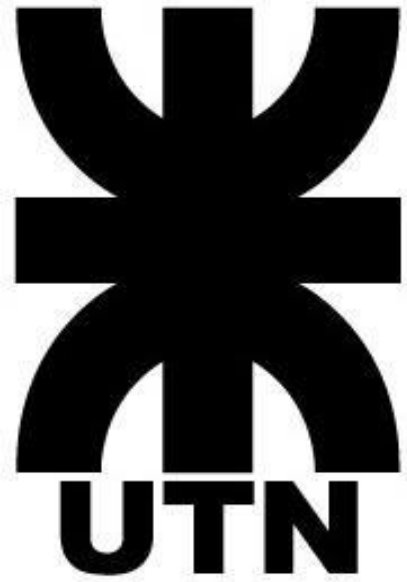

SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

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: 03/07/2023. Resistencia, Chaco

ÍNDICE

ELABORACIÓN DE GRAMÁTICA

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>Y2 X Z</section> | <section>Y2 X</section> | <section>X Z</section>
| <section>X </section>
SSEC → <simplesec>Y2 X</simplesec> | <simplesec>X</simplesec>
Y2 → INFO TITLE2 | INFO | TITLE2
```

Etiquetas básicas de párrafo

```
INFO → <info>A</info>
A → MOBJ A | ABSTRACT A | ADDRESS A | AUTHOR A | DATE A | COPY A | TITLE3 A |
A → MOBJ | ABSTRACT | ADDRESS | AUTHOR | DATE | COPY | TITLE3
ABSTRACT →<abstract>TITLE3 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>
TITLE2→ <title>G</title>
TITLE3→ <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>TITLE3 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
```

ANÁLISIS LÉXICO

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', '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' , 'APERTURA_COPYRIGHT' , 'CIERRE_COPYRIGHT']
```

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'<imageobject>'
t_CIERRE_IMAGENOBJECT = r'</imageobject>'
t_APERTURA_TGROUP = r'<tgroup>'
t_CIERRE_TGROUP = r'</tgroup>'
t_APERTURA_COPYRIGHT = r'<copyright>'
t_CIERRE_COPYRIGHT = r'</copyright>'
t_ERROR_1 = r'<[\w]+>'
t_ERROR_2 = r'<[\w]+\s[\w]+=*[\w"]+\s*/*>'
t_ERROR_3 = r'</[\w]+>'

def t_APERTURA_TITLE(t):
      r'<title>'
      return(t)
def t_CIERRE_TITLE(t):
      r'</title>'
      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)
```

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 '   #ignora espacios en blanco
```

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()   #creacion del lexer
```

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

Funciones de conversión:

```
def t_DT1(t):
      r'<[!]DOCTYPE\sarticle>'
      arch.write("<!DOCTYPE html>\n")
      return(t)
def t_TEXTO (t):
    r'[\w.,:;_%/+?¿¡!()"\'_|°¬~$&=^`{}\#@*\-,\[\]\\\s]+[\s]*
    arch.write(f'{t.value} ')
    return (t)
def t_APERTURA_PARA(t):
      r'<para>'
      arch.write("<p>\n")
      return(t)
def t_CIERRE_PARA(t):
      r'</para>'
      arch.write("</p>\n")
      return(t)

def t_APERTURA_INFO(t):
      r'<info>'
      arch.write('<p           style="color:white;background-color:green;font-
size:8px">\n')
      return(t)
def t_CIERRE_INFO(t):
      r'</info>'
      arch.write('</p>\n')
      return(t)
def t_APERTURA_IMPORTANT(t):
      r'<important>'
      arch.write('<div style="background-color:red;color:white">\n')
      return(t)
def t_CIERRE_IMPORTANT(t):
      r'</important>'
      arch.write('</div>\n')
      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*>'
      direccion= t.value
      i=0
      x=0
      newdir=""
      while i==0:       #Guarda en la variable "newdir" solo la url que aparecce
en la etiqueta link
            x=x+1
            if direccion[x] == '"':
```

```
                  x+=1
                  while direccion[x] != '"':
                        newdir+= direccion[x]
                        x+=1
                  i=1
      arch.write(f'<a href="{newdir}">\n')
      return (t)
def t_CIERRE_LINK (t):
      r'</link>'
      arch.write("</a>")
      return (t)
def t_APERTURA_INFORMALTABLE(t):
      r'<informaltable>'
      arch.write("<table>\n")
      return (t)
def t_CIERRE_INFORMALTABLE(t):
      r'</informaltable>'
      arch.write("</table>\n")
      return (t)
def t_APERTURA_THEAD(t):
      r'<thead>'
      arch.write("<thead>\n")
      global Tpos
      Tpos = 0                              #Tpos significa "posicion en tabla"
si es = 0 es porque estamos en el head
      return(t)                            #Para canda entry en head necesitamos
<th></th>, mientras que para tbody o tfoot
def t_CIERRE_THEAD(t):                      #se requiere de <td></td>
      r'</thead>'
      arch.write("</thead>\n")
      return(t)
def t_APERTURA_TBODY(t):
      r'<tbody>'
      arch.write("<tbody>\n")
      global Tpos
      Tpos = 1
      return(t)
def t_CIERRE_TBODY(t):
      r'</tbody>'
      arch.write("</tbody>\n")
      return(t)
def t_APERTURA_TFOOT(t):
      r'<tfood>'
      global Tpos
      Tpos=1
      arch.write("<tfood>\n")
      return(t)
def t_CIERRE_TFOOT(t):
      r'</tfood>'
      arch.write("</tfood>\n")
      return(t)
def t_APERTURA_ROW(t):                                  #en html todos los "row"
son tr y se diferencian adentro usando
      r'<row>'                                               #<th></th> para los
encabezados y <td></td> para el resto
      arch.write("<tr>\n")
      return (t)
```

