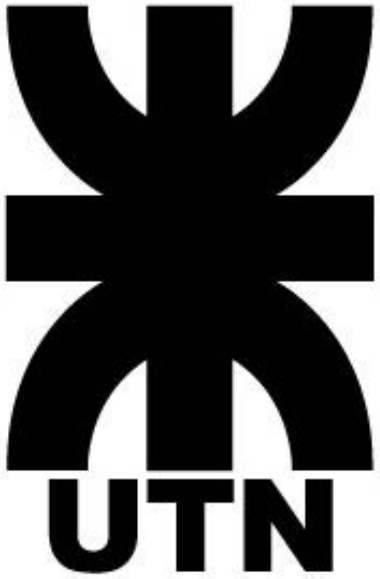

SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

TRABAJO PRÁCTICO INTEGRADOR

Diseño e implementación de Lexer y Parser y Traductor de Lenguaje RSS

Grupo: N.º 4

Integrantes:

- AGUIRRE, Camilo
- BIANCIOTTO, Joaquín
- COLOMBO, Matías Julián
- MARAIN, Yoel Mario

Carrera: Ingeniería en Sistemas de Información

Comisión: ISI A

Primer Cuatrimestre

Curso Académico: 2023

UNIVERSIDAD TECNOLOGICA NACIONAL

FACULTAD REGIONAL RESISTENCIA

Fecha y Lugar de presentación: 04/06/2023. Resistencia, Chaco

ÍNDICE

ELABORACIÓN DE GRAMÁTICA

ANÁLISIS LÉXICO

1. INTRODUCCION:

En esta primera entrega del Trabajo Practico Integrador se presenta la gramática del software desarrollada.

El objetivo de este trabajo es construir un analizador léxico y sintáctico que permita analizar, validar y transformar un subconjunto de etiquetas de la especificación DocBook/XML para la descripción de artículos. La utilidad construida recibe un archivo en formato XML y contenido Docbook y deberá indicar si está bien construido (adecuado al estándar, sin errores) de otra manera indicar los errores; adicionalmente a medida que analiza el documento deberá transformar el contenido en un documento HTML.

Para poder llevar a cabo el trabajo, primero que nada, debemos entender que es Docbook y que características posee.

DocBook es un lenguaje de marcado semántico para documentación técnica. Originalmente estaba destinado a escribir documentos técnicos relacionados con el hardware y el software de la computadora, pero puede usarse para cualquier otro tipo de documentación.
Como lenguaje semántico, DocBook permite a sus usuarios crear contenido de documentos en una forma de presentación neutral que captura la estructura lógica del contenido; ese contenido se puede publicar en una variedad de formatos, incluidos HTML, XHTML, EPUB, PDF, páginas de manual, ayuda web y ayuda HTML, sin necesidad de que los usuarios realicen ningún cambio en la fuente. En otras palabras, cuando un documento se escribe en formato DocBook, se puede transferir fácilmente a otros formatos, en lugar de tener que volver a escribirlo.

Algunos de los principales usos de Docbook son:

- Libros para publicación impresa
- Mantenimiento de sitios web
- Sitios web de preguntas frecuentes
- Documentación informática
- Producción de diapositivas de presentación
- Producir documentación generada a partir de comentarios de código

2. GRAMÁTICA:

2.1 Símbolos de la gramática:

```
Símbolo sentencia = S

No terminales = {SEC, SSEC, INFO, ABSTRACT ADRESS, AUTHOR, COPY, TITLE, SPARA,
EMPHASIS, COMMENT, LINK, EMAIL, MOBJ, FIRSTNAME, SURNAME, STREET, CITY, STATE,
PHONE, DATE, YEAR, HOLDER, IOBJ, IDATA, VOBJ, VDATA, ITEMLIST, LITEM, INFTABLE,
TGROUP, THEAD, TFOOD, TBODY, ROW, ENTRY, A, B, C, D, E, F, G, H, I, J, K, L, M,
N, O, P, Q, R, T, U, V, X, Y, Z }

Terminales= {#texto, #url, <, >, /, =, ", !DOCTYPE, article, section, simplesec,
info, abstract, address, author, copyright, title, simpara, emphasis, comment,
link, xlink:href, para, important, firstname, surname, street, city, state,
phone, email, date, year, holder, mediaobject, imageobject, imagedata, fileref,
videoobject, videodata, itemizedlist, listitem, informaltable, tgroup, thead,
tfood, tbody, row, entry, entrybl}
```

