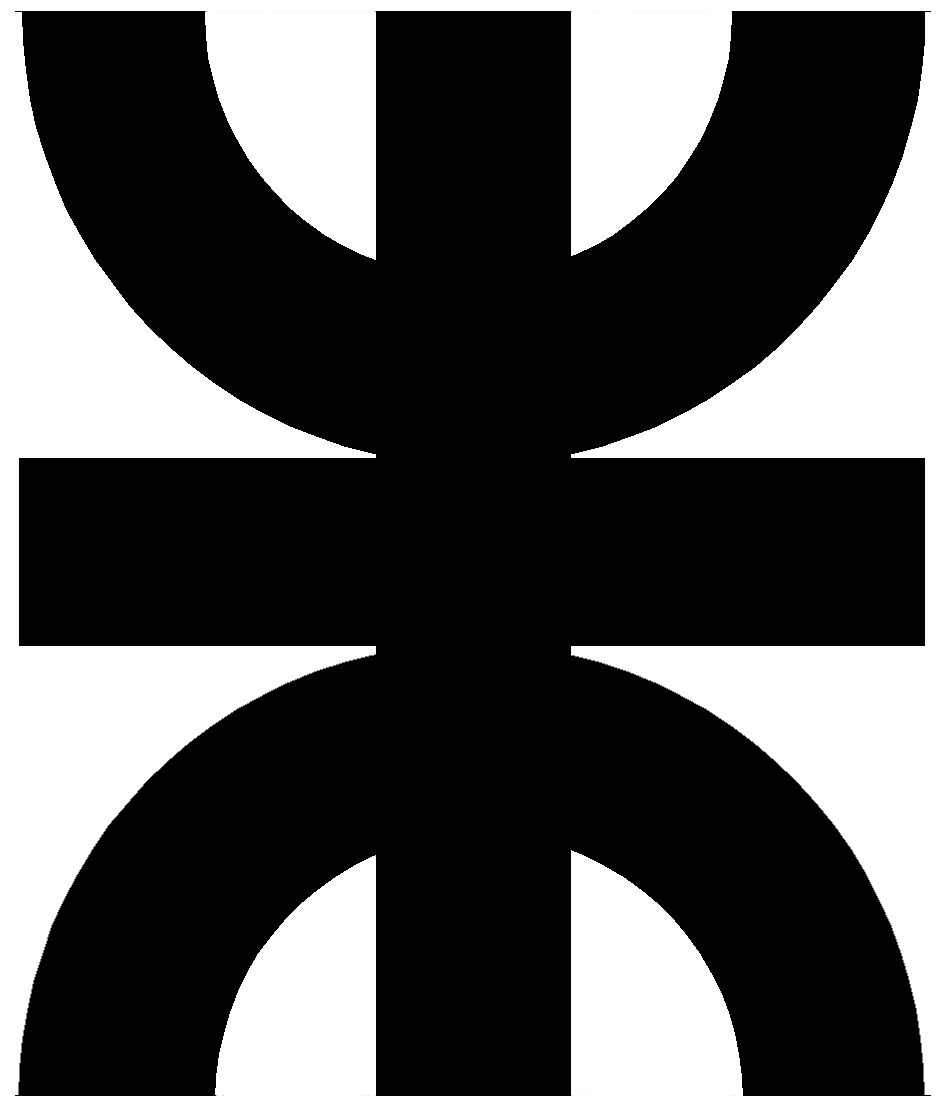
**UNIVERSIDAD TECNOLÓGICA NACIONAL FACULTAD REGIONAL RESISTENCIA**



**SINTAXIS Y SEMÁNTICA DEL LENGUAJE**

**TRABAJO PRÁCTICO INTEGRADOR: Diseño e implementación de Lexer y Parser**

**Carrera:** Ingenería en Sistemas de Información

**Ciclo:** 1° cuatrimestre

**Grupo:** 22.11

**Alumnos:** Almenar Ignacio, Bolo Luciana, Campestrini Gabriela

**Fecha de entrega:** 27/06

**2020**

Índice

[**1.** **Introducción** 2](#_Toc44174292)

[**2.** **Información sobre el intérpre y gramática** 2](#_Toc44174293)

[**3.** **Análisis léxico** 8](#_Toc44174294)

[**4.** **Análisis sintáctico** 10](#_Toc44174295)

[**5.** **Modo de obtención del intérprete** 11](#_Toc44174296)

[**6.** **Modo de ejecución del intérprete** 12](#_Toc44174297)

