**Trabajo Práctico Integrador:**

**Diseño e implementación de Lexer y Parser**

**3ra Entrega Presentación del Parser**

**Integrantes:**

* Brandan, Ian
* Dor, Augusto
* Marighetti, Jose Juan
* Sodja, Ivo

**Asignatura:** Sintaxis y Semántica de los Lenguajes.

**Carrera:** Ingeniería en Sistemas de Información.

**Primer Cuatrimestre**

**Curso académico:** 2024

**Universidad:** Universidad Tecnológica Regional (FRRe)

**Lugar y fecha:** Argentina, Chaco, Resistencia; 28/04/2024

ÍNDICE

[**Introducción 4**](#_3d4mw1vkckdg)

[**Matriz de habilidades 4**](#_kq50t4hi7t0n)

[**Alianza de grupo 4**](#_mepj86n4tn51)

[**Identidad 5**](#_9dio1wlsw24l)

[**Gramática 6**](#_s861zxwm4bfb)

[**Lexer 11**](#_xi7aymqsnzw6)

**Parser………………………………………………………………………………………………15**

# Introducción

Bienvenidos/as, este es el primer documento del trabajo práctico integrador de la materia “Sintaxis y Semántica de los Lenguajes”. El mismo, cuenta con una portada, un índice, esta parte (la introducción), la conformación del grupo y su matriz de habilidades, y, por último, la principal tarea de esta entrega, la gramática que utilizaremos. La última, sufrió varios cambios a medida que hemos ido trabajando en ella, podríamos decir que fue evolucionando junto con los conocimientos que adquirimos en clases y con las consultas planteadas a la docente tutora asignada.

# Matriz de habilidades

|  | Inglés | Programar | Edición | Documentación | Cebar  mates | Idear  soluciones |
| --- | --- | --- | --- | --- | --- | --- |
| Ian Brandan | ★ | ◯ | ★ | ★ | ◯ |  |
| Augusto Dor | ◯ | ★ |  | ◯ |  | ★ |
| Juan Marighetti | ★ | ◯ |  |  | ★ | ◯ |
| Ivo Sodja | ★ | ◯ | ◯ | ◯ |  | ★ |

# Alianza de grupo

Como grupo contamos con una comunicación constante, realizamos reuniones virtuales dos veces por semana como mínimo, realizando reuniones intensivas durante los fines de semana, las cuales coordinamos por la plataforma discord.  
 En cada etapa del trabajo práctico integrador tenemos como regla cumplir los requisitos antes de la fecha de entrega, así contamos con tiempo para poder realizar una exhaustiva revisión del trabajo realizado.

# Identidad

Asignamos la siguiente imagen que representará la identidad del grupo, la misma nace del nombre que elegimos “Uber: Sin Taxis”.



# Gramática

La gramática se encuentra descrita en el documento `Gramática.g4`. Es parte vital de este proyecto ya que describe los componentes o tokens que el lexer debe identificar en el texto de entrada.

Lo último en realizarse fue la sintaxis válida de una URL, la cual se encontraba descrita en el pdf brindado por la cátedra. En nuestra gramática, hemos definido el token URL para reconocer direcciones URL,ya que en un principio para agilizar el proceso se definió a la url como un string.

Para mayor claridad, a pesar de encontrarse en nuestro directorio, se describen a continuación las reglas principales.

