



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Lenguajes Formales y de Programación

MANUAL TÉCNICO

Henry Ronely Mendoza Aguilar
Carné: 202004810
PROYECTO 2

Objetivos y Alcances del Sistema

Objetivo Principal

- Brindar una interfaz que permita el análisis y procesamiento de datos de empresas dedicadas a ventas o que manejen inventarios en general.

Objetivos Específicos

- Diseñar una plataforma agradable e intuitiva por medio del lenguaje de programación Python.
- Aprovechar de mejor manera los recursos brindados por el editor de texto VS code y el designer de PyQt5.
- Programar un sistema funcional para cualquier usuario y hardware en el que se utilice.

Requisitos del Hardware

- 512 MB de RAM
- 20MB de disco duro para almacenamiento del software (ya que puede aumentar con la generación de imágenes).
- Procesador Pentium 2 a 266Mhz

Requisitos del Software

- Python 3
- Windows 10 (64-bit), Linux o Mac OS

Descripción de Métodos

Para el desarrollo de la aplicación se emplearon los métodos de lectura de archivos de texto plano que posee Python por defecto por lo que no se importaron librerías externas. Los métodos utilizados fueron “read_file()” el cual es el encargado de abrir el archivo que contiene los comandos y claves y registros, seguidamente se añade a recuadro de texto editable.

```
def read_file(self):
    buscar = QFileDialog.getOpenFileName()
    extension=buscar[0].split('.')

    if extension[1] == 'lfp':
        file=open(buscar[0],'r')
        content=file.read()
        file.close()

        msj = QMessageBox()
        msj.setWindowTitle('Información')
        msj.setText('Archivo cargado correctamente')
        msj.exec()
        self.archivo = content
        self.plainTextEdit.setPlainText(self.archivo)
    else:
        msj = QMessageBox()
        msj.setWindowTitle('Error')
        msj.setText('Formato de archivo incorrecto')
        msj.exec()
```

Función analizar(): en esta función se instancian las clases de los analizadores léxicos y sintácticos a su vez genera los html de tokens, errores y el árbol de derivación en pdf. Seguidamente añade los resultados

```
def analizar(self):
    archivo = self.plainTextEdit.toPlainText()
    if archivo == '':
        msj = QMessageBox()
        msj.setWindowTitle('Error')
        msj.setText('No se ha cargado el archivo')
        msj.exec()
    else:
        self.lexico.analizar(archivo)
        self.sintactico.analizar(self.lexico.tokens,self.lexico.errores)
        from arbol import consola
        self.textBrowser.setPlainText(consola)
        self.lexico.del_from_token()
        self.lexico.html_T()
        self.lexico.html_E()
```

Almacenamiento de datos: Para el almacenamiento de datos se utilizó la programación orientada a objetos en el cual se creó una clase llamada claves para almacenar dentro de la instancia de esta clase los datos que contiene el archivo de entrada en el apartado de claves y registros ya que forma una lista de listas.

```
class claves:
    def __init__(self,id):
        self.id = id #nombre de la clave
        self.valores = [] #lista

    def setValores(self,valor):
        self.valores.append(valor)

    def getValores(self):
        print('-----')
        print(self.id)
        for v in self.valores:
            print('    ->'+v)
```