```
def t_CIERRE_ROW(t):
      r'</row>'
      arch.write("</tr>\n")
      return (t)
def t_APERTURA_ENTRY(t):
      r'<entry>'
      global Tpos
      if Tpos == 0:
            arch.write("<th>\n")
      elif Tpos == 1:
            arch.write("<td>\n")
      return (t)
def t_CIERRE_ENTRY(t):
      r'</entry>'
      global Tpos
      if Tpos == 0:
            arch.write("</th>\n")
      elif Tpos == 1:
            arch.write("</td>\n")
      return (t)
def t_APERTURA_ENTRYTBL(t):
      r'<entrytbl>'
      arch.write("<table>")
      return (t)
def t_CIERRE_ENTRYTBL(t):
      r'</entrytbl>'
      arch.write("</table>")
      return (t)
def t_APERTURA_ITEMIZEDLIST(t):
      r'<itemizedlist>'
      arch.write("<ul>\n")
      return (t)
def t_CIERRE_ITEMIZEDLIST(t):
      r'</itemizedlist>'
      arch.write("</ul>\n")
      return (t)
def t_APERTURA_LISTITEM(t):
      r'<listitem>'
      arch.write("<il>\n")
      return (t)
def t_CIERRE_LISTITEM(t):
      r'</listitem>'
      arch.write("</il>\n")
      return (t)
```

La variable “Tpos” es utilizada para llevar un seguimiento de la posición actual en una tabla HTML durante el análisis léxico.
Esta variable es utilizada en las funciones t_APERTURA_ENTRY(t) y t_CIERRE_ENTRY(t), que se activan cuando se encuentra una etiqueta de apertura y cierre de una celda (<entry> y </entry> respectivamente). Dependiendo del valor de Tpos, se decide si se debe generar una etiqueta <th> o <td> correspondiente a la celda en el archivo HTML de salida.
Esta variable tiene dos posibles valores: 0 y 1.
Cuando Tpos es igual a 0, indica que el analizador léxico se encuentra dentro del encabezado de una tabla (<thead>). En este caso, se esperan etiquetas <th> para cada celda de encabezado en la tabla.

Cuando Tpos es igual a 1, indica que el analizador léxico se encuentra dentro del cuerpo de la tabla (<tbody>) o del pie de tabla (<tfoot>). En este caso, se esperan etiquetas <td> para las celdas de contenido en la tabla.

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(): #Borra lo ya escrito en pantalla
      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":                                                    #ingreso manual
      borrarPantalla()
      while True: #ciclo para ingresar datos hasta que eleccion sea 0
            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":                                         #ingreso por archivo
      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:
                  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")
   os.system("pause")
```

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.

7. Retroalimentación:

Luego de completar la segunda entrega, recibimos comentarios que señalaban la detección de caracteres ilegales en algunos ejemplos, a pesar de que en realidad no eran ilegales. Además, se identificaron problemas con algunas URLs que no funcionaban correctamente.

Para abordar estas cuestiones, llevamos a cabo una revisión minuciosa del código y corregimos las expresiones regulares utilizadas por el lexer. Mediante este proceso de revisión y corrección, logramos solucionar los problemas detectados, asegurando así el correcto funcionamiento del programa.

ANÁLISIS SINTÁCTICO

1. INTRODUCCION:

Un analizador sintáctico (o parser) es una herramienta esencial en el proceso de análisis sintáctico de un lenguaje de programación. Su función principal es recibir como entrada una secuencia de tokens generados por el lexer y construir una estructura jerárquica que represente la estructura gramatical del programa.

Mientras que el lexer se encarga de dividir el código fuente en unidades léxicas, como palabras clave, identificadores y símbolos, el parser va un paso más allá y analiza cómo estas unidades léxicas se combinan para formar las construcciones sintácticas correctas según la gramática del lenguaje.

Para que un parser funcione correctamente, es necesario proporcionarle reglas gramaticales que describan la sintaxis del lenguaje que se está analizando. Estas reglas, también conocidas como gramáticas formales, definen la estructura y las combinaciones válidas de los elementos del lenguaje.

Las reglas gramaticales se expresan típicamente en forma de gramáticas de contexto libre, utilizando notaciones como la forma de Backus-Naur (BNF) o la forma extendida de Backus-Naur (EBNF). Estas gramáticas describen las producciones y las reglas de derivación que permiten construir correctamente las sentencias del lenguaje.