[**7.** **Ejemplos** 12](#_Toc44174298)

[**8.** **Conclusiones** 15](#_Toc44174299)

[**9.** **Referencias web utilizadas** 16](#_Toc44174300)

1. **Introducción**

El trabajo práctico integrador de la materia Sintaxis y Semántica de los Lenguajes consiste en la elaboración de un intérprete para el lenguaje HTML (lenguaje de marcación de hipertexto). Se dividió en tres partes o entregas, las cuales fueron: diseño de gramática, elaboración de analizador léxico (lexer), y elaboración de analizador sintáctico (parser). Este documento se divide en nueve partes, cada una de las cuales puede ser accedida directamente utilizando el índice.

Para la primera entrega de este trabajo, pusimos en práctica los conocimientos adquiridos en las primeras unidades de la materia sobre gramáticas y lenguajes, y escribimos una gramática para HTML que se ajusta a las consideraciones especificadas en la consigna. También describimos los símbolos terminales o componentes léxicos y los símbolos no terminales. Esto, junto con una breve introducción, fue enviado en formato .pdf. Para la segunda etapa, utilizamos el lenguaje de programación Python y la librería PLY para codificar un analizador léxico. Para entender cómo hacerlo recurrimos a la documentación oficial de Python y de la librería PLY, además de consultar videotutoriales y otras páginas de internet. Por último, en la tercera parte, codificamos un analizador sintáctico, para lo cual hicimos uso de la gramática entregada en la primera parte, la cual simplemente fue adaptada al formato que utiliza la librería PLY.

**Información y requerimientos de software para ejecutar y recompilar el tp:**

Fue programado en lenguaje Python 3.8.2, 32 bits, utilizando la librería PLY versión 3.11, una implementación de lex y yacc para Python.

-Requerimientos de software para ejecutar el intérprete (archivo .exe): se -requiere sistema operativo Windows

-Requerimientos para ejecutar el intérprete desde un archivo de Python: para ejecutar desde el archivo .py, se debe tener instalado Python 3.8, lo cual se puede hacer en Windows (versión Vista o posterior), Mac OS X (para 10.9 y posterior) y Linux (en la mayoría de distribuciones Python viene preinstalado). También se requiere tener instalada la librería PLY versión 3.11.

1. **Información sobre el intérpre y gramática**

Utilizamos mayúsculas para simbolizar no-terminales y minúsculas para los terminales.

|  |  |
| --- | --- |
| **Símbolos terminales o componentes léxicos** | **Símbolos no terminales** |
| <html, charset=  name=, <h1, <h2, </h2, <h3 </h3, <h4, </h4>, <h5, </h5, <h6, </h6, <p,</p, <body, id=, </body, <title , </title, <meta, <section, </section , <div , </div , <em, </em , <mark, </mark, <hr, <br, <strong, </strong, <a, </a, <li, </li, <ol, </ol, <ul, </ul, <img, <tbody, </tbody, <thead, </thead, <tfoot, </tfoot, <tr, </tr, <td, </td, <th, </th, id=, class=, lang=, <head, </head, </html, <table, </table, class=, lang=, charset=, name=, href=, target="\_blank", target="\_parent",  target="\_self, target="\_top, width= height=, border=, alt=, src=, value=, content=, texto, “utf-8”, num\_enatributo, content\_at, , <!doctype html>, url, url\_alt, url\_relativa, leng, type=”1”, type=”a”, type=”A”, type=”i”, type=”I”, > | INICIO, HTML, HEAD, METACHAR, META\_NAME, CONTENT\_AT, TITLE, BODY, E, SECTION, GLOBAL\_ATT, DIV, P, TEXTOENTRE, H1, H2, H3, H4. H5. H6, STRONG, EM, MARK, HR, BR, A, URL. A\_ATT, TARGET, IMG, IMG\_ATT, TABLE, TABLE\_CONTENT, THEAD, TR, TR\_CONT, TD, TH, TBODY, TFOOT, UL, LI, LI\_ATT, OL, TYPE. |