2.2 Producciones:

Etiquetas estructurales del documento

```
S → <!DOCTYPE article><article> Y X Z </article>
S → <!DOCTYPE article><article> Y X </article>
S → <!DOCTYPE article><article>  X Z </article>
S → <!DOCTYPE article><article>  X </article>
Y → INFO TITLE | INFO | TITLE
X → ITEMLIST X | IMPORTANT X | PARA X | SPARA X | ADDRESS X | MOBJ X | INFTABLE
X | COMMENT X | ABSTRACT X
X → ITEMLIST | IMPORTANT | PARA | SPARA | ADDRESS | MOBJ | INFTABLE | COMMENT |
ABSTRACT
Z → SEC Z | SSEC Z
Z → SEC|SSEC
SEC → <section>Y X Z</section> | <section>Y X</section> | <section>X Z</section>
| <section>X </section>
SSEC → <simplesec>Y X</simplesec> | <simplesec>X</simplesec>
```

Etiquetas básicas de párrafo

```
INFO → <info>A</info>
A → MOBJ A | ABSTRACT A | ADDRESS A | AUTHOR A | DATE A | COPY A | TITLE A |
A → MOBJ | ABSTRACT | ADDRESS | AUTHOR | DATE | COPY | TITLE
ABSTRACT →<abstract>TITLE B</abstract> | <abstract> B</abstract>
B → PARA B | SPARA B
B → PARA | SPARA
ADDRESS →  <address>C</address>
C → #texto C | STREET C | CITY C | STATE C | PHONE C | EMAIL C
C → #texto | STREET | CITY | STATE | PHONE | EMAIL
```

```
AUTHOR→ <author>D</author>
D → FNAME D| SNAME D
D → FNAME | SNAME
COPY → <copyright>EF</copyright>| <copyright>E</copyright>
E → YEAR E | YEAR
F → HOLDER F | HOLDER
TITLE→ <title>G</title>
G → #texto G | EMPHASIS G | LINK G | EMAIL G
G → #texto | EMPHASIS | LINK | EMAIL |
SPARA → <simpara > H </simpara>
EMPHASIS → < emphasis> H </emphasis>
COMMENT → <comment> H </comment>
LINK → <link U> H </link>
U → xlink:href=”#url”
H → #texto H | EMPHASIS H | LINK H | EMAIL H | AUTHOR H | COMMENT H
H → #texto | EMPHASIS | LINK | EMAIL | AUTHOR | COMMENT
PARA → <para>I</para>
I → #texto I | EMPHASIS I | LINK I | EMAIL I | AUTHOR I | COMMENT I | ITEMLIST I
| IMPORTANT I | ADDRESS I | MOBJ I | INFTABLE I |
I → #texto |EMPHASIS | LINK | EMAIL | AUTHOR | COMMENT | ITEMLIST | IMPORTANT|
ADDRESS | MOB | INFTABLE
IMPORTANT → <important>TITLE J </important>| <important> J </important>
J → ITEMLIST J | IMPORTANT J | PARA J | SPARA J | ADDRESS J | MOBJ J | INFTABLE
J | COMMENT J | ABSTRACT J |
J → ITEMLIST  | IMPORTANT   | PARA   | SPARA  | ADDRESS  | MOBJ  | INFTABLE  |
COMMENT  | ABSTRACT  |
FIRSTNAME → <firstname> K </firstname>
SURNAME → <surname> K </surname>
STREET → <street> K </street>
CITY → <city> K </city>
STATE → <state> K </state>
PHONE → <phone> K </phone>
EMAIL → <email> K </email>
DATE → <date> K </date>
YEAR → <year> K </year>
HOLDER → <holder> K </holder>
K → #texto K | LINK K | EMPHASIS K | COMMENT K
K → #texto | LINK | EMPHASIS | COMMENT
```

