

MANUAL TÉCNICO

Proyecto No.1

Por: Mario Ernesto Marroquín Pérez-202110509

INDICE

INTRODUCCION	2
LIBRERIAS	3
CREACION DE VENTANAS.....	3
Estilo de Ventanas:	3
Creación de ventanas:.....	4
Objetos Dentro de la Ventana:	4
CLASES DENTRO DEL PROGRAMA	5
Clase Operación:	5
Clase Token	6
FUNCIONES DENTRO DEL PROGRAMA.....	6
Función Imprimir Errores.....	6
Función Analizar	6
Función Guardar Tokens	8
Función AbrirArchi	8
Función GuardarCom	8
Función EscribirHTML.....	9
Función abrirManualUsuario	9
AUTÓMATA	10

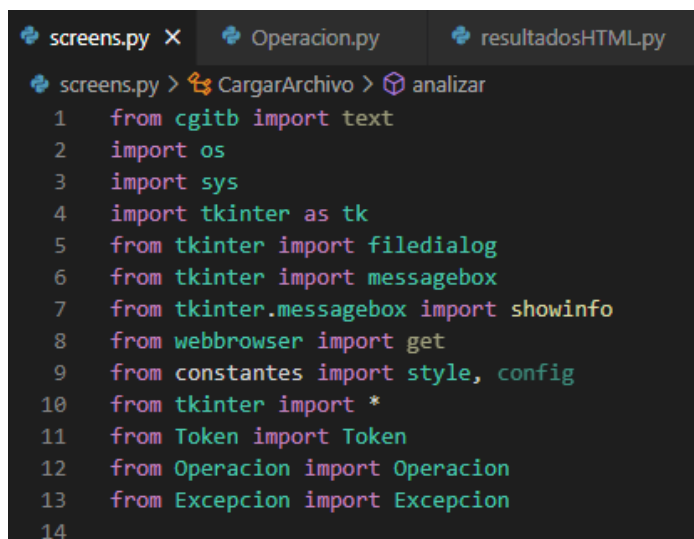
INTRODUCCION

El programa fue desarrollado en el lenguaje Python, haciendo uso de programación orientada a objetos (POO), el software es capaz de identificar un lenguaje dado, identificando los errores léxicos y ejecutando las instrucciones correspondientes, incluyendo una interfaz gráfica agradable e intuitiva para el usuario.

El programa se encuentra organizado y estructurado por clases, funciones de retorno y variables globales.

LIBRERIAS

Dentro del programa se importa una serie de librerías las cuales son utilizadas para la lectura de datos, la creación de la GUI (Interfaz Gráfica de Usuario por sus siglas en inglés). Estas librerías se importan utilizando la palabra “import”



```

screens.py X  Operacion.py  resultadosHTML.py
screens.py > CargarArchivo > analizar
1  from cgitb import text
2  import os
3  import sys
4  import tkinter as tk
5  from tkinter import filedialog
6  from tkinter import messagebox
7  from tkinter.messagebox import showinfo
8  from webbrowser import get
9  from constantes import style, config
10 from tkinter import *
11 from Token import Token
12 from Operacion import Operacion
13 from Excepcion import Excepcion
14