Cada regla gramatical define la estructura de un componente sintáctico, como una expresión, una declaración o una sentencia, y puede incluir no solo terminales (tokens generados por el lexer), sino también no terminales (símbolos que se definen por otras reglas). Las reglas se organizan de manera jerárquica, lo que permite construir la estructura sintáctica en varios niveles.

Cuando se carga una gramática en el parser, este utiliza las reglas definidas para realizar el análisis sintáctico del código fuente. El parser aplica algoritmos como el análisis descendente o el análisis ascendente para verificar si el código se ajusta a la gramática especificada. En caso de que el código no sea sintácticamente válido, se generará un mensaje de error.

En esta etapa del proyecto, hemos decidido retomar el uso de **Python** como lenguaje de programación y **GitHub** como plataforma colaborativa en línea, debido a las razones expuestas en la etapa anterior.

2. PARSER:

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.yacc', el cual proporciona herramientas para construir analizadores sintácticos utilizando la técnica de análisis sintáctico LR (Left-to-right, Rightmost derivation) en Python. Permite definir reglas gramaticales y generar un parser a partir de ellas.

Para esto, lo importamos, de la siguiente forma:

```
import ply.yacc as yacc
```

Además, importamos el módulo 'os', para realizar operaciones relacionadas con archivos y directorios, como acceder a rutas, crear directorios, etc. :

```
import os
```

El módulo 'codecs', el cual proporciona funciones para trabajar con diferentes codecs (conjuntos de reglas de codificación y decodificación) de caracteres. Se utiliza para leer y escribir archivos con diferentes codificaciones de caracteres. :

```
import codecs
```

El módulo 're', que proporciona herramientas para trabajar con expresiones regulares. Permite buscar, analizar y manipular cadenas de texto utilizando patrones específicos. :

```
import re
```

Y, finalmente, importamos los tokens generados por el lexer y el archivo de análisis léxico. Esto permite utilizar los tokens en las reglas gramaticales del parser:

```
from lexer import tokens, arch
```

2.2 Variables:

En el código, definiremos únicamente la variable "correcto", la cual inicializa con el valor 0 al comienzo del código.

```
correcto = 0
```

Esta variable se utiliza para verificar si hubo algún error de sintaxis durante el análisis del código fuente. Si se produce un error, el valor de "correcto" se actualizará a 1. De esta manera, se puede determinar si el análisis sintáctico fue exitoso o si se encontraron errores.

2.3 Definición de reglas:

Las funciones de regla en el código implementan las reglas de producción gramatical y definen cómo se debe analizar la estructura sintáctica del lenguaje en cuestión.

Estructurales del documento

```
def p_sigma(p):
    ''' sigma : DT1 APERTURA_ARTICLE y x z CIERRE_ARTICLE
        | DT1 APERTURA_ARTICLE y x CIERRE_ARTICLE
        | DT1 APERTURA_ARTICLE x z CIERRE_ARTICLE
        | DT1 APERTURA_ARTICLE x CIERRE_ARTICLE'''
def p_y(p):                                                          #y deriva
en info o en title y se corresponde solo con article
    ''' y : info title
        | info
        | title'''
def p_y2(p):                                                              #y2
deriva en info o en title2 y se corresponde solo con section y simplesection
    ''' y2 : info title2
        | info
        | title2'''
def p_x(p):
    ''' x : itemlist x
        | important x
        | para x
        | spara x
        | address x
        | mobj x
        | inftable x
        | comment x
        | abstract x
        | itemlist
        | important
        | para
        | spara
        | address
        | mobj
        | inftable
        | comment
        | abstract'''
def p_z(p):
    ''' z : sec z
        | ssec z
        | sec
        | ssec'''


def p_sec(p):
    ''' sec : APERTURA_SECTION y2 x z CIERRE_SECTION
        | APERTURA_SECTION y2 x CIERRE_SECTION
        | APERTURA_SECTION x z CIERRE_SECTION
        | APERTURA_SECTION x CIERRE_SECTION'''


def p_ssec(p):
    ''' ssec : APERTURA_SIMPLESECT y2 x CIERRE_SIMPLESECT
        | APERTURA_SIMPLESECT x CIERRE_SIMPLESECT'''
```