Descripción de cada regla de la gramática libre de contexto a utilizar

json: Regla de inicio que puede ser un objeto JSON (t\_json) o espacios en blanco (WS)

t\_json: Define un objeto JSON, comenzando con { y terminando con }

t\_contenido: Define el contenido de un objeto JSON, que puede contener una lista de empresas (t\_empresas), una versión (t\_version), y/o una firma digital (t\_firma\_digital)

*Tokens que definen campos:*  
t\_version

t\_firma\_digital

t\_empresas

t\_ingresos\_anuales

t\_nombre\_empresa

t\_fundacion

t\_pyme

t\_link

t\_direccion

t\_calle

t\_ciudad

t\_pais

t\_tipo\_direccion

t\_nombre

t\_edad

t\_cargo

t\_salario

t\_activo

t\_fecha\_contratacion

t\_proyectos

*Tokens que definen lista donde cada elemento está separado por comas:*

t\_listaempresas

t\_listadepartamentos

t\_listasubdepartamentos

t\_listaempleados

t\_listaproyectos

*Tokens que definen el campo donde se encuentra una lista:*

t\_departamentos

t\_subdepartamentos

*Tokens que definen un objeto Json :*

t\_empresa

t\_departamento

t\_subdepartamento

t\_empleado

t\_proyecto

*Tokens que definen el contenido de un objeto JSON :*

t\_contenidoempresa

t\_contenidodepartamento

t\_contenidosubdepartamentos

t\_contenidoempleado

t\_contenidoproyecto

t\_contenidoempleado

Símbolos de la gramática

*Símbolos terminales (componentes léxicos)*

EMPRESAS: '"empresas":';

VERSION: '"versión":' | '"version":';

FIRMA\_DIGITAL: '"firma\_digital":';

INGRESOS\_ANUALES: '"ingresos\_anuales":';

NOMBRE\_EMPRESA: '"nombre\_empresa":';

FUNDACION: '"fundación":' | '"fundacion":';

PYME: '"pyme":';

LINK: '"link":';

DIRECCION: '"dirección":' | '"direccion":';

EDAD: '"edad":';

CARGO: '"cargo":';

TIPO\_CARGO: '"Product Analyst"' | '"Project Manager"' | '"UX designer"' | '"Marketing"' | '"Developer"' | '"Devops"' | '"DB admin"' ;

SALARIO: '"salario":';

ACTIVO: '"activo":';

FECHA\_CONTRATACION: '"fecha\_contratación":' | '"fecha\_contratacion":';

FECHA\_INICIO: '"fecha\_inicio":';

ESTADO: '"estado":';

TIPO\_ESTADO: '"To do"' | '"In progress"' | '"Canceled"' | '"Done"' | '"On hold"' ;

FECHA\_FIN: '"fecha\_fin":';

CALLE: '"calle":';

CIUDAD: '"ciudad":';

PAIS: '"país":' | '"pais":';

DEPARTAMENTOS: '"departamentos":';

EMPLEADO\_LISTA: '"empleados":';

SUBDEPARTAMENTOS\_LISTA: '"subdepartamentos":';

PROYECTOS\_LISTA: '"proyectos":';

NOMBRE: '"nombre":';

JEFE: '"jefe":';

ABRO\_LLAVE: '{';

CIERRO\_LLAVE: '}';

ABRO\_CORCHETE: '[';

CIERRO\_CORCHETE: ']';

COMA: ',';

NULL: 'null';

// TIPOS

TRUE: 'true';

FALSE: 'false';

INT: [0-9]+;

DIGIT: [0-9];

FLOAT: DIGIT+ '.' DIGIT DIGIT;

DATE: '"' ('19' DIGIT DIGIT | '20' DIGIT DIGIT) '-' ('0'[1-9] | '1'[0-2]) '-' ('0'[1-9] | [12][0-9] | '3'[01]) '"';

URL\_STRING: [a-zA-Z0-9-.~#]+;

PROTOCOLO: 'https' | 'http';

DOMINIO: URL\_STRING ('.' URL\_STRING)+;

PUERTO: ':' INT;

RUTA: '/' URL\_STRING ( '/' URL\_STRING )\*;

URL: '"' PROTOCOLO '://' DOMINIO ( PUERTO )? ( RUTA )? '"';

STRING: '"' (~["\\] | ESC)\* '"';

WS: [ \t\r\n]+ -> skip;

fragment ESC: '\\' [\\/bfnrt];

