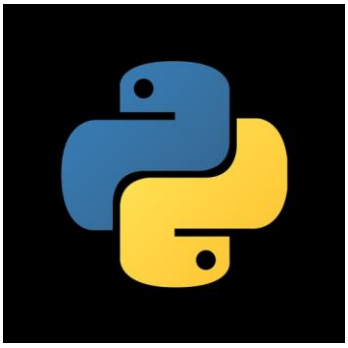
Laboratorio de Lenguajes Formales y de Programación, Sección B-.



PROYECTO 2 – LA LIGA BOT

MANUAL TECNICO

FECHA: 28/04/2022

Juan Josue Zuleta Beb
Carné: 202006353

Introducción

El presente documento describe los aspectos técnicos informáticos de la aplicación la liga bot, deseñada a través de la interfaz gráfica de Tkinter. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

Objetivos

Instruir el uso adecuado del de la instalación y comprensión del código y de la implementación de métodos, para el acceso oportuno y adecuado en la inicialización de este, mostrando los pasos a seguir en el proceso de inicialización, así como la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración.

Requisitos Mínimos del Sistema

Sistema operativo 64 bits

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- macOS 10.5 o superior
- Linux GNOME o KDE desktop
- Procesador a 1.6 GHz o superior
- 1 GB (32 bits) o 2 GB (64 bits) de RAM (agregue 512 MB al host si se ejecuta en una máquina virtual)
- 3 GB de espacio disponible en el disco duro
- Disco duro de 5400 RPM
- Tarjeta de vídeo compatible con DirectX 9 con resolución de pantalla de 1024 x 768 o más.
- Navegador web (Recomendado: Google Chrome)

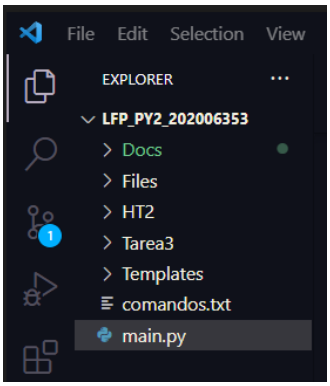
Software (Indispensable tenerlo instalado)

- IDE: Visual Studio Code
  Puede conseguirse en:
  https://code.visualstudio.com/

- Python 3.9 o posterior
  Puede conseguirse en:
  https://www.python.org/downloads/release/python-390/

Estructura raíz:

El proyecto tiene la siguiente estructura de directorios:

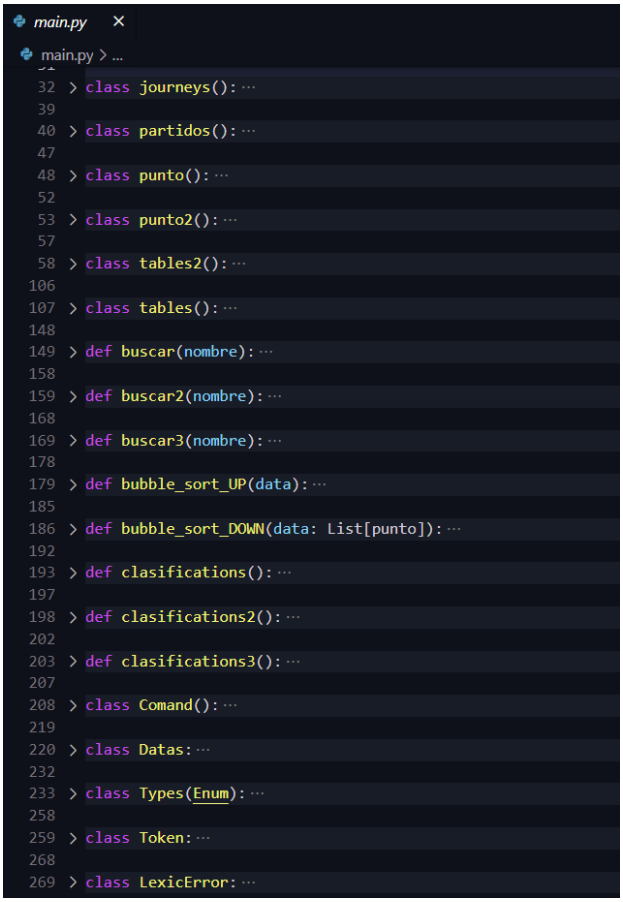


Directorio de la App: Raiz

La clase main:

Contiene el método principal de nuestra aplicación además de todas las clases y controlara todos los métodos y la implementación de la interfaz gráfica.

