Universidad San Carlos de Guatemala

MANUAL TECNICO

Nombre: Nataly Saraí Guzmán Duarte
Carné: 202001570

Manual

Técnico

Proyecto 2

Contenido

Introducción

El presente documento describe los aspectos técnicos informáticos del programa creado en lenguaje python, el programa este compuesto por distintos componentes gráficos que lo vuelven intuitivo para el usuario, el documento busca familiarizar todos los aspectos técnicos del programa, como lo es su funcionamiento correcto de la aplicación, como la aplicación de los distintos flujos que el sistema crea.

Lugar de realización: Guatemala

Fecha:17/03/2022

Responsable de elaboración: Nataly Saraí Guzmán Duarte

Información destacada

El programa “La Liga Bot” fue realizado utilizando como plataforma visual studio code , la realización del proyecto fue de aproximadamente 14 días, cuya realización fue separada en botones del menú, cada botón se realizó en aproximadamente 3 días su realización completa.

Objetivos y alcances del sistema

- Dar a conocer al programada la forma en que se realizó el programa desde su inicio cuando se instaló python, mostrando específicamente el código y explicación de porque se realizó cada bloque de código.
- Realizar un programa para que como estudiante practique con el lenguaje python.
- Construir aplicaciones que se ejecuten con interfaz gráfica creada a código.
- Crear un analizador léxico y un analizador sintáctico para los archivos o contenido que se ingrese.
- Realizar un programa donde se pueda reconocer los mensajes que ingrese el usuario dando una respuesta y generando html.

Requerimientos

- RAM: 1 GB (mínimo).
- ROM: 250 MB (mínimo).
- Arquitectura x32 bits o x64 bits.
- Sistema operativo: Windows, Linux, MacOS.
- Lenguaje de programación: Python.

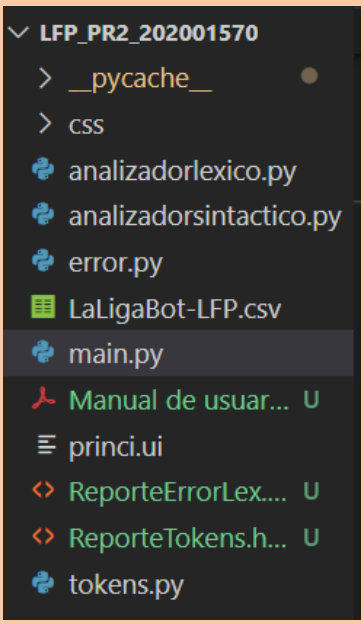
Datos técnicos para la realización del sistema

- Windows 10 Home Single Language.
- Procesador: Intel Core i5.
- RAM: 8.00 GB
- Plataforma IDE: Visual Studio Code
- Python 3.

Lógica del programa

El programa está constituido por una interfaz gráfica creada en el lenguaje de programación Python utilizando Qt Designer es simple e intuitivo con el usuario mostraremos a continuación las distintas clases que se usaron para su creación.

- Listado de clases creadas:



Cada clase cuenta con métodos que nos ayudan a que el sistema funciona de manera eficaz logrando así el uso correcto de la aplicación poniendo en práctica los distintitos condicionales y/o bucles y recursividad.

- Descripción de realización de cada clase:
  - Main: Contiene el main para ejecutar el programa.
  - Analizador Léxico: Encargada de analizar los datos ingresados señalando los tokens y errores.
  - Analizador Sintáctico: Encarga de analizar el mensaje si este escrito correctamente.
  - Tokens: Encargada de crear un objeto de tipo token.
  - Error: Encargada de crear un objeto de tipo error.
- Codificación y explicación de cada clase:
  - Clase main:
    - ✓ Constructor: Llamado de la ventana que contiene la interfaz gráfica asignación de funciones a los botones.
    - ✓ Lectura: Función que carga el contenido del archivo al área de texto.
    - ✓ Send: Función que llama a la clase del analizador.
    - ✓ Anali: Función que sale del programa
    - ✓ Borrarero: Función que reinicia la lista de errores
    - ✓ Borrartoke: Función que reinicia la lista de tokens