Básicas de párrafo

```
def p_info(p):
    ''' info : APERTURA_INFO a CIERRE_INFO'''
def p_a(p):
    ''' a : mobj a
        | abstract a
        | address a
        | author a
        | date a
        | copy a
        | title3 a
        | mobj
        | abstract
        | address
        | author
        | date
        | copy
        | title3'''
def p_abstract(p):
    ''' abstract : APERTURA_ABSTRACT title3 b CIERRE_ABSTRACT
        | APERTURA_ABSTRACT b CIERRE_ABSTRACT'''
def p_b(p):
    ''' b : para b
        | spara b
        | para
        | spara'''
def p_address(p):
    ''' address : APERTURA_ADDRESS c CIERRE_ADDRESS'''
def p_c(p):
    ''' c : TEXTO c
        | street c
        | city c
        | state c
        | phone c
        | email c
        | TEXTO
        | street
        | city
        | state
        | phone
        | email'''
def p_author(p):
    ''' author : APERTURA_AUTHOR d CIERRE_AUTHOR'''
def p_d(p):
    ''' d : fname d
        | sname d
        | fname
        | sname'''
def p_copy(p):
    ''' copy : APERTURA_COPYRIGHT e f CIERRE_COPYRIGHT
        | APERTURA_COPYRIGHT e CIERRE_COPYRIGHT'''
def p_e(p):
    ''' e : year e
        | year'''
def p_f(p):
    ''' f : holder f
        | holder'''
```

```
def p_title(p):
    ''' title : seen_AT1 APERTURA_TITLE g seen_CT1 CIERRE_TITLE '''    #la regla
seen_AT es la propuesta de ply para acciones embebidas

def p_seen_AT1(p):
    "seen_AT1 :"                                                    #se define una
regla seen_token que unicamente realiza una accion
    arch.write("<h1>\n")
def p_seen_CT1(p):
    "seen_CT1 : "
    arch.write("</h1>\n")

def p_title2(p):
    ''' title2 : seen_AT2 APERTURA_TITLE  g seen_CT2 CIERRE_TITLE'''        #Si
entra a titleN si o si tiene que abrir etiqueta hN
def p_seen_AT2(p):
    '''seen_AT2 :'''
    arch.write("<h2>\n")
def p_seen_CT2(p):
    '''seen_CT2 :'''
    arch.write("</h2>\n")

def p_title3(p):
    ''' title3 : seen_AT3 APERTURA_TITLE g seen_CT3 CIERRE_TITLE '''
def p_seen_AT3(p):
    '''seen_AT3 :'''
    arch.write("<h3>\n")
def p_seen_CT3(p):
    '''seen_CT3 :'''
    arch.write("</h3>")

def p_g(p):
    ''' g : TEXTO g
        | emphasis g
        | link g
        | email g
        | TEXTO
        | emphasis
        | link
        | email'''
def p_spara(p):
    ''' spara : APERTURA_SIMPARA h CIERRE_SIMPARA'''
def p_emphasis(p):
    ''' emphasis : APERTURA_EMPHASIS h CIERRE_EMPHASIS'''
def p_comment(p):
    ''' comment : APERTURA_COMMENT h CIERRE_COMMENT'''
def p_link(p):
    ''' link : APERTURA_LINK h CIERRE_LINK'''
def p_h(p):
    ''' h : TEXTO h
        | emphasis h
        | link h
        | email h
        | author h
        | comment h
```

```
        | TEXTO
        | emphasis
        | link
        | email
        | author
        | comment'''
def p_para(p):
    ''' para : APERTURA_PARA i CIERRE_PARA'''
def p_i(p):
    ''' i : TEXTO i
        | emphasis i
        | link i
        | email i
        | author i
        | comment i
        | itemlist i
        | important i
        | address i
        | mobj i
        | inftable i
        | TEXTO
        | emphasis
        | link
        | email
        | author
        | comment
        | itemlist
        | important
        | address
        | mobj
        | inftable'''
def p_important(p):
    ''' important : APERTURA_IMPORTANT title3 j CIERRE_IMPORTANT
        | APERTURA_IMPORTANT j CIERRE_IMPORTANT'''
def p_j(p):
    ''' j : itemlist j
        | important j
        | para j
        | spara j
        | address j
        | mobj j
        | inftable j
        | comment j
        | abstract j
        | itemlist
        | important
        | para
        | spara
        | address
        | mobj
        | inftable
        | comment
        | abstract'''
def p_fname(p):
    ''' fname : APERTURA_FIRSTNAME k CIERRE_FIRSTNAME'''
def p_sname(p):
    ''' sname : APERTURA_SURNAME k CIERRE_SURNAME'''
```

```
def p_street(p):
    ''' street : APERTURA_STREET k CIERRE_STREET'''
def p_city(p):
    ''' city : APERTURA_CITY k CIERRE_CITY'''
def p_state(p):
    ''' state : APERTURA_STATE k CIERRE_STATE'''
def p_phone(p):
    ''' phone : APERTURA_PHONE k CIERRE_PHONE'''
def p_email(p):
    ''' email : APERTURA_EMAIL k CIERRE_EMAIL'''
def p_date(p):
    ''' date : APERTURA_DATE k CIERRE_DATE'''
def p_year(p):
    ''' year : APERTURA_YEAR k CIERRE_YEAR'''
def p_holder(p):
    ''' holder : APERTURA_HOLDER k CIERRE_HOLDER'''
def p_k(p):
    ''' k : TEXTO k
        | link k
        | emphasis k
        | comment k
        | TEXTO
        | link
        | emphasis
        | comment'''
```

Imágenes y multimedia

```
def p_mobj(p):
    ''' mobj : APERTURA_MEDIAOBJECT info l CIERRE_MEDIAOBJECT
        | APERTURA_MEDIAOBJECT l CIERRE_MEDIAOBJECT'''
def p_l(p):
    ''' l : vobj l
        | iobj l
        | vobj
        | iobj'''
def p_iobj(p):
    ''' iobj : APERTURA_IMAGENOBJECT info idata CIERRE_IMAGENOBJECT
        | APERTURA_IMAGENOBJECT idata CIERRE_IMAGENOBJECT'''
def p_idata(p):
    ''' idata : APERTURA_IMAGEDATA'''
def p_vobj(p):
    ''' vobj : APERTURA_VIDEOOBJECT info vdata CIERRE_VIDEOOBJECT
        | APERTURA_VIDEOOBJECT vdata CIERRE_VIDEOOBJECT'''
def p_vdata(p):
    ''' vdata : APERTURA_VIDEODATA'''
```