Imágenes y multimedia

```
MOBJ → <mediaobject> INFO L </mediaobject> | <mediaobject> L </mediaobject>
L → VOBJ L | IOBJ L
L → VOBJ | IOBJ
IOBJ → <imageobject> INFO IDATA </imageobject>
IOBJ → <imageobject> IDATA </imageobject>
IDATA → <imagedata fileref=”#url”/>
VOBJ → <videoobject> INFO VDATA </videoobject>
VOBJ → <videoobject> VDATA </videoobject>
VDATA → <videodata fileref=”#url”/>
```

Listas

```
ITEMLIST → <itemizedlist> M </itemizedlist>
M → LITEM M | LITEM
LITEM → <listitem> J </listitem>
```

Tablas

```
INFTABLE → <informaltable> N </informaltable>
N → MOBJ N | TGROUP N
N → MOBJ | TGROUP
TGROUP → <tgroup> O </tgroup>
O → THEAD TFOOD TBODY | TFOOD TBODY | THEAD TBODY
THEAD → <thead> P </thead>
TFOOD → <tfood> P </tfood>
TBODY → <tbody> P </tbody>
P → ROW P | ROW
ROW → <row>  Q </row>
Q → ENTRY Q | ENTRYTBL Q
Q → ENTRY | ENTRYTBL
ENTRY → <entry>  R </entry>
R → ITEMLIST R | IMPORTANT R | PARA R | SPARA R | #texto R | MOBJ R | COMMENT R
| ABSTRACT R
R → ITEMLIST | IMPORTANT| PARA| SPARA | #texto| MOBJ| COMMENT | ABSTRACT
ENTRYTBL → <entrytbl> T </entrytbl>
T → THEAD TBODY | TBODY
```

3. WEBGRAFIA:

- https://wiki.archlinux.org/title/DocBook
- https://web.archive.org/web/20120123183312/http://www.dpawson.co.uk/docbook/reference.html

1. INTRODUCCION:

Un analizador léxico (o *lexer*) es una parte esencial de un compilador o intérprete que se encarga de descomponer el código fuente en una secuencia de elementos más pequeños llamados *tokens*. Estos tokens son unidades léxicas que representan los componentes individuales del lenguaje de programación, como palabras clave, identificadores, operadores, números y símbolos.

El lexer toma el código fuente como entrada y realiza un escaneo caracter por caracter, identificando y clasificando los diferentes elementos léxicos. Utiliza reglas definidas previamente para reconocer patrones y formar los tokens correspondientes.

Para la realización de este trabajo, optamos por utilizar **Python** debido a las siguientes razones:

+ **Sintaxis clara y legible:** Python se destaca por su sintaxis simple y fácil de leer, lo que facilita la comprensión y escritura de código, manteniendo también un código más limpio y organizado.
+ **Aprendizaje eficiente:** Python tiene una curva de aprendizaje suave y cuenta con una gran comunidad, lo que facilita la obtención de recursos de aprendizaje en línea y documentación clara.
+ **Amplia disponibilidad de bibliotecas y módulos:** Python cuenta con una gran cantidad de bibliotecas y módulos disponibles que facilitan la tarea de implementar funcionalidades avanzadas.

Llevamos a cabo el proyecto en una plataforma web de desarrollo colaborativo llamada **GitHub**, esta proporciona control automático de versiones, lo que permite realizar un seguimiento de los cambios realizados en el proyecto a lo largo del tiempo, facilita la colaboración en equipo, ofrece herramientas de seguimiento de problemas y solicitudes de extracción.

2. LEXER:

2.1 Módulos utilizados:

PLY (Python Lex-Yacc) es una biblioteca de análisis léxico y sintáctico. Proporciona las herramientas necesarias para construir analizadores personalizados basados en las técnicas de análisis léxico y sintáctico LEX y YACC utilizadas tradicionalmente en otros lenguajes. Sin embargo, a diferencia de LEX y YACC, que están escritos en C, PLY está escrito en Python y aprovecha las características del lenguaje y la facilidad de uso que ofrece.