```

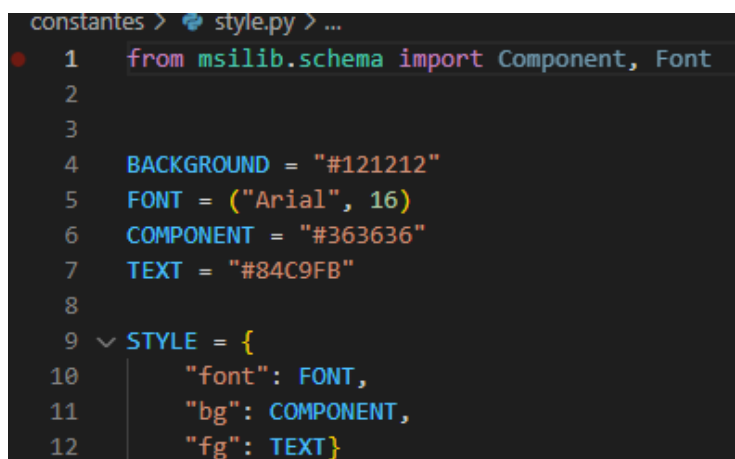
CREACION DE VENTANAS

Para la creación de ventanas se utilizó Programación Orientada a Objetos, utilizando la herencia y colocando un frame detrás del otro al momento de cambiar entre ventanas, mejorando así la experiencia del usuario al utilizar la ventana.

Utilizando las clases para la creación de una ventana principal y heredar hacia las demás ventanas “hijas”, manteniendo la misma estética, tamaño y estilo sin tener que modificar cada ventana.

Estilo de Ventanas:

Dentro del archivo “style.py” se asignaron los colores, la fuente, el tamaño de fuente, el color de fondo y el tipo de texto para las ventanas, este archivo se importa dónde se encuentran las ventanas para así aplicarles el estilo especificado.



```

constantes > style.py > ...
1  from msilib.schema import Component, Font
2
3
4  BACKGROUND = "#121212"
5  FONT = ("Arial", 16)
6  COMPONENT = "#363636"
7  TEXT = "#84C9FB"
8
9  STYLE = {
10     "font": FONT,
11     "bg": COMPONENT,
12     "fg": TEXT}

```

Creación de ventanas:

En el archivo “Manager” se configuran las ventanas de forma general y se importan para su creación, dentro de esta ventana se agrega el nombre y las configuraciones.

```

Manager.py > ...
1  import tkinter as tk
2  from typing import Container
3  from constantes import style
4  from screens import Menu, Abrir, GuardarComo, Analizar, Errores, Ayuda, CargarArchivo
5
6  class Manager(tk.Tk):
7      def __init__(self, *args, **kwargs):
8          super().__init__(*args, **kwargs)
9          self.title("PROYECTO 1_202110509")
10         container = tk.Frame(self)
11         #self.mode = "Archivo"
12         container.pack(side="top", fill="both", expand=True)
13         container.configure(background=style.BACKGROUND)
14         container.grid_rowconfigure(0, weight=1)
15         container.grid_columnconfigure(0, weight=1)
16
17         self.frames = {}
18         for F in (Menu, Abrir, GuardarComo, Analizar, Errores, Ayuda, CargarArchivo):
19             frame = F(container, self)
20             self.frames[F] = frame
21             frame.grid(row=0, column=0, sticky="nsew")
22         self.show_frame(Menu)
23
24     def show_frame(self, container):
25         frame = self.frames[container]
26         frame.tkraise()
27

```

Objetos Dentro de la Ventana:

Para la creación de objetos, por ejemplo, un botón, una caja de texto etc. Dentro de la clase creada para esa ventana, se crea una función donde se inician los objetos colocados en esa función.

```

def init_widgets(self):
    tk.Label(
        self,
        text = "=====MENÚ PRINCIPAL=====",
        justify= tk.CENTER,
        **style.STYLE
    ).pack(
        side = tk.TOP,
        fill = tk.BOTH,
        expand = True,
        padx = 22,
        pady = 11
    )
    optionFrame = tk.Frame(self)
    optionFrame.configure(background=style.COMPONENT)
    optionFrame.pack(
        side=tk.TOP,
        fill=tk.BOTH,
        expand=True,
        padx=22,
        pady=11
    )
    tk.Label(
        optionFrame,
        text = "=====Elija una opción=====",
        justify= tk.CENTER,
        **style.STYLE
    ).pack(
        side = tk.TOP,
        fill = tk.BOTH,
        expand = True,
        padx = 22,
        pady = 11
    )

```

Dentro de esta función, se establecen las características y configuración de los elementos, en un botón se establece el texto que llevará, así como los colores y estilo.