Listas

```
def p_itemlist(p):
    ''' itemlist : APERTURA_ITEMIZEDLIST m CIERRE_ITEMIZEDLIST'''
def p_m(p):
    ''' m : litem m
        | litem'''
def p_litem(p):
    ''' litem : APERTURA_LISTITEM j CIERRE_LISTITEM'''
```

Tablas

```
def p_inftable(p):
```

```
    ''' inftable : APERTURA_INFORMALTABLE n CIERRE_INFORMALTABLE'''
def p_n(p):
    ''' n : mobj n
        | tgroup n
        | mobj
        | tgroup'''
def p_tgroup(p):
    ''' tgroup : APERTURA_TGROUP o CIERRE_TGROUP'''
def p_o(p):
    ''' o : thead tfood tbody
        | tfood tbody
        | thead tbody'''
def p_thead(p):
    ''' thead : APERTURA_THEAD oo CIERRE_THEAD'''
def p_tfood(p):
    ''' tfood : APERTURA_TFOOT oo CIERRE_TFOOT'''
def p_tbody(p):
    ''' tbody : APERTURA_TBODY oo CIERRE_TBODY'''
def p_oo(p):
    ''' oo : row oo
        | row'''
def p_row(p):
    ''' row : APERTURA_ROW q CIERRE_ROW'''
def p_q(p):
    ''' q : entry q
        | entrytbl q
        | entry
        | entrytbl'''
def p_entry(p):
    ''' entry : APERTURA_ENTRY r CIERRE_ENTRY'''
def p_r(p):
    ''' r : itemlist r
        | important r
        | para r
        | spara r
        | TEXTO r
        | mobj r
        | comment r
        | abstract r
        | itemlist
        | important
        | para
        | spara
        | TEXTO
        | mobj
        | comment
        | abstract'''
def p_entrytbl(p):
    ''' entrytbl : APERTURA_ENTRYTBL t CIERRE_ENTRYTBL'''
def p_t(p):
    ''' t : thead tbody
        | tbody'''
```

2.4 Funciones:

Al final del código, se define una regla de error p_error(p).

```
def p_error(p):
    global correcto
    correcto = 1
    if p:
        print(f"Error de sintaxis {p.value}. Revise linea {p.lineno}")
        # Descartar el token de error para que siga su camino
        parser.errok()
    else:
         print("Error de sintaxis EOF")
```

La función p_error(p) se ejecuta cuando ocurre un error de sintaxis durante el análisis. En el código dado, establece la variable correcto en 1 para indicar que se ha detectado un error. También muestra un mensaje de error personalizado y proporciona información adicional sobre el token o símbolo que causó el error. Su propósito es manejar los errores de sintaxis de manera personalizada en lugar de utilizar el mensaje de error predeterminado.

Por último, se crea el objeto del parser utilizando yacc.yacc() del módulo ply.yacc.

```
parser = yacc.yacc()
```

2.5 Acciones Incrustadas:

Las acciones incrustadas, también conocidas como acciones semánticas o código de acción, son fragmentos de código que se ejecutan durante el proceso de análisis sintáctico de un analizador generado por herramientas como Yacc. Estas acciones se utilizan para realizar tareas específicas cuando se alcanza un determinado punto en la gramática mientras se procesa una entrada.

Las acciones incrustadas se colocan dentro de las reglas gramaticales y se ejecutan cuando se cumple esa regla durante el análisis. Esto permite realizar acciones adicionales, como realizar cálculos, modificar variables, realizar operaciones de impresión o incluso llamar a funciones externas.

Nosotros utilizamos múltiples acciones incrustadas, las cuales se ejecutan durante el análisis sintáctico. Cada una de ellas tiene la función de escribir las etiquetas de apertura y cierre correspondientes en el archivo arch.

La regla **p_seen_AT1(p)** Se activa al encontrar el token **seen_AT1**. Escribe la etiqueta de apertura <h1> en el archivo arch.

La regla **p_seen_CT1(p)** Se activa al encontrar el token **seen_CT1**. Escribe la etiqueta de cierre </h1> en el archivo arch.

La regla **p_seen_AT2(p)** Se activa al encontrar el token **seen_AT2**. Escribe la etiqueta de apertura <h2> en el archivo arch.

La regla **p_seen_CT2(p)** Se activa al encontrar el token **seen_CT2**. Escribe la etiqueta de cierre </h2> en el archivo arch.

La regla **p_seen_AT3(p)** Se activa al encontrar el token **seen_AT3**. Escribe la etiqueta de apertura <h3> en el archivo arch.

La regla **p_seen_CT3(p)** Se activa al encontrar el token **seen_CT3**. Escribe la etiqueta de cierre </h3> en el archivo arch.