**REGLAS DE PRODUCCIÓN**

**INICIO** <!doctype html> HTML

**HTML** <html lang=”leng” HEAD </html>

**HEAD** <head GLOBAL\_ATT> METACHAR TITLE </head> BODY

| <head> METACHAR TITLE </head> BODY

| <head GLOBAL\_ATT> METACHAR </head> BODY

| <head> METACHAR </head> BODY

**METACHAR** <meta GLOBAL\_ATT charset=”utf-8”>

| <meta charset=”utf-8”>

| <meta GLOBAL\_ATT charset=”utf-8”> META\_NAME

| <meta charset=”utf-8”> META\_NAME

**META\_NAME** <meta GLOBAL\_ATT name=CONTENT\_AT content=CONTENT\_AT> META\_NAME

| <meta GLOBAL\_ATT name=CONTENT\_AT content=CONTENT\_AT>

| <meta name=CONTENT\_AT content=CONTENT\_AT> META\_NAME

| <meta name=CONTENT\_AT content=CONTENT\_AT>

**CONTENT\_AT** content\_at | num\_enatributo

**TITLE** <title GLOBAL\_ATT> texto </title> | <title> texto </title>

**BODY** <body GLOBAL\_ATT> E </body> | <body> E </body>

**E** P | H1 | H2 | H3 | H4 | H5 | H6 | STRONG | EM | BR | MARK | UL | OL | TABLE | SECTION |IMG| DIV| A |HR

**SECTION** <section GLOBAL\_ATT> E </section>

| <section GLOBAL\_ATT> E </section> E

| <section> E </section> E

| <section> E </section>

**GLOBAL\_ATT** class=CONTENT\_AT | id=CONTENT\_AT | class=CONTENT\_AT id=CONTENT\_AT

**DIV** <div GLOBAL\_ATT> E </div>

**|** <div GLOBAL\_ATT> E </div> E

| <div> E </div>

| | <div> E </div> E

**P** <p GLOBAL\_ATT> TEXTOENTRE </p>

| <p GLOBAL\_ATT> TEXTOENTRE </p> E

| <p> TEXTOENTRE </p>

| <p> TEXTOENTRE </p> E

|<p GLOBAL\_ATT> </p>

| <p GLOBAL\_ATT></p> E

| <p> </p>

| <p> </p> E

**TEXTOENTRE** texto E | E texto | texto | E | texto E TEXTOENTRE | E texto TEXTOENTRE

**H1** <h1 GLOBAL\_ATT> texto </h1>

| <h1 GLOBAL\_ATT> texto </h1> E

| <h1> texto </h1>

**|** <h1> texto </h1> E

**|** <h1 GLOBAL\_ATT></h1>

| <h1 GLOBAL\_ATT></h1> E

| <h1></h1>

**|** <h1></h1> E

**H2→** <h2 GLOBAL\_ATT> texto </h2>

| <h2 GLOBAL\_ATT> texto </h2> E

| <h2> texto </h2>

| <h2> texto </h2> E

| <h2 GLOBAL\_ATT></h2>

| <h2 GLOBAL\_ATT></h2> E

| <h2></h2>

| <h2></h2> E

**H3→** <h3 GLOBAL\_ATT> texto </h3>

| <h3 GLOBAL\_ATT> texto </h3> E

| <h3> texto </h3>

| <h3> texto </h3> E

| <h3 GLOBAL\_ATT></h3>

| <h3 GLOBAL\_ATT></h3> E

| <h3></h3>

| <h3></h3> E

**H4→** <h4 GLOBAL\_ATT> texto </h4>

| <h4 GLOBAL\_ATT> texto </h4> E

| <h4> texto </h4>

| <h4> texto </h4> E

| <h4 GLOBAL\_ATT></h4>

| <h4 GLOBAL\_ATT></h4> E

| <h4></h4>

| <h4></h4> E

**H5→** <h5 GLOBAL\_ATT> texto </h5>

| <h5 GLOBAL\_ATT> texto </h5> E

| <h5> texto </h5>

| <h5> texto </h5> E

| <h5 GLOBAL\_ATT></h5>

| <h5 GLOBAL\_ATT></h5> E

| <h5></h5>

| <h5></h5> E

**H6→** <h6 GLOBAL\_ATT> texto </h6>

| <h6 GLOBAL\_ATT> texto </h6> E

| <h6> texto </h6>

| <h6> texto </h6> E

| <h6 GLOBAL\_ATT></h6>

| <h6 GLOBAL\_ATT></h6> E

| <h6></h6>

| <h6></h6> E

**STRONG** <strong GLOBAL\_ATT> TEXTOENTRE </strong>

| <strong> GLOBAL\_ATT> TEXTOENTRE </strong> E

