

Universidad De San Carlos de
GuatemalaFacultad de Ingeniería
Escuela de Ciencias y Sistemas



Lenguajes Formales y de
ProgramaciónSección -B

MANUAL TÉCNICO

Kevin Gerardo Ruíz Yupe
201903791
28/10/2021

Objetivos

General:

Enfatizar en el uso de clases, variables y métodos escritos en la aplicación para usuarios con conocimiento específico en el tema. Asimismo, para edición de código en futuras actualizaciones.

Específicos:

- Mostrar el IDE utilizado para el desarrollo del programa
- Describir el proceso de solución, así como las herramientas utilizadas durante el proyecto.


Introducción

Este manual pretende explicar y describir los temas que se desarrollan durante el proceso de creación del programa. De la misma forma, se indica los requerimientos del sistema, el IDE utilizado y las librerías que se importaron.




La principal función de esta aplicación es el manejo de datos a partir de un archivo con extensión .lfp, recorriendo cada carácter utilizando ciclos y condicionales. Para verificar que el archivo no contenga errores léxicos y sintácticos se hace uso de expresiones regulares y gramáticas.









Descripción de la solución

Sea crea una carpeta para el proyecto, esta carpeta contendrá los archivos .py para las clases utilizadas. Existen subcarpetas que guardan imágenes como apoyo adicional al programa. Asimismo, se tienen otras subcarpetas para los archivos css y html.

 LFP_Proyecto2_201903791	2/10/2021 16:34	Carpeta de archivos
---	-----------------	---------------------

Carpetas para la documentación y el código fuente

 Code	27/10/2021 21:00	Carpeta de archivos	
 Documentacion	28/10/2021 12:56	Carpeta de archivos	
 README	28/09/2021 17:38	Archivo de origen ...	1 KB

 __pycache__	5/10/2021 10:23	Carpeta de archivos	
 Image	28/09/2021 17:51	Carpeta de archivos	
 Registros	27/10/2021 21:04	Carpeta de archivos	
 Tabla de Errores	27/10/2021 21:01	Carpeta de archivos	
 Tabla de Tokens	27/10/2021 20:05	Carpeta de archivos	
 Error	5/10/2021 10:18	Archivo de origen ...	1 KB
 Main	28/10/2021 13:05	Archivo de origen ...	80 KB
 Token	4/10/2021 20:18	Archivo de origen ...	1 KB

Existen 3 clases para el desarrollo del proyecto:



Como Clase Principal se tiene la Clase de nombre ***Application***, se muestra a continuación el contenido:

```
class Application():
    def __init__(self):
        self.root = Tk()
        self.root.call('encoding', 'system', 'utf-8')
        self.root.title('SolutionPy')
        self.root.iconbitmap('Image/Chart.ico')
        self.root.config(bg='#f0f3f4')
        self.root.state('zoomed')
        self.text=''
        self.ListaErrores1=[]
        self.ListaTokens1=[]
        self.ListaErrores=[]
        self.ListaTokens=[]
        self.ListaTokens_T=[]
        self.ListaErrores_T=[]
        self.PalabrasReservadas=['Claves','Registros','imprimir','imprimirln','conteo','promedio','conta
        self.PalabrasReservadasDatos=[]
```

Esta clase inicializa la ventana principal y configura el tamaño y su posición.

```
self.root = Tk()
self.root.call('encoding', 'system', 'utf-8')
self.root.title('SolutionPy')
self.root.iconbitmap('Image/Chart.ico')
self.root.config(bg='#f0f3f4')
self.root.state('zoomed')
self.text=''
self.ListaErrores1=[]
self.ListaTokens1=[]
self.ListaErrores=[]
self.ListaTokens=[]
self.ListaTokens_T=[]
self.ListaErrores_T=[]
self.PalabrasReservadas=['Claves','Registros','imprimir','imprimirln','conteo','promedio','conta
self.PalabrasReservadasDatos=[]
```

Método `create_widgets`

```
def create_widgets(self):
```

Este método inicializa cada componente de la aplicación y configura el tamaño de los widgets, del mismo modo su posición.

```
#Menu=ttk.Notebook(self.root)
s =ttk.Style()
labelframe = LabelFrame(self.root, text="Opciones",bg = "#212f3d",bd=1,labelanchor='n',fg='white
labelframe.pack(ipadx=50,ipady=50)
labelframe.config(font=('Segoe UI', 20,'bold'))
#labelframe.pack(expand=True,fill=X,pady=100)
labelframe.place(x=40,y=20,width=1260,height=100)
##
self.ButtonAbrirArchivo=Button(labelframe,text='Cargar Archivo',font=('Segoe UI', 12),bd=0,pady=
self.ButtonAbrirArchivo.place(x=400, y=10,width=150,height=30)

self.ButtonAnalizarArchivo=Button(labelframe,text ="Analizar Archivo",font=('Segoe UI', 12),bd=0
self.ButtonAnalizarArchivo.place(x=600, y=10,width=150,height=30)

self.combo=ttk.Combobox(labelframe,state="readonly")
self.combo.place(x=800, y=10,width=230,height=30)
self.combo.config(font=('Segoe UI', 12,))
self.combo['values']=['Generar Reporte de Errores','Generar Reporte de Tokens','Generar Árbol de

##
self.TextoEntrada = Text(self.root, height=25, width=89, wrap='none')
```