Símbolos no terminales

Cabe aclarar que están agrupados sin saltos de línea por practicidad solo en este documento.

json, t\_json, t\_contenido, t\_version, t\_firma\_digital, t\_empresas, t\_listaempresas, t\_empresa, t\_contenidoempresa, t\_ingresos\_anuales, t\_nombre\_empresa, t\_fundacion, t\_pyme, t\_link, t\_direccion, t\_tipo\_direccion, t\_calle, t\_ciudad, t\_pais, t\_departamentos, t\_listadepartamentos, t\_departamento, t\_contenidodepartamento, t\_nombre, t\_jefe, t\_subdepartamentos, t\_listasubdepartamentos, t\_subdepartamento, t\_contenidosubdepartamentos, t\_empleados, t\_listaempleados, t\_empleado, t\_contenidoempleado, t\_edad, t\_cargo, t\_salario, t\_activo, t\_fecha\_contratacion, t\_proyectos, t\_listaproyectos, t\_proyecto, t\_contenidoproyecto, t\_fecha\_inicio, t\_estado, t\_fecha\_fin

Reglas de producción de la gramática

Las reglas de producción de la gramática describen cómo se pueden combinar los diferentes elementos terminales y no terminales de la gramática para formar frases o estructuras válidas dentro del lenguaje definido por la gramática. Se encuentran en el archivo gramática g4 y derivan de las Características de lenguaje JSON otorgadas por la cátedra donde se nos indico que atributo puede contener cada objeto por lo tanto, sólo presentamos aquí las primeras reglas de producción

json : t\_json | WS

t\_json : ABRO\_LLAVE t\_contenido CIERRO\_LLAVE

t\_contenido : t\_empresas ( COMA t\_version)? ( COMA t\_firma\_digital)?

t\_version : VERSION STRING

t\_firma\_digital : FIRMA\_DIGITAL STRING

t\_empresas : EMPRESAS ABRO\_CORCHETE t\_listaempresas CIERRO\_CORCHETE

t\_listaempresas : t\_empresa ( COMA t\_listaempresas)?

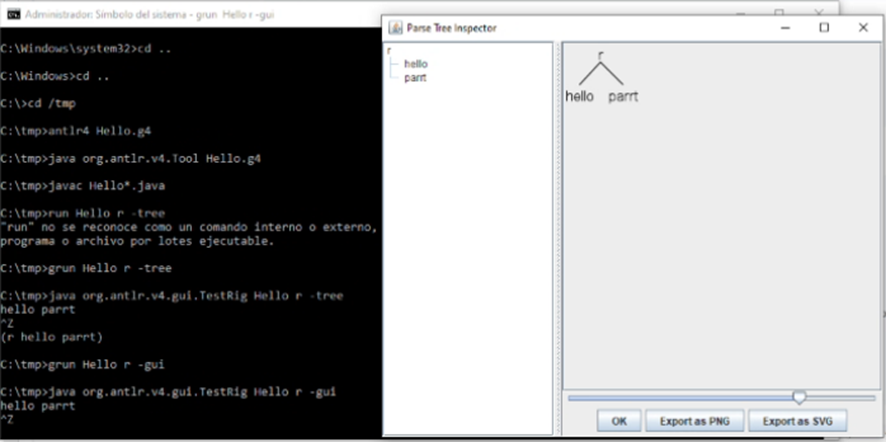
t\_empresa : ABRO\_LLAVE t\_contenidoempresa CIERRO\_LLAVE

t\_contenidoempresa : t\_nombre\_empresa COMA t\_fundacion COMA (t\_direccion COMA )? t\_ingresos\_anuales COMA t\_pyme COMA (t\_link COMA )? t\_departamentos

# Lexer

Para esta parte del trabajo, una vez definida nuestra gramática Lexer, elegimos utilizar la librería “ANTLR4”.