| <strong> TEXTOENTRE </strong>

| <strong> TEXTOENTRE </strong> E

| <strong GLOBAL\_ATT> </strong>

| <strong> GLOBAL\_ATT> </strong> E

| <strong> </strong>

| <strong></strong> E

**EM→** <em GLOBAL\_ATT> TEXTOENTRE </em>

| <em> GLOBAL\_ATT> TEXTOENTRE </em> E

| <em> TEXTOENTRE </em>

| <em> TEXTOENTRE </em> E

| <em GLOBAL\_ATT> </em>

| <em> GLOBAL\_ATT> </em> E

| <em> </em>

| <em></em> E

**MARK→** <mark GLOBAL\_ATT> TEXTOENTRE </mark>

| <mark> GLOBAL\_ATT> TEXTOENTRE </mark> E

| <mark> TEXTOENTRE </mark>

| <mark> TEXTOENTRE </mark> E

| <mark GLOBAL\_ATT> </mark>

| <mark> GLOBAL\_ATT> </mark> E

| <mark> </mark>

| <mark></mark> E

**HR→** <hr GLOBAL\_ATT> | <hr GLOBAL\_ATT> E | <hr> | <hr> E

**BR→** <br GLOBAL\_ATT> | <hb GLOBAL\_ATT> E | <br> | <br>

**A→** <a GLOBAL\_ATT A\_ATT> TEXTOENTRE <a/>

| <a A\_ATT> TEXTOENTRE <a/>

| <a GLOBAL\_ATT A\_ATT> TEXTOENTRE <a/> E

| <a A\_ATT> TEXTOENTRE <a/> E

| <a GLOBAL\_ATT A\_ATT> <a/>

| <a A\_ATT><a/>

| <a GLOBAL\_ATT A\_ATT> <a/> E

| <a A\_ATT><a/> E

**URL→** url | url\_relativa | url\_alt

**A\_ATT→** href=URL TARGET | href=URL

**TARGET→** atributo\_targetblank | atributo\_targetself | atributo\_targetparent | atributo\_targettop

**IMG→** <img GLOBAL\_ATT IMG\_ATT>

| <img IMG\_ATT>

| <img GLOBAL\_ATT IMG\_ATT> E

| <img IMG\_ATT> E

**IMG\_ATT→** alt=CONTENT\_AT src=URL width=num\_enatributo height=num\_enatributo

| alt=CONTENT\_AT src=URL width=num\_enatributo

| alt=CONTENT\_AT src=URL height=num\_enatributo

| alt=CONTENT\_AT src=URL

| src=URL width=num\_enatributo height=num\_enatributo

| src=URL width=num\_enatributo

| src=URL height=num\_enatributo

| src=URL

**TABLE→** <table GLOBAL\_ATT border=num\_enatributo> TABLE\_CONT </table>

| <table border=num\_enatributo>

| <table border=num\_enatributo> TABLE\_CONT </table>

| <table> TABLE\_CONT </table>

| <table GLOBAL\_ATT border=num\_enatributo> TABLE\_CONT </table> E

| <table border=num\_enatributo> TABLE\_CONT </table> E

| <table> TABLET\_CONT </table> E

**TABLE\_CONT→** THEAD | TR |

**THEAD→** <thead GLOBAL\_ATT> TR </thead> TBODY

| <thead> TR </head> TBODY

| <thead GLOBAL\_ATT> TR </thead> TFOOT

| <thead> TR </head> TFOOT

**TR→** <tr GLOBAL\_ATT> TR\_CONT </tr>

| <tr GLOBAL\_ATT> TR\_CONT </tr> TR

| <tr> TR\_CONT </tr>

| <tr> TR\_CONT </tr> TR

**TR\_CONT→** TH | TD | TH TR\_CONT | TD TR\_CONT

**TD→** <td GLOBAL\_ATT> TEXTOENTRE </td>

| <td> TEXTOENTRE </td>

| <td GLOBAL\_ATT> </td>

| <td> </td>

**TH→** <th GLOBAL\_ATT> TEXTOENTRE </th>

| <th> TEXTOENTRE </th>

| <th GLOBAL\_ATT> </th>

| <th> </th>

**TBODY→** <tbody GLOBAL\_ATT> TR </tbody>

| <tbody> TR </tbody>

| <tbody GLOBAL\_ATT> TR </tbody> TFOOT

| <tbody> TR </tbody> TFOOT

**TFOOT→** <tfoot GLOBAL\_ATT> TR </tfoot>

| <tfoot TR </tfoot>

**UL→** <ul GLOBAL\_ATT> LI </ul>

| <ul> GLOBAL\_ATT> LI </ul> E