- ✓ HTMLTOKENS: Creación del html de tokens
- ✓ HTMLERRORES: Creación del html de errores

```
class Ventana(QMainWindow):
    def __init__(self):
        self.resultado= []
        self.partidos_arre=[]
        self.listaTokens = []
        self.listaErrores = []
        self.i=0
        super().__init__()
        loadUi("princi.ui", self)
        global anali, analisin
        anali= analizadorlexico()
        analisin = analizadorsintactico()
        archivo=self.lectura("LaLigaBot-LFP.csv")
        partidos= archivo.split("\n")
        for partid in partidos:
            datos=partid.split(",")
            p={
                "Fecha":datos[0],
                "Temporada":datos[1],
                "Jornada":datos[2],
```

```
def abrirmanu(self):
    webbrowser.open_new_tab('Manual de usuario.pdf')

def abrirmanu1(self):
    webbrowser.open_new_tab('Manual tecnico.pdf')

def lectura(self,ruta):
    archi=open(ruta, 'r')
    conte=archi.read()
    #archi.close()
    return conte

def send(self):
    archivo = self.lineEdit.text()
    anali.analizar(archivo,self.listaTokens,self.listaErrores)
    analisin.analizar(self.listaTokens, self.listaErrores,self.partidos
    if len(self.listaErrores)==0:
        self.textEdit.append('\n'+'YOU: '+archivo)
        if len(self.resultado)>0:
            self.textEdit.append('\n'+'BOT: '+str(self.resultado[0]))
```

- Clase analizador:
  - ✓ Analizar: Función que analiza cada elemento que se encuentra en el archivo, obtiene los tokens, los errores, y llena la lista de html para luego su creación.

```
from tokens import tokens
from error import error
import re
class analizadorlexico:
    def __init__(self):
        pass

    def analizar(self, codigo, listaTokens, listaErrores):
        global cod
        cod= codigo
        #self.listaHtml = []
        # VARIABLES
        fila = 1
        columna = 1
        buffer = ''
        centinela = '$'
        estado = 0
        codigo += centinela
        # AUTOMATA
        i = 0
```

```
            buffer=""
            estado=0
            i+=1
        if cade == ">":
            columna+=1
            buffer+=cade
            listaTokens.append(tokens('Signo_mayor',buffer, fila, co
            buffer=""
            estado=0
            i+=1
        elif cade == "<":
            columna+=1
            buffer+=cade
            listaTokens.append(tokens('Signo_menor',buffer, fila, co
            buffer=""
            estado=0
            i+=1
        elif cade == "]":
            columna+=1
            buffer+= cade
```

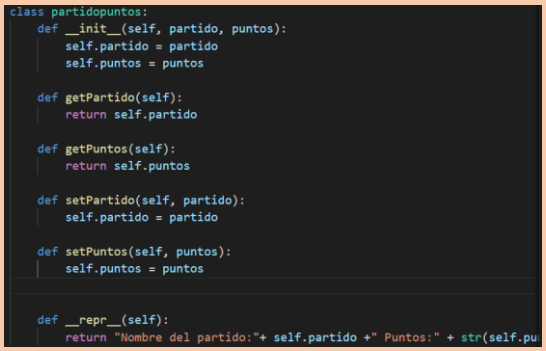
- Clase error:
  - ✓ _init_: método encargado de inicializar de variables y atributos de la clase.
  - ✓ strError: método encargado de generar una cadena para la impresión de información de la instancia de la clase.

```
class error:
    def __init__(self,tipo, descripcion, fila, columna):
        self.tipo = tipo
        self.descripcion = descripcion
        self.fila = fila
        self.columna = columna-len(descripcion)

    def strError(self):
        print("\n================================")
        print("Tipo:" +self.tipo)
        print("Descripción:  "+self.descripcion)
        print("Fila:  " +str(self.fila))
        print("Columna: "+str(self.columna))
```

- Clase tokens:
  - ✓ _init_: método encargado de inicializar de variables y atributos de la clase.
  - ✓ strToken: método encargado de generar una cadena para la impresión de información de la instancia de la clase.

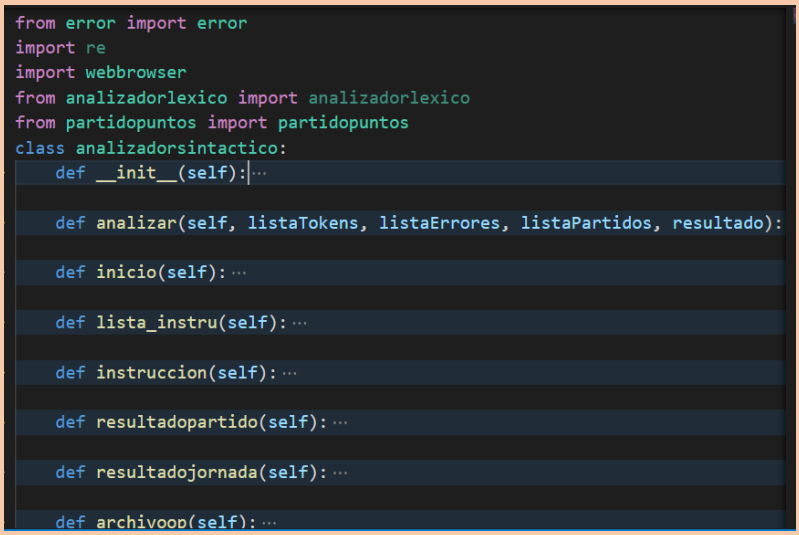
```
class tokens:
    def __init__(self, tipo,lexema, fila, columna):
        self.tipo = tipo
        self.lexema = lexema
        self.fila = fila
        self.columna = columna-len(lexema)

    def strToken(self):
        print("\n================================")
        print("Tipo:" +self.tipo)
        print("Lexema:  "+self.lexema)
        print("Fila:  " +str(self.fila))
        print("Columna: "+str(self.columna))
```

