytable import *

Imports Interfaz PyQt5: Estas librerías fueron utilizadas para la creación de la interfaz gráfica.

- from PyQt5 import QtCore, QtGui, QtWidgets

Descripción de Tokens

L = [a-zA-ZñÑ]

D=[0-9]

R=[0-9, '.']

S=['{', '}', ' [', ']', ';', '=' - ',' - '(' - ')' , ' " ', " ' "]

Cadena = "(^")*"

Comandos = [(L+)(S)("(^")*")(S+)]

#	Token	Descripción	Tipo de dato	ER
1	Reservada	Este token representa una palabra reservada	String	**L+**
2	Cadena	Este token representa una línea de texto que puede optar por todos los caracteres	String	**"(^")*"**
3	Digito	Este token representa un digito entero	Int [0,9]	**D+**
4	Letra	El carácter que puede formar un comando	String	**L+**
5	Real	Este token representa una variable real o decimal	float	**D+|R?**
6	Símbolos	Este token representa los tokens de los símbolos que pueden presentarse durante el análisis léxico	String	**S?**
7	Comandos	Este token representa las instrucciones que se pueden realizar dentro del análisis	String	**(L+)(S)("(^")*")(S+)**
8	$	Este token representa el final de la ejecución	String	**$?**