| <ul> LI </ul>

| <ul> LI </ul> E

**LI→** <li GLOBAL\_ATT> TEXTOENTRE </li>

| <li GLOBAL\_ATT> LI\_ATT TEXTOENTRE </li>

| <li> TEXTOENTRE </li>

| <li LI\_ATT> TEXTOENTRE </li>

|<li GLOBAL\_ATT> TEXTOENTRE <li> LI

| <li GLOBAL\_ATT LI\_ATT> TEXTOENTRE </li> LI

| <li> TEXTOENTRE </li> LI

| <li LI\_ATT> TEXTOENTRE </li> LI

| <li GLOBAL\_ATT> </li>

| <li GLOBAL\_ATT LI\_ATT> </li>

|<li> </li>

|<li LI\_ATT> </li>

| <li LI\_ATT></li>

| <li GLOBAL\_ATT> <li> LI

| <li GLOBAL\_ATT LI\_ATT> </li> LI

| <li> </li> LI

| <li LI\_ATT> </li> LI

**LI\_ATT→** value=num\_enatributo

**OL→** <ol GLOBAL\_ATT TYPE LI </ol>

| <ol GLOBAL\_ATT> LI </ol>

| <ol TYPE> LI </ol>

| <ol> LI </ol>

| <ol GLOBAL\_ATT TYPE LI </ol> E

| <ol GLOBAL\_ATT> LI </ol> E

| <ol TYPE> LI </ol> E

| <ol> LI </ol> E

**TYPE→** type=”1”| type=”a”| type=”A”| type=”i”| type=”I”|

1. **Análisis léxico**

El analizador léxico fue programado en Python 3.8 y para la construcción de este hicimos uso de las librerías ply, re, codecs, os, tkinter y sys. Definimos los tokens o componentes léxicos deseados, los cuales serán parametros de busqueda del analizador a la hora de ejecutarlo. Para muchos tokens, utilizamos funciones que permiten especificar una expresión regular.

A la hora de definir las funciones con expresiones regulares que cada uno de estos tokens contiene, se usó un orden jerárquico de tal forma que dé prioridad a algunos tokens a la hora de realizar el análisis.

En muchas funciones, se define el formato general de una “categoría de token”, y se la vincula con el diccionario “reserved”, de manera que al encontrar un string con ese formato, busca en el diccionario el nombre al que está asociado. Por ejemplo:

def t\_tag\_format(t):

r'[ ]\*<[ ]\*[a-zA-Z/0-9]+[ ]\*'

while ' ' in t.value:

new = t.value.replace(" ", "")

t.value = new

t.type = reserved.get(t.value.lower(),'tag\_format')

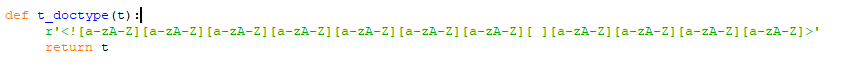
if t.type == 'tag\_format':

print('Error. Tag mal escrito:', t.value)

else:

return t

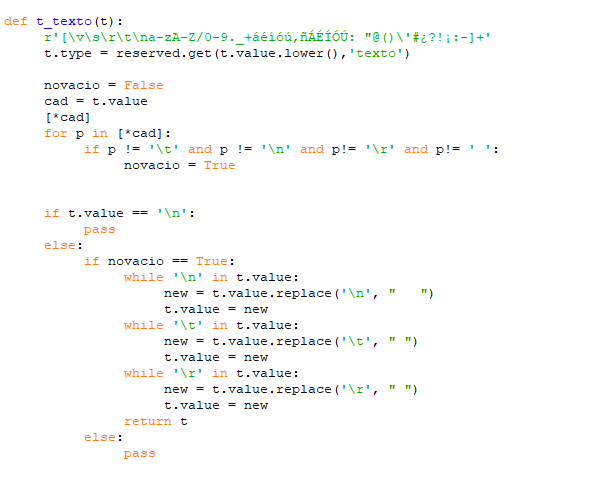
Si encuentra un string que coincide con la expresión regular propia de un tag pero no se encuentra en el diccionario reserved, imprime un mensaje de error avisando que el tag está mal escrito. Esto sucederá en el caso de escribir por ejemplo <htm, <k, etc.

A su vez, algunas funciones no hacen referencia a una “categoría de token”, sino que directamente utilizan una expresión regular para definir un token específico. Se utilizó una función con expresión regular porque los tokens sueltos causaban algunos problemas. Este el caso de t\_doctype:

Cabe mencionar que escribimos la expresión regular utilizando [a-z-AZ] varias veces porque no pudimos encontrar una opción para que automáticamente se ignore la sensibilidad a mayúsculas y minúsculas.