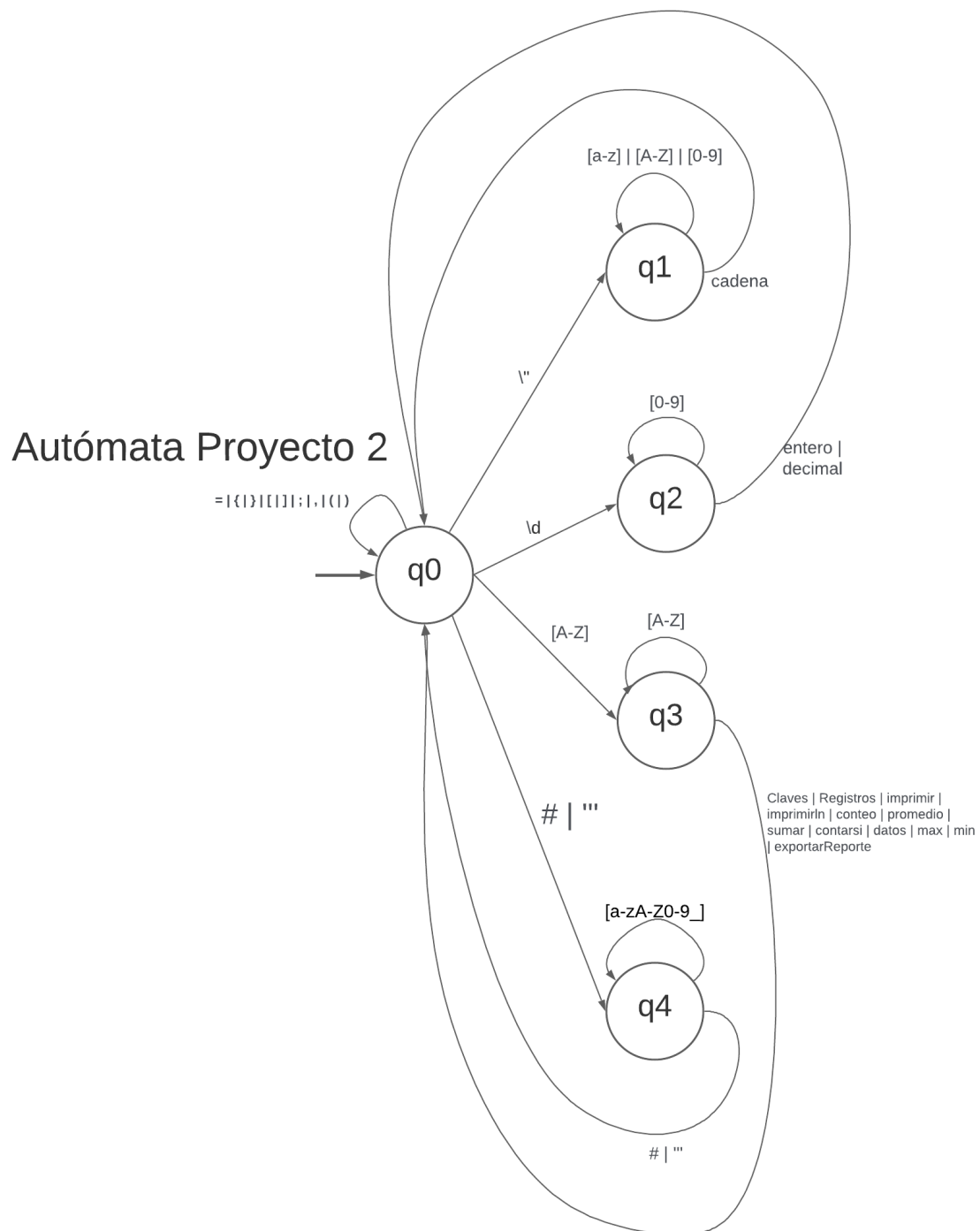
Autómata: Para el análisis de tokens dentro de el programa se procedió a crear una clase llamada Analizador. En esta se implementó una función llamada analizar en la que se contiene un autómata finito determinista el cual se utilizó como un Analizador Léxico. En el cual se aceptan los siguientes tokens:

Tabla de Explicación de Tokens

Token	Patrón	Color
claves	Claves	Azul
registro	Registros	Azul
imprimir	imprimir	Azul
imprimirln	imprimirln	Azul
conteo	conteo	Azul
promedio	promedio	Azul
contarsi	contarsi	Azul
datos	datos	Azul
sumar	sumar	Azul
maximo	max	Azul
minimo	min	Azul