3. IMPLEMENTACIÓN:

Ahora podemos utilizar el Parser creado para analizar texto.
Para esto, lo primero que debemos hacer es importar nuestro parser:

```
from parser1 import parser
```

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():                #Borra lo ya escrito en pantalla
      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 Sintactico")
print ("1 para ingresar datos a mano\n2 si quiere cargar datos desde un
archivo\n")
op = input()
errores = []
```

3.1 Ingreso manual:

Si la opción elegida es "1", se realiza un ingreso manual de datos. El programa limpia la pantalla y entra en un bucle infinito para permitir al usuario ingresar datos a analizar.
Acción seguida, solicita al usuario que ingrese la cadena a analizar, llama al objeto parser para analizar la cadena y verifica el valor de la variable correcto importada desde parser1. Si es igual a cero, imprime "sintaxis correcta".
Finalmente, pregunta al usuario si desea continuar o terminar el programa.Si la respuesta es "0", rompe el bucle y finaliza el programa.

```
if op == "1":                                  #ingreso manual
    borrarPantalla()
    while True:    #ciclo para ingresar datos hasta que eleccion sea 0
        print("ingrese lo que quiere analizar")
        cadena = input()
        parser.parse(cadena)
        from parser1 import correcto
        if correcto == 0:
            print("sintaxis correcta")
        print("desea continuar?\n1 para continuar\n0 para terminar")
        eleccion = input()
        if eleccion == "0": break
```

```
        borrarPantalla()
    arch.close()
```

3.2 Ingreso por archivo:

Si el usuario elige la opción “2”, el programa realiza un ingreso de datos a través de un archivo. Comienza mostrando una lista de archivos en una carpeta específica y permite al usuario seleccionar uno de ellos.
Luego, lee el contenido del archivo seleccionado y lo pasa como entrada al analizador sintáctico. Si el análisis es exitoso, se crea una copia del archivo en formato HTML con el nombre correspondiente en la ubicación especificada.
Cabe destacar que si ya existe un archivo con el mismo nombre en la carpeta de destino, se eliminará antes de realizar el cambio de nombre.
Finalmente, se muestra el resultado y se pausa la ejecución.

```
elif op == "2":                                         #ingreso por archivo
    n = 0
    dir = '../doc/prueba/'                      #elegir el archivo
    with os.scandir(dir) as 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" not in j and ".xml" not 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"../doc/prueba/{ruta}","r",encoding="utf-8") as maestro:
            parser.parse(maestro.read())
            from parser1 import correcto
            if correcto == 0:
                print("sintaxis correcta")
            arch.close()
            if ".xml" in ruta:
                cambio = ruta.replace(".xml","")
            elif ".txt" in ruta:
                cambio = ruta.replace(".txt","")
            with os.scandir('../src/html_generados/') as htmls:
                for k in htmls: #si ya existe un archivo con el mismo nombre
(por multiples ejecuciones por ej) lo borra.
                        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" )
            input("Pulsa cualquier tecla para continuar...")
    else:
        print("numero invalido")
        input("Pulsa cualquier tecla para continuar...")
```

Implementamos una funcionalidad en el programa que borra automáticamente cualquier archivo con una extensión diferente a .txt o .xml. Esto significa que, al leer archivos, el programa solo mostrará aquellos que tengan esas extensiones específicas.
De esta manera, nos aseguramos de que solo se procesen y muestren los archivos relevantes para el funcionamiento correcto del programa, eliminando los archivos no deseados.

Si el número ingresado no es válido, se muestra un mensaje de error.

```
else:
      print("volve a empezar")
      input("Pulsa cualquier tecla para continuar...")
```

4. Demostración Práctica:

Para poner a prueba el funcionamiento del parser y proporcionar ejemplos concretos, llevamos a cabo un conjunto de pruebas utilizando diferentes archivos de prueba.

LISTAS.xml

```
<!DOCTYPE article>
<article>
    <info>
        <title>Archivo de prueba presentacion del grupo </title>

        <aabstract>
            <title>integrantes </title>
            <para>A continuacion se mostraran los nombres de los distintos
integrantes del grupo </para>
        </abstract>

        <author>
            <firtname>Camilo</firtname>
            <surname>Aguirre </surname>
        </author>
        <author>
            <firtname> Joaquin </firtname>
            <surname> Bianciotto </surname>
        </author>
        <author>
            <firtname> Julian </firtname>
            <surname> Colombo </surname>
        </author>
        <author>
            <firtname> Yoel </firtname>
            <surname> Maraim </surname>
        </author>
    </info>

    <comment> correo electronico de contacto <email> grupo4@mail.com </email>
</comment>

    <section>
        <title> Las 3 entregas parciales del tpi son las siguiente: </title>

        <itemizedlist>
            <listitem>
                <para>1er entrega: Documentacia del proyecto y la gramatica a
generar, <emphasis> el 23 de abril </emphasis> </para>
            </listitem>

            <listitem>
                <para>2da entrega: Presentacion del lexer que reconozca los
tokens del leguaje, <emphasis> el 4 de junio </emphasis> </para>
            </listitem>

            <listitem>
                <para>3er entrega (entrega final): presentacion del tpi
completo, lexer y parse, incluyendo toda la presentacion del tpi antes de la
clase con una exposicion hora de no menos de 20 minutos <emphasis> el 2 junio
</emphasis> </para>
```