Método *select_file*

```
def select_file(self):
```

En este método se define el cuadro de diálogo para seleccionar el archivo .lfp. Se tiene un **Try** para verificar que el archivo exista o si no se seleccionó ningún archivo. Si no cumple, se desplegará un **messagebox** de error. Al contrario, se desplegará uno de proceso exitoso.

```
filetypes = (('Archivos lfp', '*.lfp'),('Todos los archivos', '*.*'))
try:
    a=open(filedialog.askopenfilename(title='Abrir un archivo',filetypes=filetypes,initialdir='.',
    self.text = a.read()
    a.close()
    messagebox.showinfo(title='Información', message='Archivo cargado exitosamente')
    self.TextoEntrada.insert(END,self.text)
except FileNotFoundError:
    messagebox.showerror(title='Error', message='No se eligió ningún archivo')
    return None

#archivo = filedialog.askopenfile(title='Abrir un archivo',initialdir='./',filetypes=filetypes)
```

Método *Lectura*

```
def Lectura(self):
```

Este método verifica si el archivo de entrada .lfp está vacío o si contiene texto. Si está vacío entonces se desplegará un **messagebox** de error. Si no, se llama al método **AnalizarLéxico**

```
TextoAnalisis=self.TextoEntrada.get('1.0','end-1c')
if TextoAnalisis.isspace() or self.TextoEntrada.get('1.0','end-1c')==':
    messagebox.showerror(title='Error', message='No se pudo analizar la entrada, intenta de nuev
else:
    messagebox.showinfo(title='Información', message='Lectura exitosa')
    TextoAnalizar=self.TextoEntrada.get('1.0','end-1c')
    #print(TextoAnalizar)
    self.AnalisisLexico(TextoAnalizar)
    TextoAnalizar=''
```

Método Auxiliares

```
def isLetra(self,caracter):
    if((ord(caracter) >= 65 and ord(caracter) <= 90) or (ord(caracter) >= 97 and ord(caracter) <= 122)):
        return True
    else:
        return False

def isNumero(self,caracter):
    if ((ord(caracter) >= 48 and ord(caracter) <= 57)):
        return True
    else:
        return False

def isSimbolo(self,caracter):
    if ((ord(caracter)==61) or (ord(caracter)==123) or (ord(caracter)==125) or (ord(caracter)==91) or (ord(caracter)==93)):
        return True
    else:
        return False
```

Estos métodos auxiliares verifican si la entrada es una Letra, Numero o Símbolo permitidos según la expresión regular. Retorna un valor verdadero si está dentro del rango según el código ascii.

Método AnalisisLéxico

En el método Analizar se crean diferentes condiciones para el archivo de entrada, de este modose lleva un control con los estados. Cada estado sigue el proceso del autómata finito determinista mostrado en el archivo pdf de esta carpeta.

```
def AnalisisLexico(self,Texto):
```

```

contador=0
fila = 1
columna = 0
estado=0
Tk_Identificador=''
Tk_Simbolo=''
Tk_ComillasDobles=''
TK_Numeral=''
Tk_CSimple=''
Tk_ComentarioMulti=''
Tk_ComentarioLinea=''
Tk_Numero=''
Tk_Cadena=''
Texto=Texto+'~'
for c in Texto:
    if estado==0:
        if self.isLetra(c):
            Tk_Identificador=Tk_Identificador+c
            estado=1
        elif self.isSimbolo(c):
            Tk_Simbolo=c
            estado=2
            contador+=1
            #columna += 1
            token=Token(contador,Tk_Simbolo,fila,columna,'Simbolo')
            self.ListaTokens1.append(token)
            Tk_Simbolo=''

```

Cada vez que se reconoce un token se crea un objeto de la clase Token:

```

class Token:
    def __init__(self,Numero,lexema,fila,columna,token):
        self.Numero=Numero
        self.lexema=lexema
        self.fila=fila
        self.columna=columna
        self.token=token

```

Se agrega a la lista de Tokens para posteriormente ser analizado.