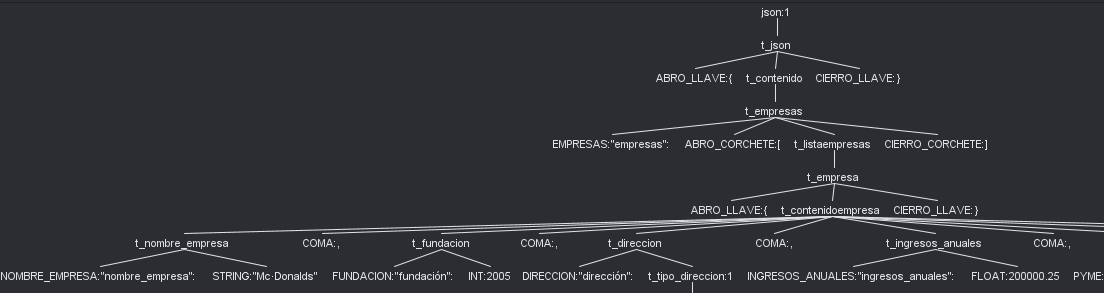
Esto era un nuevo desafío, puesto que buscando en Youtube había muy escaso material y pocos tutoriales sobre el uso de la librería. Pero empezamos descargándola para utilizarla en el lenguaje “Python”. En este paso, tuvimos que, cada uno, agregar al Path de variables del sistema la ruta del archivo. Llegado este punto, intentamos probar cómo funcionaba con un ejemplo proporcionado por el foro donde fue publicada la librería. Luego de varios intentos pudimos ver un árbol sintáctico o *parse tree* que indicaba que estábamos en buen camino para la consigna del proyecto integrador (parte final).



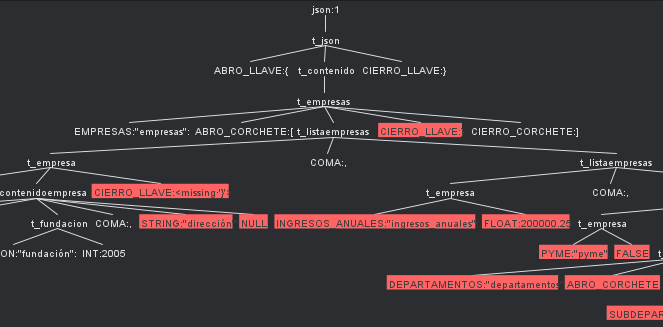
Acá empiezan los problemas porque no sabíamos cómo darle todo el formato del archivo .json que está como ejemplo en el aula y, como dije anteriormente, era difícil encontrar información sobre esta librería para avanzar con el proyecto. Evaluamos cambiar de librería e intentar conseguir alguna que tenga más información. Justo en ese momento, uno de nuestros integrantes, Augusto, encontró un foro donde un usuario contaba un proyecto que había creado usando ANTLR y Python, justo lo que necesitábamos. Ahondando un poco más, descubrió que había una forma “fácil” de usar la librería y era utilizando PyCharm (un entorno de desarrollo integrado (IDE) diseñado específicamente para el desarrollo en Python) con un Plugin de la misma librería.

Nota:<https://medium.com/@ab.rhmn97/build-an-api-query-language-with-antlr-in-python-7313dba222e7>

Esto nos motivó ya que esta extensión aligeraba mucho nuestra tarea e investigación. Probando ejemplos, en la terminal mostraba todo lo que estaba bien en el texto y marcaba los errores para que puedan ser corregidos.

Para ejemplificar, mostramos el parse tree que proporciona esta extensión ante un archivo json correcto:  


Y la notificación al encontrar un error, por ejemplo, tipeando una comilla simple demás en donde iría el string dirección:



Esta extensión nos ayuda a concluir como el lexer procesa el archivo de entrada .json dividiéndolo en tokens.

Finalmente una vez entendimos que habíamos cumplido con los requerimientos necesarios para esta etapa se buscó que nuestros archivos en formato ‘.py’ se pudiese ejecutar en cualquier computadora, inclusive si esta no cuenta con python o las librerías que hemos utilizado, para también cumplir con el formato de entrega del proyecto acordado.