Nosotros utilizamos el módulo 'ply.lex', el cual proporciona herramientas necesarias para definir y ejecutar reglas de análisis léxico, es decir, para reconocer tokens en un flujo de texto.

Para esto, lo importamos, de la siguiente forma:

```
import ply.lex as lex
```

Además, importamos el módulo 're' para realizar coincidencias de expresiones regulares.

```
import re
```

El módulo 'codecs', para trabajar con codificaciones de caracteres.

```
import codecs
```

El módulo 'os', para realizar operaciones relacionadas con el sistema operativo.

```
import os
```

Y el módulo 'sys', para acceder a funcionalidades específicas del intérprete de Python.

```
import sys
```

2.2 Listas:

Definimos 2 listas importantes, la primera será una lista vacía llamada 'error_caracter_ilegal' para almacenar caracteres ilegales encontrados durante el análisis léxico.

```
error_caracter_ilegal=[]
```

Además, definimos una lista llamada 'tokens' que contiene los nombres de los tokens reconocidos por el analizador léxico.

2.2.1 Definición de tokens:

```
tokens = [ 'DT1','DT2','APERTURA_ARTICLE', 'CIERRE_ARTICLE' , 'APERTURA_PARA',
'CIERRE_PARA', 'TEXTO', 'APERTURA_INFO' , 'CIERRE_INFO' , 'APERTURA_TITLE' ,
'CIERRE_TITLE' , 'APERTURA_ITEMIZEDLIST', 'CIERRE_ITEMIZEDLIST',
'APERTURA_IMPORTANT' , 'CIERRE_IMPORTANT' , 'APERTURA_SIMPARA', 'CIERRE_SIMPARA'
, 'APERTURA_ADDRESS' , 'CIERRE_ADDRESS' , 'APERTURA_MEDIAOBJECT' ,
'CIERRE_MEDIAOBJECT' , 'APERTURA_INFORMALTABLE' , 'CIERRE_INFORMALTABLE' ,
'APERTURA_COMMENT' , 'CIERRE_COMMENT' , 'APERTURA_ABSTRACT' , 'CIERRE_ABSTRACT'
, 'APERTURA_SECTION' , 'CIERRE_SECTION' , 'APERTURA_SIMPLESECT' ,
'CIERRE_SIMPLESECT' , 'APERTURA_EMPHASIS' , 'CIERRE_EMPHASIS' , 'APERTURA_LINK'
, 'CIERRE_LINK' ,  'APERTURA_FIRSTNAME' , 'CIERRE_FIRSTNAME' ,
'APERTURA_SURNAME' , 'CIERRE_SURNAME' , 'APERTURA_STREET' , 'CIERRE_STREET' ,
'APERTURA_CITY' , 'CIERRE_CITY' , 'APERTURA_STATE' , 'CIERRE_STATE' ,
```

```
'APERTURA_PHONE' , 'CIERRE_PHONE' , 'APERTURA_EMAIL' , 'CIERRE_EMAIL' ,
'APERTURA_DATE' , 'CIERRE_DATE' , 'APERTURA_YEAR' , 'CIERRE_YEAR' ,
'APERTURA_HOLDER' , 'CIERRE_HOLDER', 'APERTURA_IMAGEDATA' ,
'APERTURA_VIDEOOBJECT' , 'CIERRE_VIDEOOBJECT' , 'APERTURA_IMAGENOBJECT' ,
'CIERRE_IMAGENOBJECT' , 'APERTURA_VIDEODATA', 'APERTURA_LISTITEM' ,
'CIERRE_LISTITEM' , 'APERTURA_TGROUP' , 'CIERRE_TGROUP' , 'APERTURA_THEAD' ,
'CIERRE_THEAD' , 'APERTURA_TFOOT' , 'CIERRE_TFOOT' , 'APERTURA_TBODY' ,
'CIERRE_TBODY' , 'APERTURA_ROW' , 'CIERRE_ROW' , 'APERTURA_ENTRY' ,
'CIERRE_ENTRY' , 'APERTURA_ENTRYTBL' , 'CIERRE_ENTRYTBL' , 'APERTURA_AUTHOR' ,
'CIERRE_AUTHOR' , 'ERROR_1' , 'ERROR_2' , 'ERROR_3' , 'newline' ]
```