Si se encuentra un error se crea un objeto de la Clase Error:

```

class Error:
    def __init__(self,tipo,fila,columna,descripcion,caracter):
        self.tipoError=tipo
        self.filaError=fila
        self.columnaError=columna
        self.descripcion=descripcion
        self.caracter=caracter

```

*Método **Estado_Inicial***

```
def estado_inicial(self):
    self.TextConsola.config(state='normal')

    #for l in self.ListaTokens:
    #    print(l.lexema,l.token)
    self.p_Claves1()

def p_Claves1(self):
    if self.ListaTokens[0].token=='Palabra Reservada' and self.ListaTokens[0].lexema=='':
        self.ListaTokens.pop(0)
        self.p_Claves2()
    else:
        error=Error('Sintactico',self.ListaTokens[0].fila,self.ListaTokens[0].columna,'')
        self.ListaErrores1.append(error)
        self.ListaTokens.pop(0)
        self.p_Claves2()

def p_Claves2(self):
    if self.ListaTokens[0].token=='Simbolo' and self.ListaTokens[0].lexema=='=':
        self.ListaTokens.pop(0)
        self.p_Claves3()
```

Este método llama a la función p_Claves1 cuya función es verificar si el primer elemento de la lista de tokens es igual al carácter o palabra esperada durante la ejecución

Métodos *Acciones*

```
def acciones(self):
    textoim=''
    textosalto=''
    if self.Comandos is None:
        self.TextConsola.insert(INSERT,'No hay comandos para ejecutar D:')
    else:
        for s in range(len(self.Comandos)):
            if self.Comandos[s][0]=='imprimir':
                self.TextConsola.insert(END,'> '+self.Comandos[s][1])
            elif self.Comandos[s][0]=='imprimirln':
                self.TextConsola.insert(END,'> '+self.Comandos[s][1]+'\\n')
            elif self.Comandos[s][0]=='exportarReporte':
                self.tabla_registros(self.Comandos[s][1])
                self.TextConsola.insert(END,'> Tabla generada')
            elif self.Comandos[s][0]=='conteo':
                self.TextConsola.insert(END,f'> {len(self.Registros)}')
            elif self.Comandos[s][0]=='datos':
                texto1=''
                for l in self.Claves:
                    self.TextConsola.insert(END,f'    {l}')

                self.TextConsola.insert(END,'\\n')
                for j in range(len(self.Registros)):
                    for k in range(len(self.Registros[j])):
                        self.TextConsola.insert(END,f'    {self.Registros[j][k]} |')
```

Este método recorre la lista de comandos e inserta en la Consola (Text Widget) según el nombre de la instrucción.