Se encontró un paquete llamado pyinstaller, el cual se usa para empaquetar el código Python en aplicaciones ejecutables independientes para varios sistemas operativos.

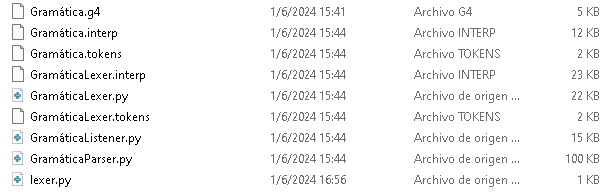




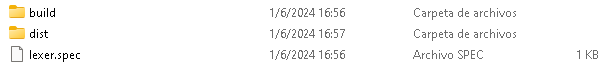
Por último se organizó el directorio en formato requerido,quedándonos de la siguiente forma.  


En la carpeta prueba encontramos el archivo de prueba brindado por la cátedra para testear nuestro lexer  


En src, nuestro archivo fuente, se encuentran los documentos utilizados para el funcionamiento de nuestra gramática, tales como el lexer y el parser.

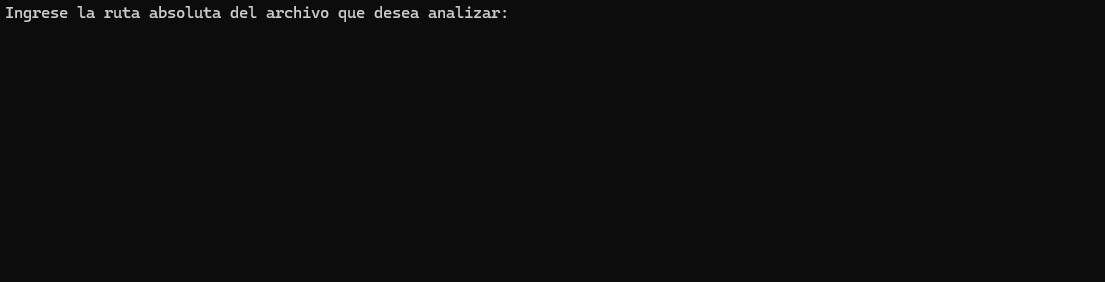


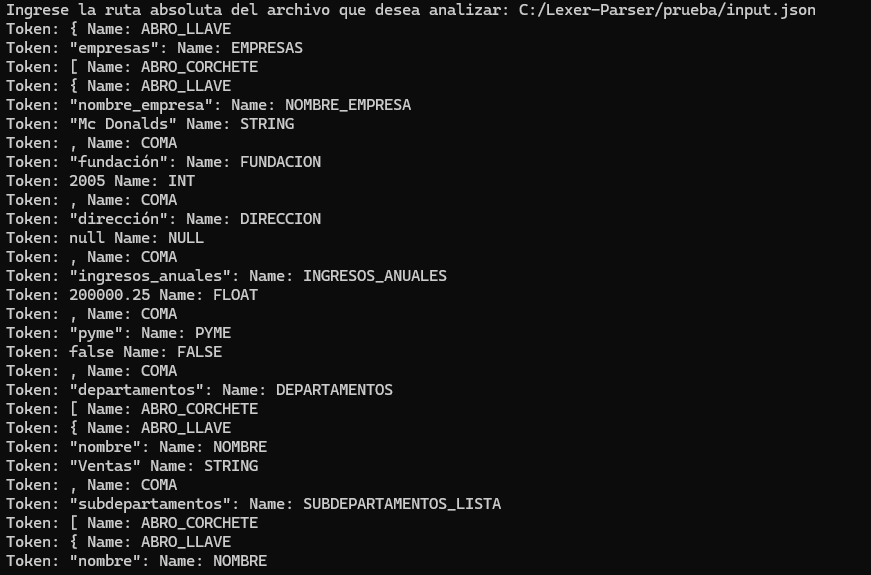
En bin se encuentra nuestro ejecutable, dentro de la carpeta dist.