2.3 Expresiones regulares:

Ahora planteamos las expresiones regulares de cada token definido en la lista anterior mediante funciones. Estas funciones se ejecutan cuando se encuentra una coincidenciay las utilizamos para realizar operaciones de análisis léxico.

```
t_APERTURA_ARTICLE = r'<article>'
t_CIERRE_ARTICLE = r'</article>'
t_APERTURA_SIMPARA = r'<simpara>'
t_CIERRE_SIMPARA = r'</simpara>'
t_APERTURA_ADDRESS = r'<address>'
t_CIERRE_ADDRESS = r'</address>'
t_APERTURA_MEDIAOBJECT = r'<mediaobject>'
t_CIERRE_MEDIAOBJECT = r'</mediaobject>'
t_APERTURA_COMMENT = r'<comment>'
t_CIERRE_COMMENT = r'</comment>'
t_APERTURA_ABSTRACT = r'<abstract>'
t_CIERRE_ABSTRACT = r'</abstract>'
t_APERTURA_SECTION = r'<section>'
t_CIERRE_SECTION = r'</section>'
t_APERTURA_SIMPLESECT = r'<simplesect>'
t_CIERRE_SIMPLESECT = r'</simplesect>'
t_APERTURA_EMPHASIS = r'<emphasis>'
t_CIERRE_EMPHASIS = r'</emphasis>'
t_APERTURA_AUTHOR = r'<author>'
t_CIERRE_AUTHOR = r'</author>'
t_APERTURA_FIRSTNAME = r'<firstname>'
t_CIERRE_FIRSTNAME = r'</firstname>'
t_APERTURA_SURNAME = r'<surname>'
t_CIERRE_SURNAME = r'</surname>'
t_APERTURA_STREET = r'<street>'
t_CIERRE_STREET = r'</street>'
t_APERTURA_CITY = r'<city>'
t_CIERRE_CITY = r'</city>'
t_APERTURA_STATE = r'<state>'
t_CIERRE_STATE = r'</state>'
t_APERTURA_PHONE = r'<phone>'
t_CIERRE_PHONE = r'</phone>'
t_APERTURA_EMAIL = r'<email>'
t_CIERRE_EMAIL = r'</email>'
t_APERTURA_DATE = r'<date>'
t_CIERRE_DATE = r'</date>'
t_APERTURA_YEAR = r'<year>'
t_CIERRE_YEAR = r'</year>'
```

```
t_APERTURA_HOLDER = r'<holder>'
t_CIERRE_HOLDER = r'</holder>'
t_APERTURA_VIDEOOBJECT = r'<videoobject>'
t_CIERRE_VIDEOOBJECT = r'</videoobject>'
t_APERTURA_IMAGENOBJECT = r'<imagenobject>'
t_CIERRE_IMAGENOBJECT = r'</imagenobject>'
t_APERTURA_TGROUP = r'<tgroup>'
t_CIERRE_TGROUP = r'</tgroup>'
t_APERTURA_THEAD = r'<thead>'
t_CIERRE_THEAD = r'</thead>'
t_APERTURA_TFOOT = r'<tfood>'
t_CIERRE_TFOOT = r'</tfood>'
t_APERTURA_TBODY = r'<tbody>'
t_CIERRE_TBODY = r'</tbody>'
t_APERTURA_ENTRYTBL = r'<entrytbl>'
t_CIERRE_ENTRYTBL = r'</entrytbl>'
t_ERROR_1 = r'<[\w]+>'
t_ERROR_2 = r'<[\w]+\s[\w]+=*[\w"]+\s*/*>'
t_ERROR_3 = r'</[\w]+>'
```

2.4 Funciones:

Definimos una función 't_ignore' que especifica los caracteres que deben ser ignorados por el lexer, como espacios en blanco o tabulaciones.

```
t_ignore = ' \t'
```