Por último, existen aquellas funciones que definen las expresiones regulares para lo que podríamos llamar “tokens variables”, como texto, contenido de atributos, urls, etc. Tenemos 3 tokens distintos para el contenido de atributos: num\_enatributos, que solo acepta número enteros, atributo\_type (para el atributo propio de listas ordenadas) y content\_at, que acepta letras, números enteros y espacios. Para las URLs, tenemos también tres funciones para cubrir todos los casos: url, que es la más general, url\_alt, y url\_relativa.

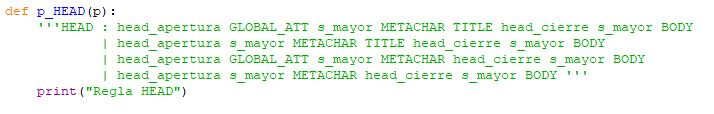
Definimos la función “error” para el caso en que existiese algún carácter que no coincidiese con ningún token definido. También hicimos que se ignoren los saltos de líneas(\n), carriage returns(\r) y tabs (\t), ya que no resultan relevantes para el análisis léxico. En principio, dichos caracteres se identifican como texto. Sin embargo, la función de texto está programada de manera tal que si el token consiste sólo de esos caracteres, sin ningún símbolo relevante como letras o números, se lo saltea. Y en caso de que un texto que queremos mostrar los contenga, se los reemplaza por un espacio, para que no se impriman como “\n”, “t\”, etc, y hacerlo más ameno a la vista:



1. **Análisis sintáctico**

Para la creación del parser usamos de igual manera el lenguaje Python 3.8, y utilizamos las librerías previamente importadas para el Lexer. Importamos los tokens del Lexer previamente programado con el comando ‘*from Lexer import tokens*’.

Posteriormente definimos funciones (con el formato *def NO\_TERMINAL(p)* ), utilizando una función para cada no terminal previamente definido en la gramática. Así, definimos los no terminales INICIO, HTML, HEAD, METACHAR, META\_NAME, CONTENT\_AT, TITLE, BODY, E, SECTION, GLOBAL\_ATT, DIV, P, TEXTOENTRE, H1, H2, H3, H4. H5. H6, STRONG, EM, MARK, HR, BR, A, URL. A\_ATT, TARGET, IMG, IMG\_ATT, TABLE, TABLE\_CONTENT, THEAD, TR, TR\_CONT, TD, TH, TBODY, TFOOT, UL, LI, LI\_ATT, OL, TYPE. Como terminales usamos los componentes léxicos previamente establecidas en el lexer, respetando el nombre que anteriormente le habíamos dado. Para diferenciar las producciones que un no terminal definido en la función podía poseer usamos, de igual manera que usamos en la práctica de la materia, el carácter “|”. Al final de la función imprimimos por pantalla el nombre del no terminal usado.

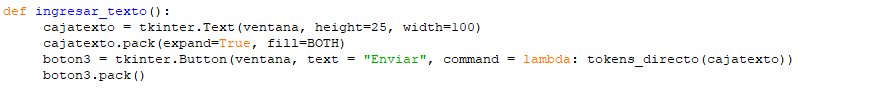


Luego de definir todos los no terminales usados, definimos una función p\_error que nos indicará cuando estamos cometiendo un error sintáctico a la hora del análisis y en que linea del archivo html o txt se encuentra.

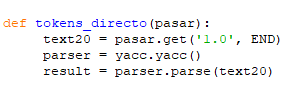


También utilizamos las siguientes funciones para incorporar la interfaz gráfica:

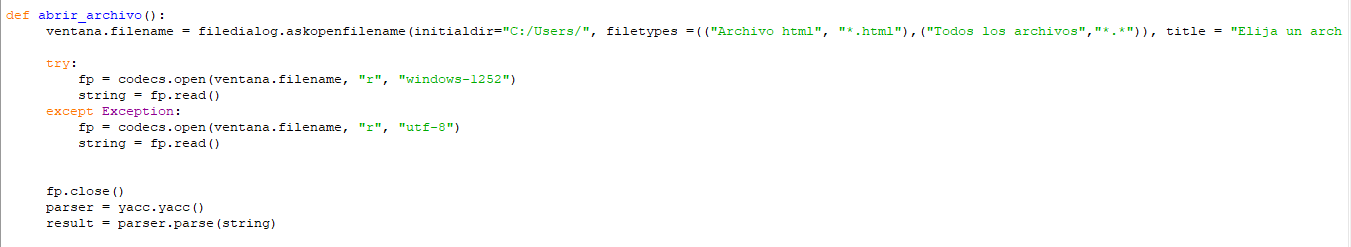
**ingresar\_texto():** abre una ventana con una caja de texto en la cual el usuario puede ingresar el texto a ser analizado. Es la función que envía cajatexto a tokens\_directo(pasar)