Tabla de Tokens

$L = [a-z, A-Z, \tilde{n}\tilde{N}]$

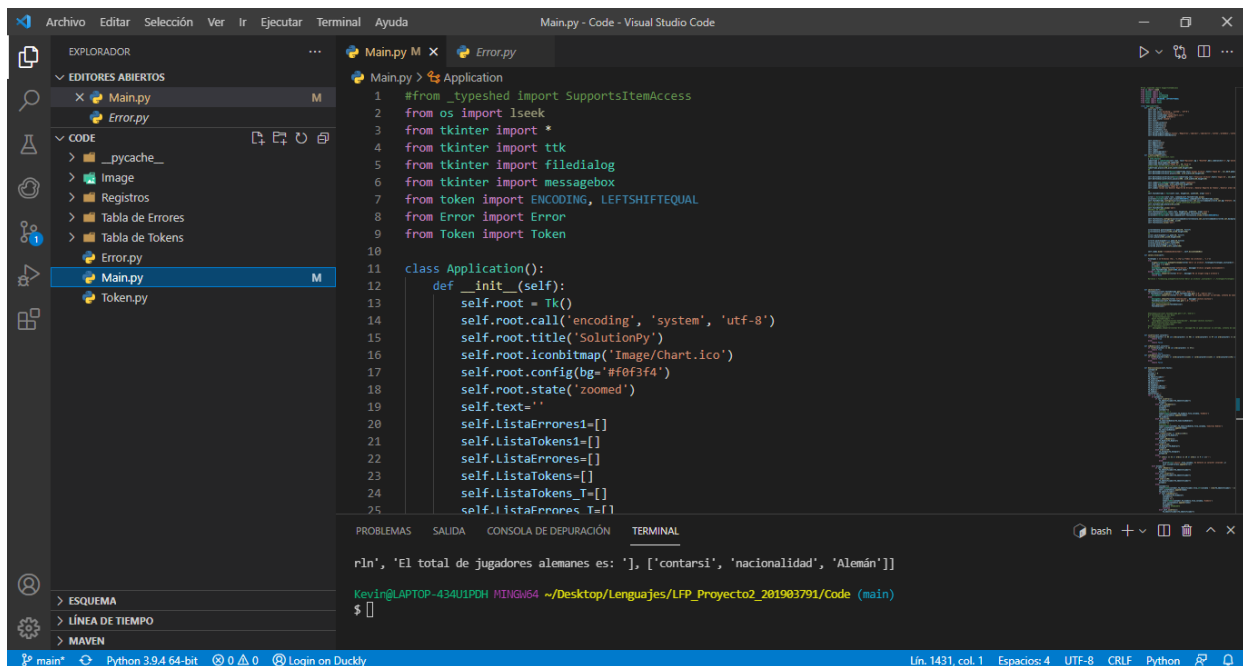
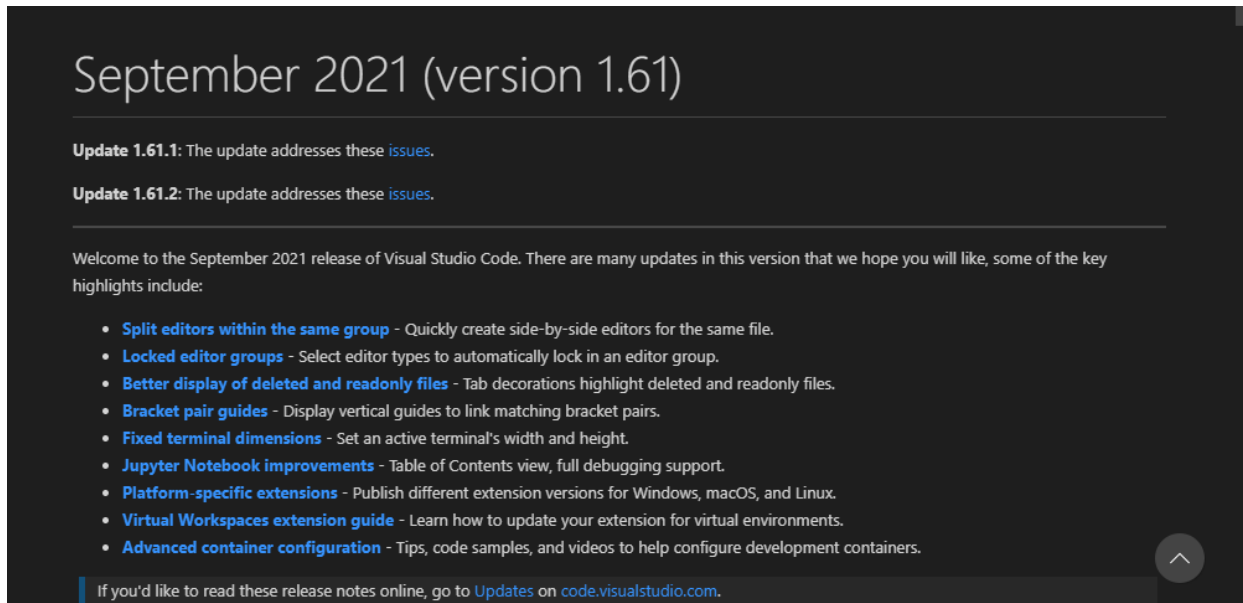
$S = ['=', ' \{ , ' \} ', '[, '] ', ' , ' , ' (, ') ', ' ; ']$

$D = [0,1,2,3,4,5,6,7,8,9]$

Identificador	$(L(L D '_')^*)\$$
Símbolo	$(SS^*)\$$
Numero	$(('+' '-')?(D+)('.' D+)?)\$$
Cadena	$('"' (^ ") '"')\$$
Comentario multilínea	$(' ' ' ' ' ' ' ' (^ ' ' ') * ' ' ' ' ' ' ' ' ' ')\$$
Comentario Linea	$('#' (^ \n)^*)\$$

Editor de código utilizado para la aplicación

Visual Studio Code de Microsoft versión 1.61.2



Requerimientos para Python

- Modern Operating System:
 - Windows 7 or 10
 - Mac OS X 10.11 or higher, 64-bit
 - Linux: RHEL 6/7, 64-bit (almost all libraries also work in Ubuntu)
- x86 64-bit CPU (Intel / AMD architecture)
- 4 GB RAM
- 5 GB free disk space

Librerías Utilizadas

```
from os import lseek
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
from tkinter import messagebox
from token import ENCODING, LEFTSHIFTEQUAL
from Error import Error
from Token import Token
```

- **Tkinter:** Se considera un estándar para la interfaz gráfica de usuario para Python yes el que viene por defecto con la instalación para Microsoft Windows.