- Clase Partido puntos:
  - ✓ _init_: método encargado de inicializar de variables y atributos de la clase.
  - ✓ _repr_: método encargado de generar una cadena para la impresión de información de la instancia de la clase.



```
class partidopuntos:
    def __init__(self, partido, puntos):
        self.partido = partido
        self.puntos = puntos

    def getPartido(self):
        return self.partido

    def getPuntos(self):
        return self.puntos

    def setPartido(self, partido):
        self.partido = partido

    def setPuntos(self, puntos):
        self.puntos = puntos


    def __repr__(self):
        return "Nombre del partido:"+ self.partido +" Puntos:" + str(self.pu
```

- Clase Analizador Sintáctico:
  - ✓ _init_: método encargado de inicializar de variables y atributos de la clase.
  - ✓ Analizar: Función que analiza cada elemento que se encuentra en el archivo, obtiene los tokens, los errores, y busca a que función corresponde.
  - ✓ Inicio: Busca cada token
  - ✓ Lista instrucción: Busca si el primer token corresponde alguna gramática.
  - ✓ Instrucción: según el primer token manda a cada función.
  - ✓ Resultado partido: Función que corrobora la primera gramática y obtiene datos.
  - ✓ Resultado jornada: Función que corrobora la segunda gramática y obtiene datos.
  - ✓ Total de goles temporada: Función que corrobora la tercera gramática y obtiene datos.
  - ✓ Tabla general temporda: Función que corrobora la cuarta gramática y obtiene datos.
  - ✓ Temporada equipo: Función que corrobora la quinta gramática y obtiene datos.
  - ✓ Top equipos: Función que corrobora la sexta gramática y obtiene datos.
  - ✓ Adios: Función que corrobora la séptima gramática y obtiene datos.
  - ✓ HTMLS: Función que genera los html según corresponda.



```
from error import error
import re
import webbrowser
from analizadorlexico import analizadorlexico
from partidopuntos import partidopuntos
class analizadorsintactico:
    def __init__(self):···

    def analizar(self, listaTokens, listaErrores, listaPartidos, resultado):

    def inicio(self):···

    def lista_instru(self):···

    def instruccion(self):···

    def resultadopartido(self):···

    def resultadojornada(self):···

    def archivoop(self):···
```

Tabla de tokens y patrones

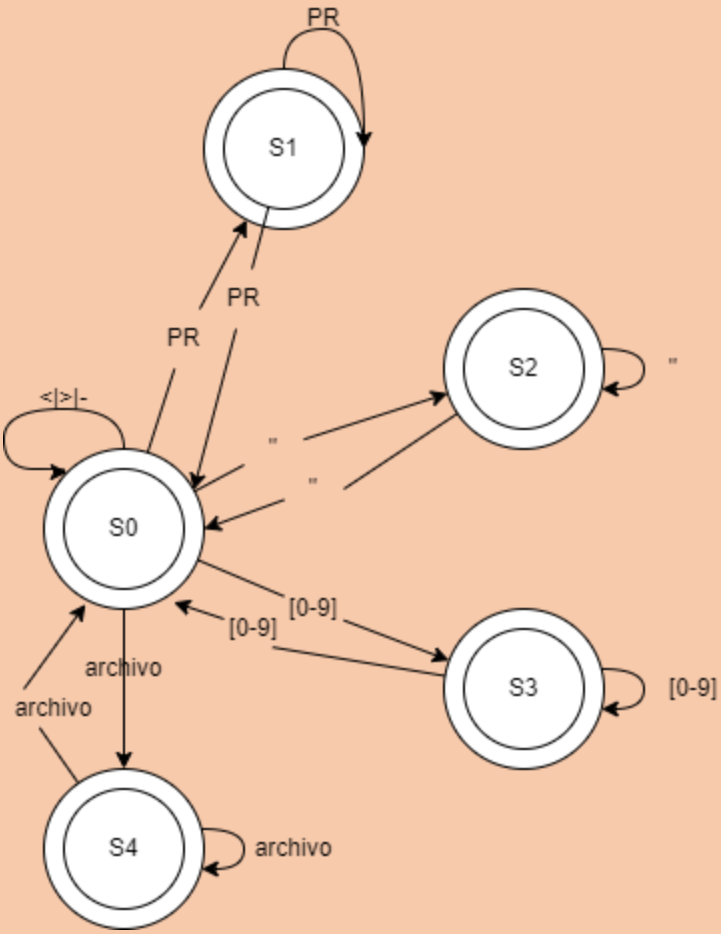
La tabla de tokens y patrones presentada a continuación, presenta los tokens que nuestro autómata y/o analizador léxico reconoce como parte de nuestro alfabeto:

----------Expresión regular----------
((RESULTADO|VS|TEMPORADA|JORNADA|-f|GOLES|LOCAL|VISITANTE|TOTAL| TABLA TEMPORADA|PARTIDOS|-ji|-jf|TOP|-n|SUPERIOR|INFERIOR|ADIOS|<|>|-|\s*"[\w\s,\.]+"|[0-9]| \s*<[0-9-]+>|[\w\s,\.]+| -[\w\s,\.]+)*)$

ID	Token	Patrón	Lexema
0	RESULTADO	[A-Z]	RESULTADO
1	VS	[A-Z]	VS
2	TEMPORADA	[A-Z]	TEMPORADA
3	JORNADA	[A-Z]	JORNADA
4	GOLES	[A-Z]	GOLES
5	LOCAL	[A-Z]	LOCAL
6	VISITANTE	[A-Z]	VISITANTE
7	TOTAL	[A-Z]	TOTAL
8	TABLA TEMPORADA	[A-Z]	TABLA TEMPORADA
9	PARTIDOS	[A-Z]	PARTIDOS
10	TOP	[A-Z]	TOP
11	SUPERIOR	[A-Z]	SUPERIOR
12	INFERIOR	[A-Z]	INFERIOR
13	ADIOS	[A-Z]	ADIOS
14	Corchete_Abrir	[	[
15	Corchete_Cerrar	]	]
16	Signo_menor	<	<
17	Signo_mayor	>	>
18	Guion	-	-
19	Numero	[0-9]	12
20	Fechas	[0-9]	<2019-2020>
21	Cadena	\s*"[\w\s,\.]+"	“Valencia”
22	Archivo	[A-Za-z0-9_]	reporteEspanol
23	-f	-[\w\s,\.]+	-f
24	-ji	-[\w\s,\.]+	-ji
25	-n	-[\w\s,\.]+	-n
26	-jf	-[\w\s,\.]+	-jf

Diagrama del autómata

En esta sección mostramos el diagrama del autómata aplicado en el analizador léxico construido para el proyecto:



Gramática libre del contexto

En esta sección mostramos la gramática libre del contexto aplicado en el analizador sintáctico construido para el proyecto:

NO TERMINALES= MAYUSCULAS
Tokens=minúsculas
INCIO→ LISTA_INSTR <EOF>
LISTA_INSTRU→ INSTRUCCIÓN LISTA_INSTR
|Epsilon
INSTRUCCION→RESULTADOPARTIDO
|RESULTADOJORNADA
|TOTALGOLESTEMPORADA
|TABLAGENERALTEMPORADA
|TEMPORADAEQUIPO
|TOPEQUIPOS
|SALIDABOT
RESULTADOPARTIDO→ resultado cadena vs cadena temporada signo_menor año guion año signo_mayor
RESULTADOJORNADA→ jornada numero temporada signo_menor año guion año signo_mayor ARCHIVOOP
ARCHIVOOP→ -f identificador
|Epsilon
TOTALGOLESTEMPORADA→ goles CONDICION cadena temporada signo_menor año guion año signo_mayor
CONDICION→ local
|visitante
|total
TABLAGENERALTEMPORADA→ tabla temporada signo_menor año guion año signo_mayor ARCHIVOOPP
TEMPORADAEQUIPO→ partidos cadena temporada signo_menor año guion año signo_mayor ARCHIVOOPP1 ARCHIVOP1
ARCHIVOP1→ -ji numero -jf numero
|Epsilon
TOPEQUIPOS→ top CONDICION1 temporada signo_menor año guion año signo_mayor ARCHIVOP2
CONDICION1→ superior
|inferior
ARCHIVOP2→ -n numero
|Epsilon
SALIDABOT→adios