**tokens\_directo(pasar):** recibe como parámetro cajatexto y asigna el contenido que el usuario ingresa en la caja de texto (cajatexto.get) a una nueva variable llamada text20. Luego ordena realizar el análisis sintáctico del string guardado en text20.

****

**abrir\_archivo():** abre un buscador de archivos donde el usuario deberá seleccionar de forma manual el .txt o .html a analizar



1. **Modo de obtención del intérprete**

Se tienen los siguientes archivos en la entrega:

**entrega\_lexer.py**: código fuente de donde se generó el ejecutable del lexer (en carpeta src)

**entrega\_lexer.exe** ejecutable del lexer (en carpeta bin)

**entrega\_parser.py**: código fuente de donde se generó el ejecutable del parser (en carpeta src)

**entrega\_parser.exe** ejecutable del parser (en carpeta bin)

Los archivos parser\_entrega y lexer\_entrega deben estar en la misma carpeta si tenemos Python instalado y queremos probar el intérprete desde los archivos fuente. Hay que tener los archivos de la librería PLY en esa misma carpeta también. Se debe prestar atención a los comentarios dejados en el archivo entrega\_lexer, ya que ahí se explica que partes del código deben cambiarse para ejecutar el parser o el lexer de manera separada.

En cuanto a los archivos .exe, estos se puden ejecutar directamente y por separado sin problemas. Ante dudas sobre el uso, se puede consultar el manual de usuario en la carpeta doc.

1. **Modo de ejecución del intérprete**

Se tienen 2 opciones a la hora de elegir lo que se desea analizar:

* **Ingreso directo:** esta opción abre un cuadro de texto donde el usuario debe insertar el texto a analizar.
* **Buscar archivo:** abre una ventana mediante la cual el usuario puede buscar en la pc el documento de texto o html a analizar.

1. **Ejemplos**

Ejemplo 1 de un archivo que el intérprete puede analizar:

<!doctype html>

<html lang="es" >

<head class="estoesunaclase">

<meta charset="utf-8">

<title> Esto un titulo que va en head </title>

</head>

<body>

<h1> Dentro del body podemos tener un heading </h1>

<h2> Pero dentro del heading solo puedo ir texto </h2>

<p> Dentro de la etiqueta de párrafo <strong> pueden ir otras etiquetas </strong> </p>

<hr>

<div id="identified" >

<h3> Este es otro tipo de heading </h3>

<ul>

<li> Lorem Ipsum </li>

<li id="listaid" value="1" > Algo </li>

</ul>

<ol type="a" >

<li> Esto está en una lista ordenada </li>

<li> La lista se ordena alfabéticamente </li>

</ol>

<ol type="i">

<li> Los elementos de lista pueden no tener ningun texto adentro </li>

<li></li>

<li value="5"> También puede tener atributos </li>

</ol>

<img alt="esto se muestra si la imagen no se ve" src="gatito.png" >

<img src="https://upload.wikimedia.org/wikipedia/commons/6/61/HTML5\_logo\_and\_wordmark.svg">

<em> como vemos, el atributo alt puede no estar </em>

<hr>

</div>

<mark> Ahora una tabla que muestra las materias de segundo año de Ingeniería en Sistemas </mark>

<table border="2">

<thead>

<tr>

<th> Materia </th>

<th> Cuatrimestre </th>

</tr>

</thead>

<tbody>

<tr>

<td> Sintaxis y Semántica de los Lenguajes </td>

<td> Primero </td>

</tr>

<tr>

<td> Física II </td>

<td> Primero </td>

</tr>

<tr>

<td> Análisis de Sistemas </td>

<td> Anual </td>

</tr>

<tr>

<td> Paradigmas de Programación </td>

<td> Segundo </td>

</tr>

<tr>

<td> Química </td>

<td> Segundo </td>

</tr>

<tr>

<td> Análisis Matemático II </td>

<td> Anual </td>

</tr>

</tbody>

</table>

</body>

</html>

Otro ejemplo:

<!doctype html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="palabras claves" content="código HTML, tags, en sintaxis">

<title>Prueba para testear el parser</title>

</head>

<body>

<SECTION class="intro">

<h3> Alumnos de ISI B </h3>

<h2> Los integrantes del grupo son </h2>

<strong> ALMENAR IGNACIO <br>

BOLO LUCIANA <br>

CAMPESTRINI LUCIANA <br>

</strong>

<em> El lenguaje utilizado fue Python 3.8 </em>

<h3> Modo de obtención del Parser </h3>

<p> Si bien se nos presentaron dificultades a la hora de programar el parser <br>

pudimos resolverlos mediante consultas a los profesores y guiandonos por paginas en internet </p>

<h2> ¿Por qué usamos Python cómo lenguaje de programación? </h2>

<p> Python fue utilizado como lenguaje ya que nos parecio un lenguaje bastante sencillo <br>

y uno de los intengrantes del grupo ya tenia conocimientos sobre este </p>

<hr>

<section id="caracteristicas">

<h3 id="elab">Pros y contras</h3>

<ul>

<li>

<strong> Pros </strong>

<p> en los pro de nuestro parser encontramos que es bastante sencillo de utilizar <br>

la interfaz gráfica es agradable, y analiza todos los componentes de forma correcta <br>

también, cuenta con opción de ingreso directo o insertar el texto de forma manual </p>

</li>

</ul>

</section>

<hr>

<section id="conclusiones">

<h3 id="elab">Conclusión sobre el TFI</h3>

<ul>

<li>

<strong> SOBRE EL TFI </strong>

<p> En el desarrollo de este Trabajo Final Integrador aprendimos varios conceptos estudiados en la materia <br>

tales como <strong> el uso de terminales, no terminales y producciones </strong> <br>

también hicimos uso de conocimientos previamente estudiado tales como <strong> funciones, variables </strong> etc </p>

</li>

</ul>

</section>

</section>

</body>

</html>

1. **Conclusiones**

A la hora de empezar a programar el Lexer elegimos el lenguaje de programación Python 3.8 porque habíamos escuchado buenos comentarios sobre el mismo y un integrante del grupo tenía conocimientos. El intérprete programado funciona correctamente, se pudieron analizar de forma exitosa los ejemplos propuestos. En la codificación de este incorporamos muchos conceptos aprendidos tanto en la práctica como en la teoría de la materia, tales como la creación de expresiones regulares, el uso de terminales y no terminales y la construcción de producciones gramaticales en sí. También pudimos hacer uso de otros conocimientos previamente estudiados en materias de primer año tal como la definición de funciones y la estructura de éstas, como importar librerías externas, y la definición de variables (palabras reservadas, tokens, etc).

Podemos concluir mediante la ejecución de este Trabajo Práctico Integrador que incorporamos de manera exitosa los conocimientos necesarios para poder entender de forma correcta el análisis léxico y sintáctico. Además, pudimos familiarizarnos con HTML y el lenguaje de programación Python. En especial las integrantes que no teníamos experiencia previa con mismos notamos que aprendimos mucho. Creemos que esto forma parte del conocimiento de tecnologías y lenguajes de programación que debe tener un Ingeniero en Sistemas, y que resulta útil.

Como puntos débiles podemos mencionar que:

-No se imprime ningún texto al final diciendo, por ejemplo, “el código es sintácticamente/léxicamente correcto” en caso de que no haya ningún error.

-Al usar la interfaz gráfica, si se realizan ciertas cosas como presionar el botón de abrir archivo y luego cerrar la ventana sin haber seleccionado ningún archivo, se genera un error. Tal vez sería mejor que este no se muestre.

-En los ejecutables se muestran advertencias porque quedaron algunos tokens declarados que no se usaron.

-No permite terminar la ejecución al presionar ctrl + d.

Como puntos fuertes podemos mencionar que:

-Tiene interfaz gráfica.

-Su uso es bastante sencillo.

-Permite ver la línea y posición en que ocurrió un error.

-Reconoce tildes y otros caracteres propios del idioma español.

-Lee tanto archivos html como txt.

1. **Referencias web utilizadas**

<https://docs.python.org/> (para consultas sobre Python)

<https://www.dabeaz.com/ply/ply.html> (para consultas sobre la librería PLY)

<https://stackoverflow.com/> (para consultas sobre Python y PLY)

<https://www.w3schools.com/> (para consultas sobre Python)

<https://www.geeksforgeeks.org/> (para consultas sobre Python)