Las clases y métodos utilizados son los siguientes:



DESCRIPSION DE LOS METODOS Y CLASES:

Los métodos y clases utilizados para este programa fueron pensados y diseñados específicamente para el desarrollo e implementación de este, siguiendo rigurosamente los requerimientos de diseño inicialmente propuestos en el manual de requerimientos.

Clase Token:

Tiene como función recibir un constructor con todos los datos de tipo token que sean reconocidos para que puedan ser almacenados en la memoria de la aplicación y posteriormente utilizados.

```python
class Token:
    def __init__(self, token: Types, lexeme: str, row: int, col: int) -> None:
        self.token: Types = token
        self.lexeme: str = lexeme
        self.row: int = row
        self.col: int = col
```

Clase LexicError:

Tiene como función recibir un constructor con todos los datos de tipo error léxico que sean reconocidos para que puedan ser almacenados en la memoria de la aplicación y posteriormente utilizados.

```
class LexicError:
    def __init__(self, line: int, col: int, char: str) -> None:
        self.line: int = line
        self.col: int = col
        self.char: str = char
```

Clase SintaxError:

Tiene como función recibir un constructor con todos los datos de tipo error sintactico que sean reconocidos para que puedan ser almacenados en la memoria de la aplicación y posteriormente utilizados.

```
class SintaxError:
    def __init__(self, line: int, col: int, last_token: Types, expected_tokens: List[Types]) -> None:
        self.line: int = line
        self.col: int = col
        self.last_token: Types = last_token
        self.expected_tokens: List[Types] = expected_tokens
```

Función AFD (Automata Finito Determinista):

Tiene como función analizar el listado de caracteres escritos en el área de texto de la interfaz grafica de la aplicación. Esto lo realiza mediante la comparación de caracteres, reconociendo caracteres para los cuales fue preparado y para aquellos a los que detectara como errores, este es el principal encargado del funcionamiento de la aplicación, el diseño del autómata finito determinista se basó en el estudio del lenguaje presentado para la resolución del problema planteado con la aplicación de la liga bot, el autmata esta diseñado para encontrar la mayor cantidad posible de salidas exitosas para una cadena esperando siempre ser reconocida por el lenguaje.

El proceso de diseño fue el siguiente:

Definir los Elementos:

D = Digitos = [0 – 9]

W = Literal = [A - Z, a - z, Ñ, ñ]

F = [“_” , “ . ”]

Tokens:	Patrones:	Expression Regular:	Ejemplos:
tk_res	Inicia con una letra o un simbolo ala que le siguen cero o muchas veces cualquier combinación de letras.	**“-”?W+**	RESULTADO VS -f
tk_num	Inicia con un Digito seguido de cero o un digito.	**DD?**	1 11 50
Tk_year	Inicia con un < seguido de 4 digitos que representan un año seguido de un guion medio seguido de 4 digitos que representan un año cerrado por un >	**<D+”-”D+>**	<2020-2021> <2019-2020> <2021-2022>
string	Inicia con comillas seguido de una cadena de texto que puede incluir espacios, terminado por comillas	**“(^”)*”**	Real Madrid Villarreal Barcelona
tk_ID	Inicia con una letra seguida de letras, números o guiones bajos puede terminar la expresión con un numero.	**W(W|D|F)***	Temporada.html reporteGlobal1 reporte_general1

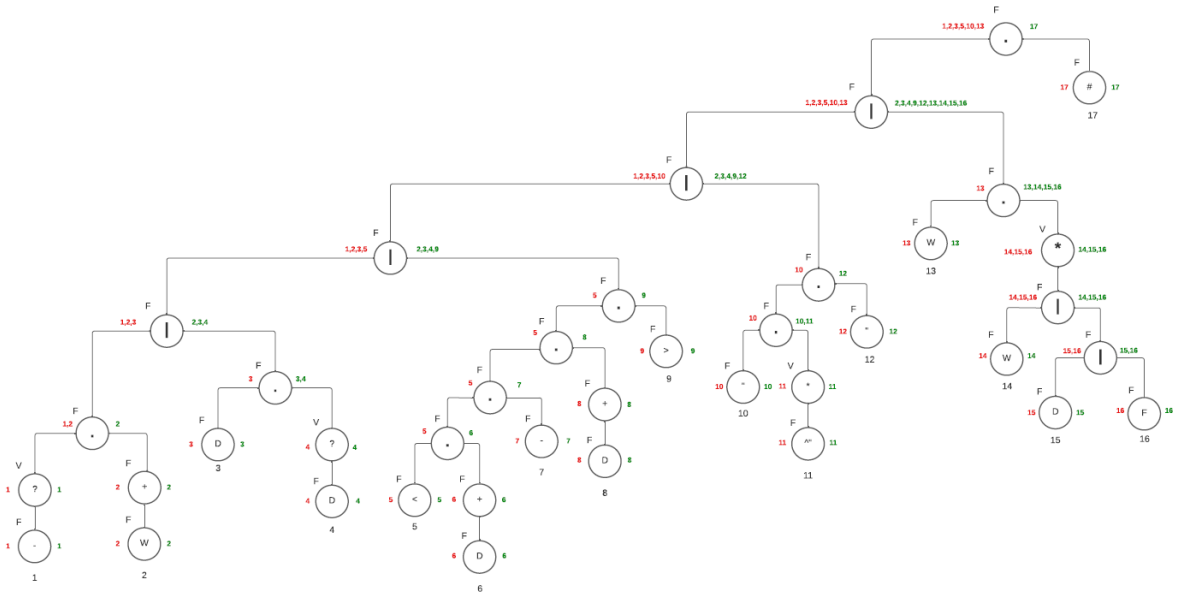
Expresión regular:

“-”?W+ | DD? | <D+”-”D+> | “(^”)*” | W(W|D|F)*

1. **Concatenar símbolo de aceptación al final de la ER.**

(“-”?W+ | DD? | <D+”-”D+> | “(^”)*” | W(W|D|F)*)#

2. **Construir el árbol binario de sintaxis.**
3. **Identificar cada hoja con terminales.**
4. **Calcular por cada nodo del árbol: Anulable, First, Last.**



5. Calcular Siguientes:

Valor	Hoja	Siguiente
-	1	2
W	2	2,17
D	3	4,17
D	4	17
<	5	6
D	6	6,7
-	7	8
D	8	8,9
>	9	17
“	10	11,12
^”	11	11,12
“	12	17
W	13	14,15,16,17
W	14	14,15,16,17
D	15	14,15,16,17
F	16	14,15,16,17
#	17	---

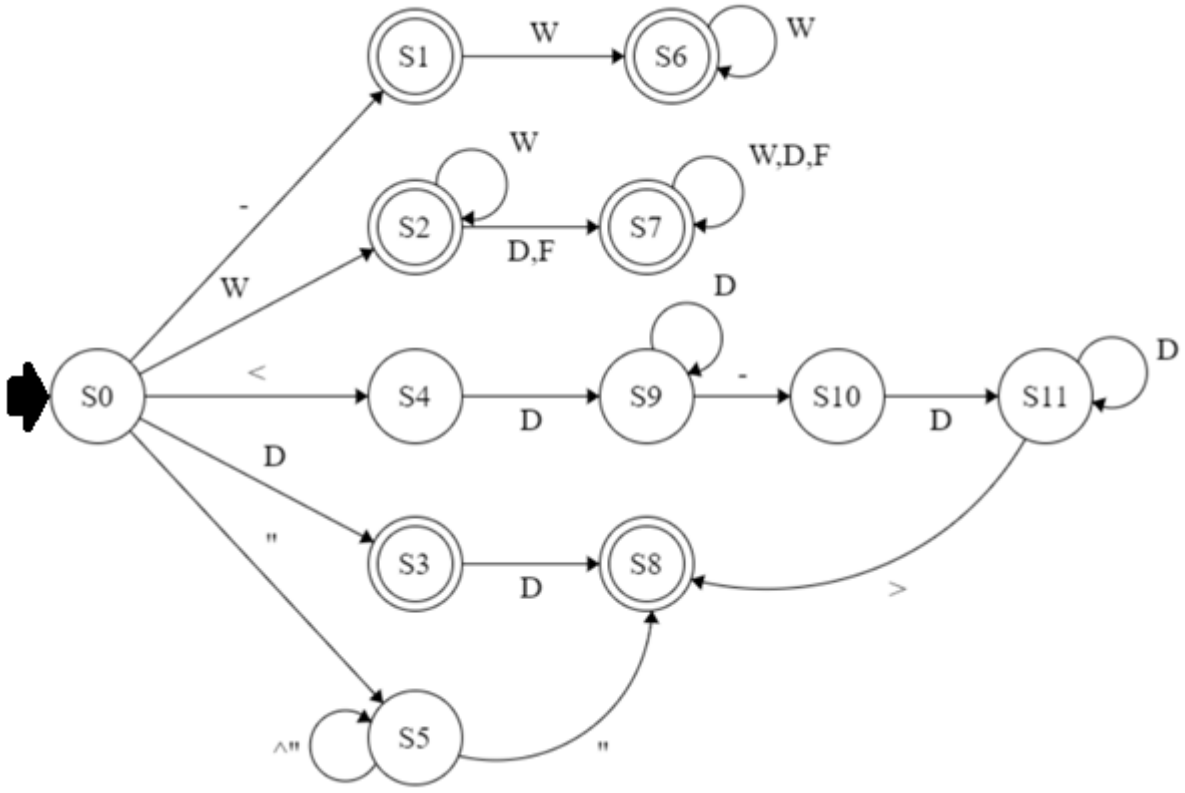
6. Tabla de transiciones:

	Estado	Valores	Siguientes
Inicio	S0	-,W,D,<,”,W 1,2,3,5,10,13	- : {2} = S1 W : {2,14,15,16,17} = S2 D: {4,17} = S3 <: {6} = S4 “: {11,12} = S5
Aceptación	S1	W,# 2,17	W: {2,17} = S6
Aceptación	S2	W,W,D,F,# 2,14,15,16,17	W: {2,14,15,16,17} = S2 D: {14,15,16,17} = S7 F: {14,15,16,17} = S7
Aceptación	S3	D,# 4,17	D: {17} = S8
	S4	D 6	D: {6,7} = S9
	S5	^”,” 11,12	^” : {11,12} = S5 “ : {17} = S8
Aceptación	S6	W,# 2,17	W: {2,17} = S6
Aceptación	S7	W,D,F,# 14,15,16,17	W: {14,15,16,17} = S7 D: {14,15,16,17} = S7 F: {14,15,16,17} = S7
Aceptación	S8	# 17	
	S9	D,- 6,7	D : {6,7} = S9 - : {8} = S10
	S10	D 8	D: {8,9} = S11
	S11	D,> 8,9	D : {8,9} = S11 > : {17} = S8

7. Tabla de estados

	Estados	Sigma							
		W	**D**	**F**	**^”**	**<**	**>**	**“**	**-**
0	S0	S2	S3			S4		S5	S1
#	S1	S6							
#	S2	S2	S7	S7					
#	S3		S8						
	S4		S9						
	S5				S5			S8	
#	S6	S6							
#	S7	S7	S7	S7					
#	S8								
	S9		S9						S10
	S10		S11						
	S11		S11				S8		

7.1 Autómata Finito Determinista



Clase Syntax:

Tiene como función analizar el listado de tokens ingresados y el orden en el cual estos han ingresado. Esto lo realiza mediante la comparación de caracteres, reconociendo caracteres para los cuales fue preparado y para aquellos a los que detectara como errores, este es el principal encargado del funcionamiento correcto de los comandos ingresados, implementa una gramática de tipo 2 por medio de la cual se estableció el orden de los terminales y no terminales para alcanzar el orden requerido para el programa. El diseño del analizador sintáctico se basó en el estudio del lenguaje presentado para la resolución del problema planteado con la aplicación, el analizador está diseñado para encontrar la mayor cantidad posible de salidas exitosas para una cadena esperando siempre ser reconocida por el lenguaje.

La gramática utilizada fue la siguiente:

Gramática tipo 2 (Análisis Sintáctico)**:**

<Inicio> : : = <RESULTADO>

| <JORNADA>

| <GOLES>

| <TABLA>

| <PARTIDOS>

| <TOP>

| <ADIOS>

<Repetir> : : = <RESULTADO>

| <JORNADA>

| <GOLES>

| <TABLA>

| <PARTIDOS>

| <TOP>

| <ADIOS>

<RESULTADO> : : = res_result string res_vs string res_temp tk_year

<JORNADA> : : = res_jorn tk_num res_temp tk_year <JORNADA'>

<JORNADA'> : : = tk_FLAGF tk_ID

| épsilon

<GOLES> : : = res_gol <COND>

<COND> : : = res_local string res_remp tk_year

| res_visitante string res_remp tk_year

| res_total string res_remp tk_year

<TABLA> : : = res_tabla res_temp tk_year <TABLA'>

<TABLA'> : : = tk_FLAGF tk_ID

| épsilon

<PARTIDOS> : : = res_partidos string res_temp tk_year <PARTIDOS'>

<PARTIDOS'> : : = tk_FLAGF tk_ID

| tk_FLAGJI tk_num

| tk_FLAGJF tk_num

| épsilon

<TOP> : : = res_top <COND2>

<COND2> : : = res_superior res_temp tk_year <TOP'>

| res_inferior res_temp tk_year <TOP'>

<TOP'> = tk_FLAGN tknum

| epsilon

<ADIOS> : : = res_adios

Función scanner:

Tiene como función analizar el listado de tokens ya clasificados por el autómata y comparando los tokens puede definir cuales tokens tiene un lexema igual al lexema esperado para algún comando, por lo tanto, la función del scanner es encontrar las palabras que nos ayudaran a construir las funciones de los comandos, esto se llevara a cabo mediante comparaciones y mediante ordenamiento de palabras.