Definimos una función 't_error' que maneja los errores de caracteres no reconocidos.

```
def t_error(t):
  print("caracter ilegal %s" % t.value[0])
  t.lexer.skip(1)
```

Definimos una función 't_newline' que cuenta el número de nuevas líneas en el código fuente para realizar un seguimiento de los números de línea.

```
def t_newline(t):
  r'\n+'
  t.lexer.lineno += len(t.value)
```

Creamos el lexer llamando a la función lex.lex(). Esto inicializa el lexer con las reglas y funciones definidas previamente.

```
lexer = lex.lex()
```

3. Conversión a HTML:

Otra de las funcionalidades que tiene nuestro trabajo es la de traducir el documento, generando un archivo de texto HTML, transformando algunas etiquetas XML en etiquetas HTML.

Apertura de archivo:

Abrimos un archivo llamado 'archivo.html' en modo escritura y codificado en UTF-8 para almacenar el resultado del análisis léxico.

```
arch= open("src/html_generados/archivo.html","w",encoding="utf-8")
```

Funciones de conversión:

```
def t_DT1(t):
      r'<[!]DOCTYPE\sarticle>'
      arch.write("<!DOCTYPE html>")
def t_TEXTO (t):
    r'[\w._%+?¿¡!()"|°¬$&={}\#@*-]+'  #falta ver caracteres especiales
    arch.write(f'{t.value} ')
    return (t)
def t_APERTURA_PARA(t):
      r'<para>'
      arch.write("<p>")
      return(t)
def t_CIERRE_PARA(t):
      r'</para>'
      arch.write("</p>")
      return(t)
def t_APERTURA_TITLE(t):
      r'<title>'
      arch.write("<h1>")
      return(t)
def t_CIERRE_TITLE(t):
      r'</title>'
      arch.write("</h1>")
      return(t)
def t_APERTURA_INFO(t):
      r'<info>'
      arch.write('<div style="color:white;background-color:green;font-
size:8pts"><p>')
      return(t)
def t_CIERRE_INFO(t):
      r'</info>'
      arch.write('</p></div>')
      return(t)
def t_APERTURA_IMPORTANT(t):
      r'<important>'
      arch.write('<div style="background-color:red;color:white">')
      return(t)
def t_CIERRE_IMPORTANT(t):
      r'</important>'
      arch.write('</div>')
      return(t)
def t_APERTURA_IMAGEDATA (t):
```

```
      r'<imagedata\s+fileref=""[(http(s)?|ftp(s)?):\/\/(www\.)?a-zA-Z0-
9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&//=]*)"\s*[/]>'

      return(t)

def t_APERTURA_VIDEODATA (t):

      r'<videodata\s+fileref="[(http(s)?|ftp(s)?):\/\/(www\.)?a-zA-Z0-
9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&//=]*)"\s*[/]>'

      return(t)

def t_APERTURA_LINK (t):

      r'<link\s+xlink:href ="[(http(s)?|ftp(s)?):\/\/(www\.)?a-zA-Z0-
9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&//=]*)"\s*[/]>'

      arch.write(f'<a href="{t.value}">esto es un link</a>')

      return (t)
def t_APERTURA_INFORMALTABLE(t):
      r'<informaltable>'
      arch.write("<table>")
      return (t)
def t_CIERRE_INFORMALTABLE(t):
      r'</informaltable>'
      arch.write("</table>")
      return (t)
def t_APERTURA_ROW(t):
      r'<row>'
      arch.write("<tr>")
      return (t)
def t_CIERRE_ROW(t):
      r'</row>'
      arch.write("</tr>")
      return (t)
def t_APERTURA_ENTRY(t):
      r'<entry>'
      arch.write("<td>")
      return (t)
def t_CIERRE_ENTRY(t):
      r'</entry>'
      arch.write("</td>")
      return (t)
def t_APERTURA_ITEMIZEDLIST(t):
      r'<itemizedlist>'
      arch.write("<ul>")
      return (t)
def t_CIERRE_ITEMIZEDLIST(t):
      r'</itemizedlist>'
      arch.write("</ul>")
      return (t)
def t_APERTURA_LISTITEM(t):
      r'<listitem>'
      arch.write("<il>")
      return (t)
def t_CIERRE_LISTITEM(t):
      r'</listitem>'
```

```
        arch.write("</il>")
        return (t)
```

4. Implementación:

Ahora podemos utilizar el Lexer creado para analizar texto.