```
            </listitem>
        </itemizedlist>

        <para>Con esto se concluye el archivo de prueba, donde se experimento
con distinitas etiquetas anidadas, como asi tmb listas</para>
    </section>
</article>
```

El parser identificó los siguientes errores:

Línea 6 -> Se abre “<aabstract>”, etiqueta no definida en la gramática.

Línea 8 -> Aparece un “<para>”, el mismo no puede aparecer (Como nunca se abrió el abstract, mal escrito en la línea 6, el para no se encuentra dentro de una estructura reconocida).

Línea 9 -> Se cierra un abstract que nunca se abre.

Líneas 12, 16, 20 y 24 -> Se abren y cierran etiquetas “<firtname>”, etiquetas no definidas en la gramática.

TABLAS.xml

```
<!DOCTYPE article>
<article>
    Archivo de prueba sobre tablas </title>

    <abstract><para>Se vera a continuacion una tabla de puntos sobre del top 5
de la liga de futbol argentino <emphasis> campeonatado 2023 </emphasis>, a modo
de demostracion </para>
    </abstract>

    <informaltable>
        <tgroup>
            <thead>
                <row>
                    <entry>
                        <important>
                            <title> CAMPEONATO ARGENTINO 2023 </title>
                            <comment>prueba</comment>
                        </important>
                    </entry>
                </row>
            </thead>
            <tbody>
                <row>
                    <entry>
                        <important>
                            <title> EQUIPO </title>
                            <comment>prueba2</comment>
                        </important>
                    </entry>
                </row>
                <row>
                    <entry>
                        <para>  River plate</para>
                        <para> 40 </para>
```

```
                    </entry>
                </row>
                <row>
                    <entry>
                        <para> San lorenzo</para>
                        <para> 35 </para>
                    </entry>
                </row>
                <row>
                    <entry>
                        <para> talleres </para>
                        <para> 31 </para>
                    </entry>
                </row>
                <row>
                    <entry>
                        <para> Estudiantes </para>
                        <para> 31 </para>
                    </entry>
                </row>
                <row>
                    <entry>
                        <para> Rosario central </para>
                        <para> 30 </para>
                    </entry>
                </row>
            </tbody>
        </tgroup>
    </informaltable>

    <section>
        <para> final del archivo de prueba de tablas </para>
    </section>
</article>
```

El parser identificó los siguientes errores:

Línea 3 -> Aparece texto sin una etiqueta de apertura, se cierra un title que nunca se abre.

También probamos nuestro parser con un archivo en el que incluímos la mayoría de las etiquetas posibles:

pruebafinal.xml

```
<!DOCTYPE article>
<article>
    <info>

        <mediaobject>
         <imageobject>
            <info>
                <abstract> <para>Logo de la utn. </para> </abstract>
            </info>
            <imagedata
fileref="https://es.wikipedia.org/wiki/Archivo:UTN_logo.jpg" />
```

```
            </imageobject>
        </mediaobject>

        <abstract>
            <para>                                                          <link
xlink:href="https://es.wikipedia.org/wiki/Archivo:UTN_logo.jpg" /> </para>
        </abstract>

        <abstract>
            <title>integrantes </title>
            <para>A continuacion se mostraran los nombres de los distintos
integrantes del grupo </para>
        </abstract>

        <author>
            <firstname>Camilo</firstname>
            <surname>Aguirre </surname>
        </author>
        <author>
            <firstname> Joaquin </firstname>
            <surname> Bianciotto </surname>
        </author>
        <author>
            <firstname> Julian </firstname>
            <surname> Colombo </surname>
        </author>
        <author>
            <firstname> Yoel </firstname>
            <surname> Maraim </surname>
        </author>

        <abstract>
            <title>Enlaces de interes</title>
            <para> Somos el grupo n°4 de ISI-A en la materia sintaxis y semantica
de los lenguajes de la Universdiad tecnologica nacional regional
Resistencia</para>
            <para> los enlace proporcionado nos llevara a nuestro repositorio
online en github asi como tambien otra informacion de interes</para>
        </abstract>

    </info>

    <title>Archivo de prueba presentacion del grupo </title>

    <para>
        <important><para> Respositorio de github </para></important>
        <link
xlink:href="https://github.com/joaquinbianciotto/TPI_interprete_docbook"      >
Click aqui </link>
    </para>
    <para>
        <important><para> Pagina web de la Universidad </para></important>
        <link xlink:href="https://www.frre.utn.edu.ar/" > <comment> Click aquí
</comment> </link>
    </para>
```

```
    <section>
        <title> Las 3 entregas parciales del tpi son las siguiente: </title>

        <itemizedlist>
            <listitem>
                <para>1er entrega: Documentacia del proyecto y la gramatica a
generar, <emphasis> el 23 de abril </emphasis> </para>
            </listitem>

            <listitem>
                <para>2da entrega: Presentacion del lexer que reconozca los
tokens del leguaje, <emphasis> el 4 de junio </emphasis> </para>
            </listitem>