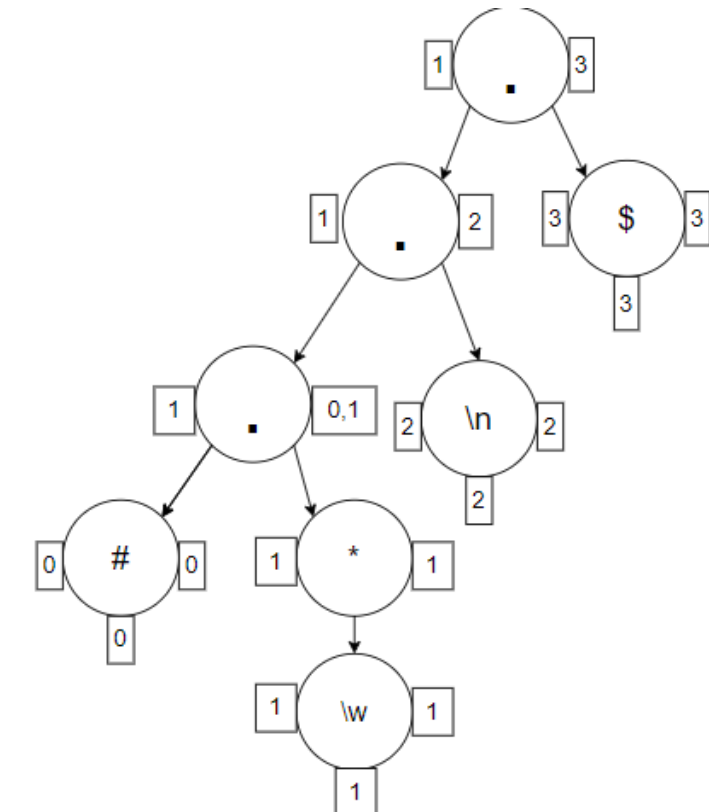
reporte	exportarReporte	Azul
llavea	{	Turquesa
llavec	}	Turquesa
corchetea	[Turquesa
corchetec]	Turquesa
puntocomas	;	Turquesa
coma	,	Turquesa
igual	=	Turquesa
parentesisas	(Turquesa
parentesiscs)	Turquesa
cadena	Todo carácter entre “	Verde
decimal	[0-9].[0-9]	Amarillo
entero	[0-9]	Amarillo
error	Todo carácter o condición no contenida en las filas anteriores	Rojo

Para la interpretación lógica del autómata programada en el sistema se efectuó el siguiente diagrama:



MÉTODO DEL ARBOL PARA “Comentario de una línea”

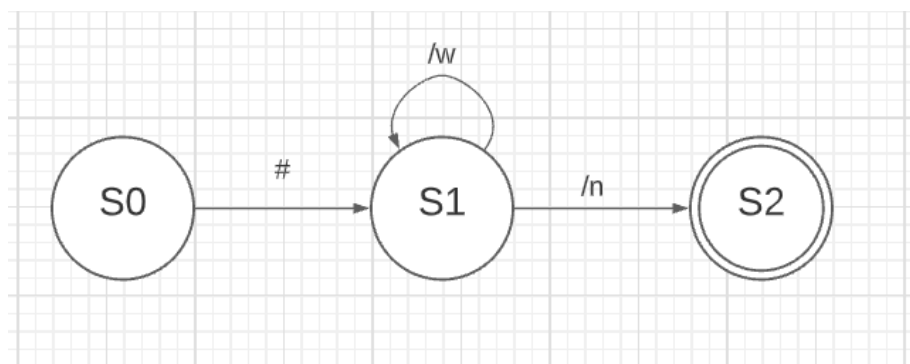
Árbol binario



Nodo	Terminal	Siguiente
0	#	1, 2
1	\w	1, 2
2	\n	3
3	\$	

Estado	#	\w	\n
S0	S1	-	-
S1	-	S1	S2
S2	-	-	-

Autómata:



MÉTODO DEL ARBOL PARA “Comentario multilinea”

Expresión regular: `[“”(\w\n)*“”]`

Árbol binario:

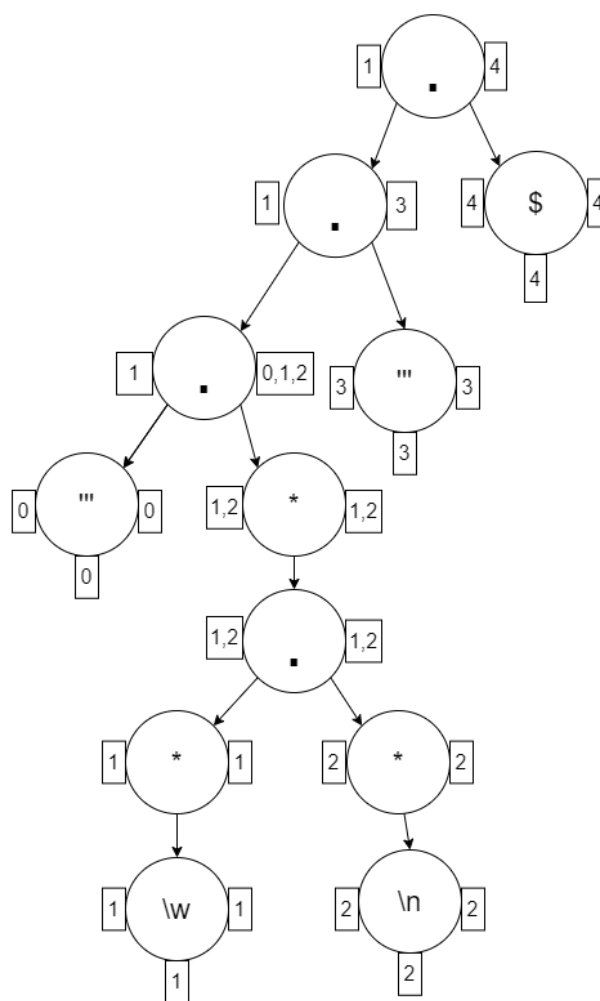


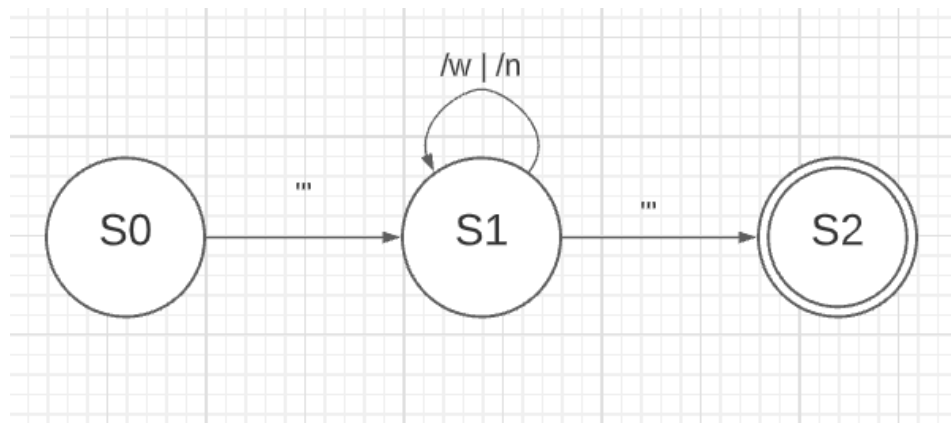
Tabla de siguientes:

Nodo	Terminal	Siguiente
0	"	1, 2, 3
1	\w	1, 2, 3
2	\n	1, 2, 3
3	"	4
4	\$	

Tabla de transición:

Estado	#	\w	\n
S0	S1	-	-
S1	-	S1	S2
S2	-	-	-

Autómata:



MÉTODO DEL ARBOL PARA “entero”

Expresión regular: $[\backslash d+]$

Árbol binario:

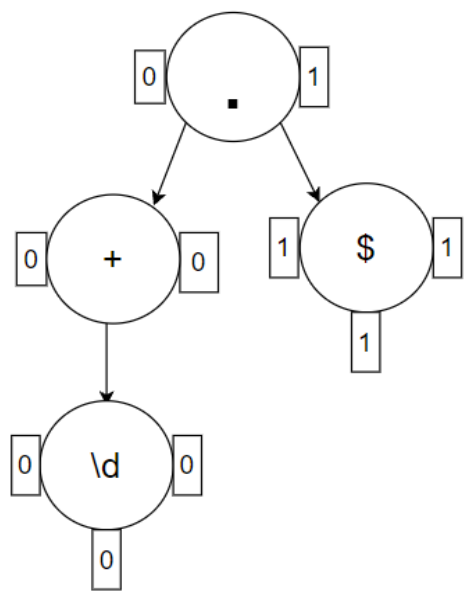


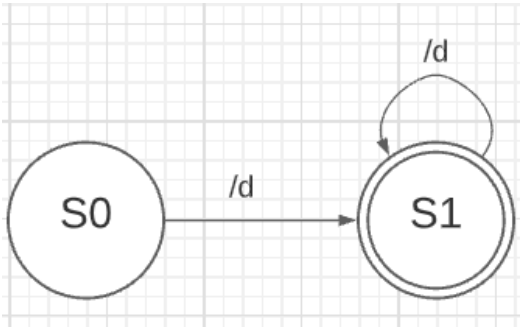
Tabla de siguientes:

Nodo	Terminal	Siguiente
0	$\backslash d$	1
1	\$	

Tabla de transición:

Estado	$\backslash d$
S0	S1
S1	-

Autómata:



MÉTODO DEL ARBOL PARA “decimales”

Expresión regular: $[(\backslash d) \cdot (\backslash d)]$

Árbol binario

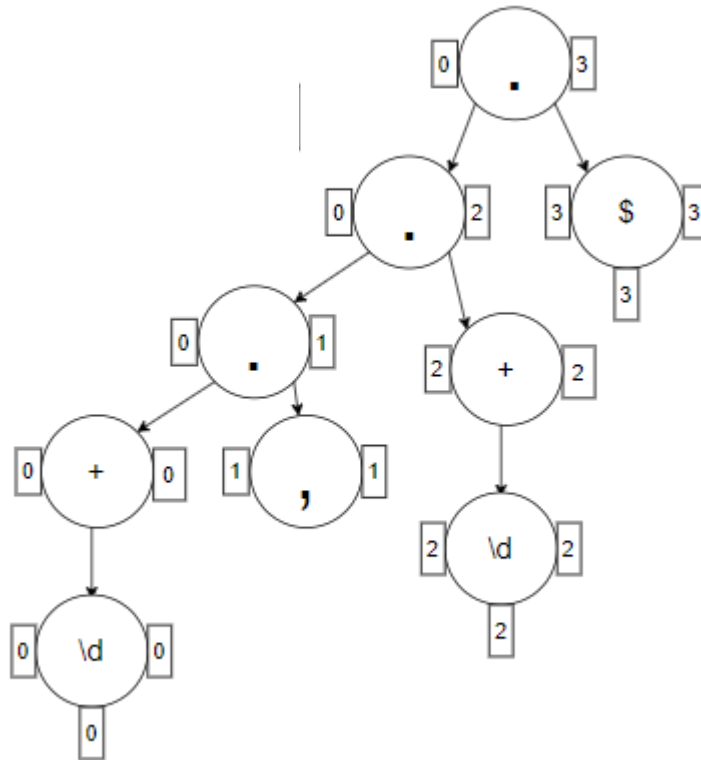


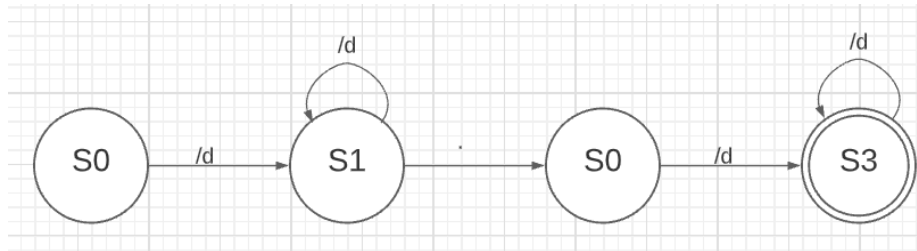
Tabla de siguientes:

Nodo	Terminal	Siguiente
0	\d	1
1	.	2
2	\d	3
3	\$	

Tabla de transición:

Estado	\d	.
S0	S1	-
S1	-	S2
S2	S3	-
S3	-	-

Autómata:



MÉTODO DEL ARBOL PARA “cadenas”