Función AI_BOT:

Tiene como función recibir respuestas de la función scanner para manipular los datos obtenidos por medio de los comandos ingresados por el usuario y así comparar los datos con la base de datos local para poder enviar respuestas al usuario sobre las peticiones realizadas. Esta función es la clave del funcionamiento y la interacción con el usuario ya que permite enviar las respuestas esperadas y realizar los análisis pertinentes al manejo de la memoria dinámica y el manejo de los objetos dentro de la aplicación.

Función process_tokens:

Tiene como función analizar el texto que se encuentre en el área de texto de la interfaz grafica y está por medio del automata retornara un listado de tokens por medio del cual se desplegara un formulario con el listado de tokens actuales.

```
def process_tokens(tokens):

    def extract_names(tokens: List[Types]):
        return list(map(lambda t: t.name, tokens))

    env = Environment(loader=FileSystemLoader('Templates/'),
                      autoescape=select_autoescape(['html']))
    template = env.get_template('report_tokens.html')

    html_file = open('oficial_report_tokens.html', 'w+', encoding='utf-8')
    html_file.write(template.render(tokens=tokens, extract_names = extract_names))
    html_file.close()
    startfile('oficial_report_tokens.html')
```

Función process_errors:

Tiene como función analizar el texto que se encuentre en el área de texto de la interfaz gráfica y está por medio del autómata y del analizador sintáctico retornara un listado de errores por medio del cual se desplegara un formulario con el listado de errores actuales esto incluye los errores a nivel léxico, así como también los errores a nivel sintáctico.

```
def process_errors(errs,syntaxs):
    def extract_names(tokens: List[Types]):
        return list(map(lambda t: t.name, tokens))

    env = Environment(loader=FileSystemLoader('Templates/'),
                      autoescape=select_autoescape(['html']))
    template = env.get_template('report_errors.html')

    html_file = open('oficial_report_errors.html', 'w+', encoding='utf-8')
    html_file.write(template.render(errs=errs, syntaxs = syntaxs, extract_names = extract_names))
    html_file.close()
    startfile('oficial_report_errors.html')
```

Clase display_gui:

Tiene como función desplegar la interfaz gráfica al usuario mediante la librería Tkinter.

```
class display_gui():
    def __init__(self) -> None:
        self.root = tk.Tk()
        self.frame = Frame()

        self.lexeme = ''
        self.lexeme2 = ''
        self.lexeme3 = ''

        self.root.title('La Liga Bot')
        self.root.geometry('850x400')

        self.frame.config(width=850, height=400)
        self.frame.place(x=0, y=0)

        self.btn_RE = Button(self.frame, width = 17, text="Reporte de Errores", command = self.errors)
        self.btn_RE.place(x=713, y=30)

        self.btn_LLE = Button(self.frame, width = 17, text="Limpiar log de Errores", command=self.LOE_clear)
        self.btn_LLE.place(x=713, y=60)

        self.btn_RT = Button(self.frame, width = 17, text="Reporte de Tokens", command = self.tokens)
        self.btn_RT.place(x=713, y=90)

        self.btn_LLT = Button(self.frame, width = 17, text="Limpiar log de Tokens", command= self.LOT_clear)
        self.btn_LLT.place(x=713, y=120)

        self.btn_MU = Button(self.frame, width = 17, text="Manual de Usuario")
        self.btn_MU.place(x=713, y=150)

        self.btn_MT = Button(self.frame, width = 17, text="Manual Tecnico")
        self.btn_MT.place(x=713, y=180)

        self.btn_Send = Button(self.frame, width = 17, text="Enviar", command = lambda: self.analizer(self.text_area.get()))
        self.btn_Send.place(x=713, y=348)

        self.chat_area = Text(self.frame,  height = 19, width = 84)
        self.chat_area.place(x=30, y=30)

        Bot = '{} \n'.format('Bienvenido a la Liga Bot, Ingrese un Comando')
        self.chat_area.insert(END,Bot,("BOT",))
        self.chat_area.tag_configure("BOT",justify="left")
        self.chat_area.configure(state="disable")

        self.text_area = Entry(self.frame)
        self.text_area.place(x=30, y=350, width=675 , height=25)

        self.root.resizable(0,0)
        self.root.mainloop()
```