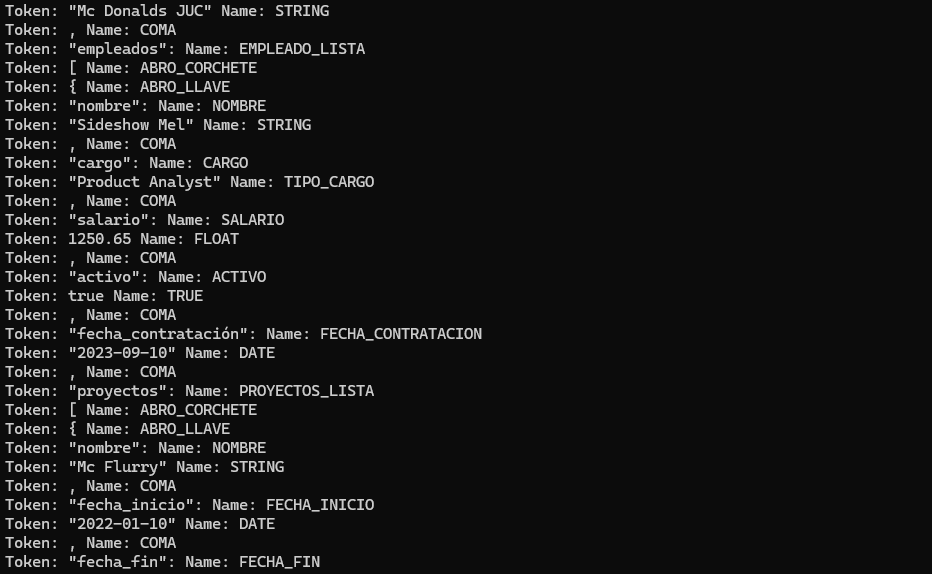


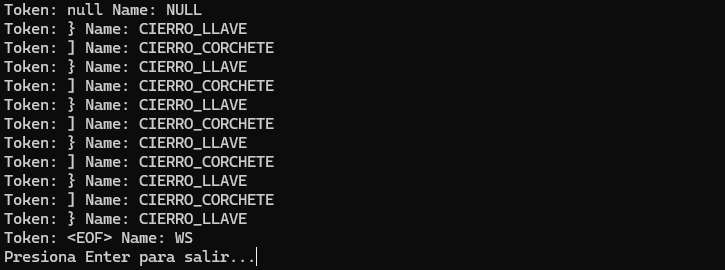


Creemos que es necesario dar un breve tutorial del funcionamiento de este archivo ejecutable

Al entrar al mismo, es requerido ingresar la ruta absoluta del archivo a analizar, es decir, su dirección, desde el disco c hasta su ubicación.  






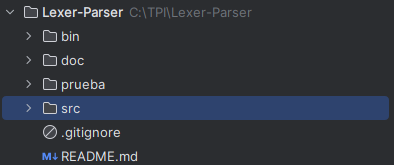


Como puede verse el programa al ser ejecutado reconoce cada token y devuelve el nombre que le ha sido asignado al mismo, una vez muestra toda esta información el usuario puede salir de la consola. Finalizando así la explicación de la implementación del lexer.

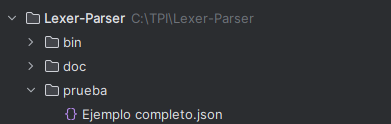
# Parser

**• Análisis sintáctico**: Descripción del archivo del parser utilizado para definir la gramática libre de contexto.

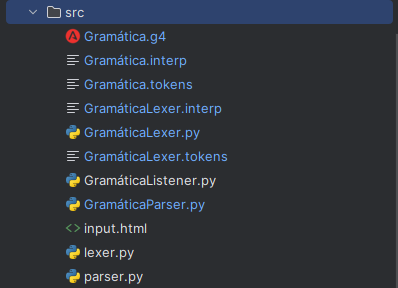
Para comenzar hicimos grandes cambios en nuestro directorio.