Para esto, lo primero que debemos hacer es importar nuestro lexer:

```
from lexer import lexer
```

También importaremos los módulos 'os' y 'sys', que proporcionan funciones relacionadas con el sistema operativo y la funcionalidad del intérprete de Python, respectivamente.

```
import os
import sys
```

E importaremos el objeto ‘arch’ del módulo ‘lexer’.

```
from lexer import arch
```

Creamos una función llamada 'borrarPantalla', que se utiliza para borrar la salida en la pantalla. La implementación depende del sistema operativo en el que se está ejecutando el programa. En sistemas tipo Unix (como Linux y macOS), se usa el comando **clear** para borrar la pantalla. En sistemas Windows, se usa el comando **cls**.

```
def borrarPantalla():
    if os.name == "posix":
        os.system("clear")
    elif os.name == "ce" or os.name == "nt" or os.name == "dos":
        os.system("cls")
```

Y vamos a darle al usuario la opción de elegir si desea ingresar datos manualmente o si desea analizar un archivo:

```
print("Hola este es el analizador Lexico")
print("1 para ingresar datos a mano\n2 si quiere cargar datos desde un
archivo\n")
op = input()
errores = []
```

4.1 Ingreso manual:

Si el usuario elige la opción "1", se realiza un ingreso manual de datos. El programa solicita al usuario que ingrese el texto a analizar. Luego, se pasa la cadena de entrada al analizador léxico lexer y se generan los tokens correspondientes. Si se detecta un error léxico, se muestra un mensaje de error. Después de cada análisis, se le pregunta al usuario si desea continuar o terminar el programa.

```
if op == "1":
    borrarPantalla()
    while True:
        print("ingrese lo que quiere analizar")
        cadena = input()
        lexer.input(cadena)
        while True:
            tok = lexer.token()
            if not tok:
                break
```

```
            if tok.type == "ERROR_1" or tok.type == "ERROR_2" or tok.type ==
"ERROR_3":
                print(f"error lexico en linea {tok.lineno} ({tok.value})")
            else:
                print(tok)
        print("desea continuar?\n1 para continuar\n0 para terminar")
        eleccion = input()
        if eleccion == "0":
            break
        borrarPantalla()
```

4.2 Ingreso por archivo:

Si el usuario elige la opción "2", el programa realiza un ingreso de datos a través de un archivo. El programa muestra una lista de archivos en el directorio "prueba/" y pide al usuario que elija un archivo para leer. Si el archivo seleccionado es válido, se abre y su contenido se pasa al analizador léxico lexer. Se generan los tokens correspondientes y se muestra la salida. Si se encuentra un error léxico, se muestra un mensaje de error. Finalmente, se realiza un proceso de manipulación de archivos y se pausa la ejecución del programa.

```
elif op == "2":
      n = 0
      dir = 'prueba/'                   #elegir el archivo
      with os.scandir(dir) as ficheros:
            print(type(ficheros))
            ficheros = [fichero.name for fichero in ficheros if
fichero.is_file()]    #ficheros es una lista con los archivos de la carpeta
prueba
      for j in ficheros:
            if ".txt" in j:
                  ficheros.remove(f"{j}")
      for i in ficheros:

            print(f"{n+1}: {ficheros[n]}")
            n +=1
      print("elegi el archivo para leer")
      op2 = input()
      borrarPantalla()
      if int(op2) <= n:
            ruta = ficheros[int(op2)-1]
            with open(f"prueba/{ruta}","r",encoding="utf-8") as maestro: #esto
ya funciona para cualquier fichero en prueba/
                  print(f"abierto archivo: {ruta}")
                  lexer.input(maestro.read())
                  while True:
                        tok = lexer.token()

                        if not tok:break
                        if tok.type == "error":
                              errores.append(tok.value)
                        if tok.type == "ERROR_1" or tok.type == "ERROR_2" or
tok.type == "ERROR_3":
                              print(f"error lexico en linea {tok.lineno}
({tok.value})")
                        else:
```

```
                              print(tok)
                  cambio = ruta.replace(".xml","")
                  arch.close()
                  with os.scandir('src/html_generados/') as htmls:
                        for k in htmls:
                              if k.name == f"{cambio}.html":

os.remove(f"src/html_generados/{cambio}.html")

os.rename("src/html_generados/archivo.html",f"src/html_generados/{cambio}.html"
)
                  os.system("pause")
      else:
            print("numero invalido")
            os.system("pause")
```