            <listitem>
                <para>3er entrega (entrega final): presentacion del tpi
completo, lexer y parse, incluyendo toda la presentacion del tpi antes de la
clase con una exposicion hora de no menos de 20 minutos <emphasis> el 2 junio
</emphasis> </para>
            </listitem>
        </itemizedlist>

        <para>Con esto se concluye el archivo de prueba, donde se experimento
con distinitas etiquetas anidadas, como asi tmb listas</para>
    </section>

    <simplesect>
        <title> Profesores de la catedra</title>
        <informaltable>
        <tgroup>
            <thead>
                <row>
                    <entry>
                        Titular
                    </entry>

                    <entry>
                        Jefe de trabajos practicos
                    </entry>
                    <entry>
                        Auxiliar de primero
                    </entry>
                    <entry>
                        Auxiliar de segunda
                    </entry>
                </row>
            </thead>
            <tbody>
                <row>
                    <entry>
                        ESP.ing.Gabriela Tomaselli
                    </entry>
                    <entry>
                        Ing. Rodrigo Vigil
```

```
                    </entry>
                    <entrytbl>
                        <tbody>
                            <row>
                                <entry>
                                    Ing. Nicolas Tortosa
                                </entry>
                            </row>
                            <row>
                                <entry>
                                    Ing. Juliana Jorre
                                </entry>
                            </row>
                        </tbody>
                    </entrytbl>
                    <entry>
                        Franco Ballesteros/Ballesta
                    </entry>
                </row>
            </tbody>
        </tgroup>
    </informaltable>
    </simplesect>
    <section>
        <para>Gracias por leernos. </para>
    </section>
</article>
```

El parser no identificó ningún error, la sintaxis es correcta.

5. 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:

Cambios necesarios en la gramática: Realizamos modificaciones gramaticales para garantizar el correcto funcionamiento del parser y la traducción a HTML.

Problemas en la traducción de Links: Enfrentamos desafíos al momento de traducir los enlaces a HTML, ya que el programa interpretaba las etiquetas como parte del enlace y las incluía en el texto final. Resolvimos este problema al almacenar la etiqueta en una cadena de texto y restringir la traducción únicamente a lo que se encontraba entre las comillas.

Errores de Sintaxis erróneos: Experimentamos situaciones en las que el programa detectaba errores de sintaxis inexistentes. El problema radicaba en una carga incorrecta de la gramática en el parser, lo que resultó en reglas gramaticales mal configuradas. Solucionamos esto corrigiendo las reglas gramaticales del parser.

Problemas en la traducción de Títulos: En la entrega anterior, el lexer se encargaba de traducir los títulos a HTML. Sin embargo, en esta última entrega surgieron problemas debido a que, a diferencia de DocBook que presenta un único tipo de título, HTML posee diferentes tipos de títulos que deben utilizarse según la estructura en la que se encuentran.

Para abordar este desafío, modificamos la gramática y ahora el parser se encarga de la traducción de los títulos.

Al realizar la traduccion en el parser, podemos realizar acciones dependiendo del momento de la derivacion en que nos encontremos, por ello y gracias a acciones incrustada, antes de derivar el title del documento una accion embebida escribe la etiqueta <h1>, lo mismo para el cierre. Cuando se encuentre por derivar el titulo de una seccion, una accion inscrustada escribira el <h2> correspondiente con su cierre.

Problemas en la creación del ejecutable: El ejecutable solo funcionara correctamente cuando se encontraba en la carpeta de inicio y no en la carpeta de bin. Para resolver esto, realizamos modificaciones en todas las rutas para que funcionaran correctamente desde la carpeta de bin.

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

9) Junio 18
- Primera prueba de código generador de Parser
- Cambios en la gramática

10) Junio 20
- Primera generación exitosa del Parser
- Corrección de errores en la redacción
- Modificación de la expresión regular de las URL
- Modificación de la expresión regular del texto
- Eliminación del tokens no utilizados en el Lexer

11) Junio 27
- Modificación de expresiones regulares
- Cambios en la funcionalidad del lexer
- Implementación de variables para traducir los títulos a html desde el Lexer
- Implementación de variable “Tpos” para traducir las tablas a html
- Implementación de variable “correcto” para identificar errores sintácticos
- Implementación del parser
- Creación del código principal
- Cambios visuales en la traducción a html
- Corrección de errores en las definiciones de reglas sintácticas
- Implementación de la función BorrarPantalla() en el Parser
- Cambios en los archivos de prueba

12) Junio 29
- Cambios en la funcionalidad, la traducción de títulos a html pasa a realizarla el parser
- Cambios de funcionamiento para mejorar la compatibilidad del programa
- Mejoras en el parser, permitiendo la lectura de múltiples errores

13) Julio 2
- Cambios en la traducción de etiquetas link
- Cierre de archivos
- Corrección de errores
- Cambios generales, se comentó el código
- Creación de ejecutable final
- Corrección de rutas

WEBGRAFIA

- https://wiki.archlinux.org/title/DocBook
- https://web.archive.org/web/20120123183312/http://www.dpawson.co.uk/docbook/ref erence.html
- https://ply.readthedocs.io/en/latest/ply.html#
- https://ply.readthedocs.io/en/latest/ply.html#embedded-actions