Expresión regular: $[(\backslash w \mid \backslash s \mid \backslash n)^+]$

Árbol binario

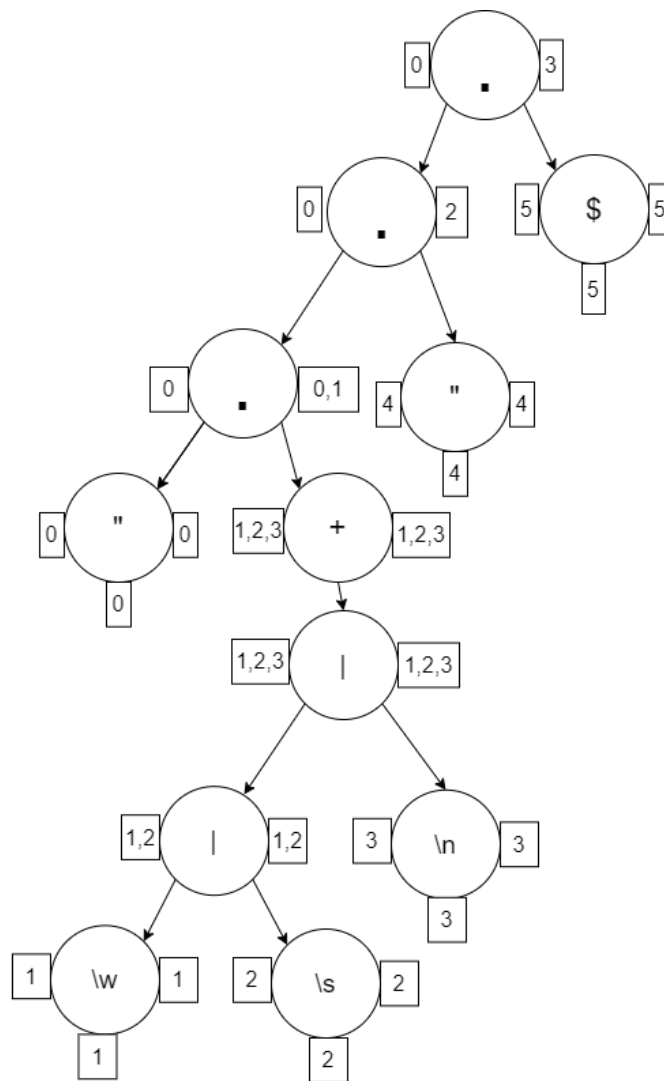


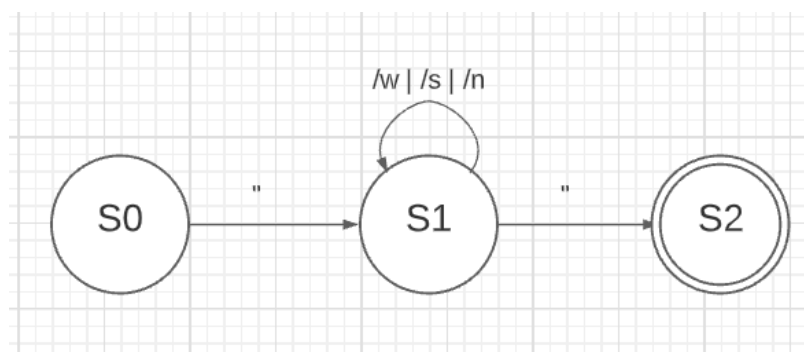
Tabla de siguientes:

Nodo	Terminal	Siguiente
0	"	1,2,3,4
1	\w	1,2,3,4
2	\s	1,2,3,4
3	\n	1,2,3,4
4	"	5
5	\$	



Tabla de transición:

Estado	"	\w	\s	\n
S0	S1	-	-	-
S1	S2	S1	S1	S1
S2	S3	-		

Autómata:



Programación de autómeta: Como se mencionó anteriormente el autómeta se creó dentro de la clase analizador léxico.

```
automata.py >  analizador_lexico >  analizar
1  from arbol import *
2  from Token import token
3  from Error import error
4  import re
5
6  class analizador_lexico:
7      def __init__(self):
8          self.tokens = []
9          self.errores = []
10
11     def analizar(self, codigo):
12         self.tokens = []
13         self.errores = []
14
15         linea = 1
16         columna = 1
17         buffer = ''
18         sentinel = '$'
19         estado = 0
20         codigo += sentinel
```

```

i = 0
while i < len(codigo):
    c = codigo[i]

    if estado == 0:
        if c == '=':
            buffer += c
            self.tokens.append(token(buffer, 'igual', linea, columna))
            buffer = ''
            columna += 1
        elif c == '{':
            buffer += c
            self.tokens.append(token(buffer, 'llavea', linea, columna))
            buffer = ''
            columna += 1
        elif c == '}':
            buffer += c
            self.tokens.append(token(buffer, 'llavec', linea, columna))
            buffer = ''
            columna += 1
        elif c == ';':
            buffer += c
            self.tokens.append(token(buffer, 'puntocoma', linea, columna))
            buffer = ''
            columna += 1
        elif c == ',':
            buffer += c
            self.tokens.append(token(buffer, 'coma', linea, columna))
            buffer = ''
```

```

elif estado == 1:
    if c == '"':
        buffer += c
        self.tokens.append(token(buffer, 'cadena', linea, columna))
        buffer = ''
        columna += 1
        estado = 0
    elif c == '\n':
        buffer += c
        linea += 1
        columna = 1
    elif c == '\r':
        buffer += c
    else:
        buffer += c
        columna += 1
elif estado == 2:
    if re.search('\d', c):
        buffer += c
        columna += 1
    elif c == '.':
        buffer += c
        columna += 1
    else:
        if buffer.find('.') != -1:
            self.tokens.append(token(buffer, 'decimal', linea, columna))
            buffer = ''
            i -= 1
            estado = 0