Si ninguna de las opciones anteriores es seleccionada, se muestra el mensaje "volve a empezar".

```
else:
    print("volve a empezar")
```

5. Historial de cambios:

1) Mayo 22
- Primera prueba de código generador de Lexer

2) Mayo 24
- Adición de etiquetas básicas de párrafo

3) Mayo 26
- Creación de carpetas finales del trabajo
- Implementación de lectura de archivos xml
- Interactividad con el usuario
- Creación de archivo de prueba
- Primer planteo de conversión a html
- Creación de archivo html con el mismo nombre que el xml
- Adición de nuevas etiquetas

4) Mayo 27
- Traducciones de etiquetas de links, tablas y listas
- Creación de carpetas para los archivos html
- Arreglo del problema que había al abrir archivos
- Primera prueba de las tablas
- Se arregló la generación de archivos html
- Se generaron más archivos de prueba

5) Mayo 28
- Se añadieron archivos de error

6) Mayo 31
- Modificación de la expresión regular de las URL

7) Junio 3
- Implementación de control de errores de escritura
- Implementación de control de línea en la que aparece un token
- Adición de etiquetas
- Implementación de control de caracteres especiales
- Adición de los errores correspondientes a los nuevos controles
- Cambios estéticos
- Implementación de función para borrar la salida en la pantalla
- Arreglo de errores correspondientes a la apertura de los archivos

8) Junio 4
- Cambios concernientes a la verificación de las URL
- Creación de ejecutable
- Arreglo de problemas relacionados con la apertura del ejecutable

6. Dificultades:

A lo largo de la realización de este trabajo, nos encontramos con una serie de dificultades que afectaron en mayor o menor medida al tiempo que empleamos para terminar el proyecto, sin embargo, logramos afrontarlas exitosamente, las más notables incluyen:

Etiquetas correctas incorrectamente tratadas: El analizador léxico tomaba como incorrectas algunas etiquetas que estaban correctamente escritas, esto se solucionó mediante la implementación de una función que reconocía y trataba adecuadamente las etiquetas que previamente eran consideradas erróneas.

Reconocimiento incorrecto de las URL por parte del analizador léxico: Este problema se debió a una definición incorrecta de las expresiones regulares utilizadas para identificar las URL. Sin embargo, logramos superar este obstáculo al revisar exhaustivamente las expresiones regulares y realizar modificaciones sucesivas hasta obtener los resultados deseados.

El programa lanzaba un error al momento de ejecutarlo: Implementamos PyInstaller, una herramienta de empaquetado de aplicaciones en Python, que nos permitió crear ejecutables independientes a partir de nuestros scripts. Sin embargo, nos encontramos con un inconveniente relacionado con la apertura de archivos debido a un error en la forma en que pasamos los argumentos. Una vez identificado el problema y corregido, logramos resolver este primer inconveniente.

No obstante, posteriormente nos enfrentamos a otro problema relacionado con los archivos. Afortunadamente, encontramos una solución efectiva importando el archivo abierto como una variable y cerrándolo dentro del bloque principal (main). Esta solución nos permitió gestionar adecuadamente el archivo y evitar cualquier conflicto o error adicional.

En resumen, mediante la corrección de los argumentos de apertura de archivos y la importación adecuada del archivo abierto como una variable, pudimos solucionar los inconvenientes relacionados con los archivos en nuestro proyecto.