La carpeta “madre” es Lexer-Parser. Las entradas que ejecutamos a pos de comprobar nuestro trabajo se encuentran en: Lexer-Parser > Prueba:

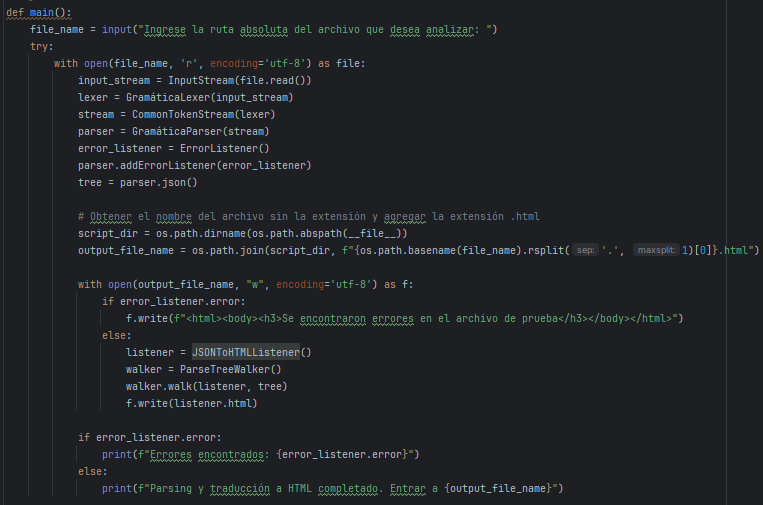


Nuestro analizador sintáctico se encuentra dentro de la carpeta src, junto a él se crearon carpetas para trabajar en separado las distintas funciones. La carpeta gramática.g4 es aquella que se realizó en la etapa anterior.



En primera instancial código de nuestro parser importa librerías y otras carpetas tales como GramáticaLexer, GramáticaParser, GramáticaListener.

Define clases tales como JSONToHTMLListener, el cual trata errores del mismo y cuenta dentro suyo funciones para dicha tarea entonces salteando esta definición de clases podemos decir que la parte del código la cual toma mayor relevancia es aquella donde se solicita la entrada al usuario y se invocan las funciones a utilizar.



**• Modo De Ejecución**

Una vez el programa es ejecutado por medio del archivo.exe, se abrirá la consola la cual pide al usuario que ingrese la ruta absoluta del archivo de entrada a analizar y donde según resulte el análisis del archivo va a listar los errores del mismo por pantalla o lo va a traducir a un formato .HTML, entregándonos la ruta de la carpeta donde se encuentra el mismo, este archivo se encuentra diseñado para ser agradable a la vista de nuestro usuario.

**• Conclusiones:**

Como grupo concluimos que la dificultad de un trabajo de esta complejidad es grande. Herramientas como la matriz de identidad sirven como una guía inicial. Luego en el desarrollo la comunicación es importante por lo cual la plataforma GitHub ha sido determinante para el éxito de este proyecto.

Hemos tenido contratiempos, ya que la librería que creíamos más adecuada resultó ser bastante nueva, por lo cual no se encontraba tanta información o guías disponibles para otras alternativas tales como ply.

A pesar de esto y gracias a la colaboración de nuestra tutora la cual vía mail nos ha solucionado el estancamiento en pequeños baches del proceso, tales como problemas con la url o la presentación en html.

Consideramos que hemos aprendido y adquirido tanto habilidades y conocimientos con esta actividad.

• Bibliografía o referencias web.

Utilizamos blogs y páginas para entender nuestra biblioteca y también nuestro editor

<https://github.com/antlr/antlr4/blob/master/doc/getting-started.md>

https://www.jetbrains.com/guide/python/