```

```

        estado = 0
elif estado == 3:
    if re.search('[a-zA-Z]', c):
        buffer += c
        columna += 1
    else:
        if buffer == 'Claves':
            self.tokens.append(token(buffer, 'claves', linea, columna))
        elif buffer == 'Registros':
            self.tokens.append(token(buffer, 'registros', linea, columna))
        elif buffer == 'imprimir':
            self.tokens.append(token(buffer, 'imprimir', linea, columna))
        elif buffer == 'imprimirln':
            self.tokens.append(token(buffer, 'imprimirln', linea, columna))
        elif buffer == 'conteo':
            self.tokens.append(token(buffer, 'conteo', linea, columna))
        elif buffer == 'promedio':
            self.tokens.append(token(buffer, 'promedio', linea, columna))
        elif buffer == 'contarsi':
            self.tokens.append(token(buffer, 'contarsi', linea, columna))
        elif buffer == 'datos':
            self.tokens.append(token(buffer, 'datos', linea, columna))
        elif buffer == 'sumar':
            self.tokens.append(token(buffer, 'sumar', linea, columna))
        elif buffer == 'max':
            self.tokens.append(token(buffer, 'maximo', linea, columna))
        elif buffer == 'min':
            self.tokens.append(token(buffer, 'minimo', linea, columna))

```

Analizador Sintáctico: Dentro del programa para el análisis del lenguaje se implementó un analizador sintáctico empleando la siguiente gramática independiente del contexto:

```
INICIO = LISTA_INSTRUCCIONES

LISTA_INSTRUCCIONES = INSTRUCCION LISTA_INSTRUCCIONES2

LISTA_INSTRUCCIONES2 = INSTRUCCION LISTA_INSTRUCCIONES2
                        | EOF ($)

INSTRUCCION = INS_CLAVES
              | INS_REGISTRO
              | INS_IMPRIMIR
              | INS_IMPRIMIRLN
              | INS_CONTEO
              | INS_CONTARSI
              | INS_PROMEDIO
              | INS_DATOS
              | INS_SUMAR
              | INS_MAXIMO
              | INS_MINIMO
              | INS_REPORTEE

INS_REGISTRO = registros igual corchetea LISTA_REGISTROS corchetec

LISTA_REGISTROS = REGISTRO LISTA_REGISTROS2

LISTA_REGISTROS2 = REGISTRO LISTA_REGISTROS2
                  | EPSILON (], corchetec)

REGISTRO = llavea LISTA_VAL_REG llavec

LISTA_VAL_REG = VAL_REG LISTA_VAL_REG2

LISTA_VAL_REG2 = VAL_REG LISTA_VAL_REG2
                | EPSILON (}, llavec)

VAL_REG = cadena
          | entero
          | decimal

INS_CLAVES = claves igual corchetea LISTA_CLAVES corchetec

LISTA_CLAVES = VAL_CLS LISTA_CLAVES2

LISTA_CLAVES2 = coma VAL_CLS LISTA_CLAVES2

VAL_CLS = cadena
|

INS_IMPRIMIR = imprimir parentesisA cadena parentesisC puntocomA

INS_IMPRIMIRLN = imprimirln parentesisA cadena parentesisC puntocomA

INS_DATOS = datos parentesisA parentesisC puntocomA

INS_CONTEO = datos parentesisA parentesisC PuntoyComa

INS_PROMEDIO = promedio parentesisA cadena parentesisC puntocomA

INS_MAX = maximo parentesisA cadena parentesisC puntocomA

INS_MIN = minimo parentesisA cadena parentesisC puntocomA

INS_SUMAR = sumar parentesisA cadena parentesisC puntocomA

INS_REPORTE = reporte parentesisA cadena parentesisC puntocomA

INS_CONTAR_SI = contarsi parentesisA cadena coma entero parentesisC puntocomA
```

Programación de la gramática: Para el análisis se procedió a programar dos clases una llamada Analizador Sintáctico la cual emplea la recursión para entrar en los diferentes estados no terminales, a su vez se creó un paquete que contiene múltiples clases el cual se nombró árbol, se creó una clase por cada método recursivo en la clase Analizador Sintáctico y a su vez un método mas que funciona para los estados terminales.

```
class analizador_sintactico:
    def __init__(self):
        self.tokens = []
        self.erroros = []
        self.i = 0

> def val_reg(self): ...
> def lista_val_reg2(self): ...
> def lista_val_reg(self): ...
> def registro(self): ...
> def lista_registros2(self): ...
> def lista_registros(self): ...
> def ins_registros(self): ...
> def val_cls(self): ...
> def lista_claves2(self): ...
> def lista_claves(self): ...
> def ins_claves(self): ...

18
19 > class expresion: ...
42
43 > class IntruccionPromedio() : ...
89
90 > class IntruccionContarsi() : ...
42
43 > class IntruccionDatos() : ...
00
01 > class IntruccionSumar() : ...
45
46 > class IntruccionMax() : ...
89
90 > class IntruccionMin() : ...
33
34 > class IntruccionReporte() : ...
32
33 > class IntruccionConteo() : ...
67
68 > class IntruccionClaves() : ...
06
07 > class ListaClaves2() : ...
38
39 > class ListaClaves() : ...
70
71 > class ListaValReg2() : ...
02
03 > class ListaValReg() : ...
41
42 > class ListaRegistros2() : ...
```

Árbol de derivación: Como toda gramática se puede representar mediante un árbol de derivación este procede a crearse mediante la herramienta Graphviz la cual se implementaron los métodos getNodos en cada clase del paquete árbol.

```
dot = Graph('arbol', 'png')
dot.format = 'pdf'
dot.attr(splines = 'false')
dot.node_attr.update(shape = 'circle')
dot.edge_attr.update(color = 'black')
i = 0
def start():
    global i
    i += 1
    return i

def getNodos(self):
    global dot
    idnodo = str(start())
    dot.node(idnodo, 'expresion')

    idlit = str(start())
    dot.node(idlit, 'literal')

    idexp = str(start())
    dot.node(idexp, self.valor)

    dot.edge(idlit, idexp)
    dot.edge(idnodo, idlit)

    return idnodo
```

Generación de reportes html: Para la creación de los reportes se utilizarán 2 métodos img_html el cual se encarga de generar los reportes de las imágenes generadas a partir de las celdas contenidas en el archivo de entrada.

```
def img_html(self,imagenes):
    file = open('imagenes.html','w')
    contenido='<!DOCTYPE html>'
    <html lang="en">
    <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>BITXELART</title>
    </head>
    <body>'''
    for img in imagenes:
        tamaño=math.sqrt((img.getAncho()*img.getAlto()))/(img.getFilas()*img.getColumnas())
        contenido+='\n<h1 style="text-align: center;">'+img.getNombre()+'\n</h1>'
        contenido+='\n<table style="height:'+str(img.getAlto())+'px; width: '+str(img.getAncho())+'px; margin-
        aux=img.getCeldas().eRows.primerO
        while aux != None:
            actual=aux.acceso
            contenido+='\n<tr>'
            while actual != None:
                if actual.valor.getBl():
                    contenido+='\n<td style="background-color: '+actual.valor.getColor()+'; height: '+str(tama
                else:
                    contenido+='\n<td> </td>'
            actual=actual.derecha
        contenido+='\n</tr>'
        contenido+='\n</table>'
    file.write(contenido)
    file.close()
```

El segundo es para la creación de los reportes de tokens y errores el cual se encuentra dentro de la clase analizador como html_T para los tokens y html_E para los errores. Los cuales tienen el mismo funcionamiento que el método img_html.

```
> def html_T(self): ...
> def html_E(self): ...
```