CLASES DENTRO DEL PROGRAMA

Cada clase dentro del programa representa una ventana, las configuraciones, el estilo y la posición del frame se heredan de la clase madre “Manager”, el contenido dentro de estas clases pueden ser configuraciones adicionales, propias de esa ventana, objetos dentro de la ventana o una función de cálculo u generación de HTML.

```
class Menu(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.configure(background=style.BACKGROUND)
        self.controller = controller
        #self.opcion = tk.StringVar(self, value="Archivo")

        self.init_widgets()
```

Las clases dentro del programa todas son utilizadas de la misma manera, para crear ventanas o guardar los datos de entrada (ingresados a través del archivo), las clases que guardan las operaciones se encuentran en un archivo de Python diferente al principal, siempre dentro de la misma carpeta.

```
Excepcion.py > ...
1 class Excepcion:
2
3     def __init__(self, error, tipo, columna, fila):
4         self.error = error
5         self.tipo = tipo
6         self.columna = columna
7         self.fila = fila
8
```

Clase Operación:

Dentro de esta clase se obtienen todas las operaciones de entrada (sumas, restas, división, etc.), las operaciones se almacenan dentro de un arreglo y posteriormente son analizadas a través de condicionales.

```
import re
import math

class Operacion:
    def __init__(self, tipo):
        self.tipo = tipo
        self.operandos = []
```

Clase Token

Dentro de la clase Token se almacena la columna, fila y el lexema de entrada.

```
Token.py > Token > __init__
1 class Token:
2     def __init__(self, fila, columna, lexema):
3         self.fila = fila
4         self.columna = columna
5         self.lexema = lexema
```

FUNCIONES DENTRO DEL PROGRAMA

Función Imprimir Errores

La función imprimir errores se encarga de elaborar una tabla en HTML y devolver dicha tabla con todos los errores léxicos encontrados.

```
def imprimirErrores(self):
    errores = ""
    # print('-'*43)
    # print ("| {:<7} | {:<7} | {:<4} | {:<4} |".format('Lexema','Tipo','Columna','fila'))
    # print('-'*43)
    for error in self.tabla_errores:
        #print ("| {:<7} | {:<7} | {:<4} | {:<4} |".format(error.error, error.tipo, error.columna, error.fila))
        errores += ('<td align = "center">'+error.error+'</td>\n<td align = "center">'+error.tipo+'</td>\n<td align = "center">'+error.columna+'</td>\n<td align = "center">'+error.fila+'</td>\n')
    html = ('<table border="1" style="width:100%" align = "center">\n'+
        '<tr>\n'+
        '<th>Lexema</th>\n'+
        '<th>Tipo</th>\n'+
        '<th>Columna</th>\n'+
        '<th>Fila</th>\n'+
        '</tr>\n'+
        errores)
    self.tabla_errores.clear()
    return html
```

Función Analizar

Esta función se encarga de analizar la entrada, en esta función se programó el Autómata Finito Determinista con el cual se recorre los caracteres ingresados y se reconoce los diferentes tipos de operaciones.

```
def analizar(self, cadena, operacion:Operacion):
    etiqueta_abre = ''
    etiqueta_cierra = ''
    token = ''
    operaciones = []
    cierre = False
    tipo = operacion
```

Dentro de esta función se utilizó un ciclo While para recorrer la cadena ingresada siempre y cuando exista un carácter como mínimo, la función también se compone de “if” condicionales para analizar la cadena ingresada, y dirigirse a los respectivos estados del autómata.

Ignoramos los espacio y saltos de línea con la siguiente condición:

```
if char == '\n':
    self.fila += 1
    self.columna = 0
    cadena = cadena[1:]
    continue
elif char == ' ':
    self.columna += 1
    cadena = cadena[1:]
    continue
```

Comprobamos la entrada de la cadena en un if para movernos a los diferentes estados:

```
if self.estado_actual == 0:
    if char == '<':
        self.guardar_token(char)
        self.estado_actual = 1
        self.estado_anterior = 0

elif self.estado_actual == 1:
    if char.lower() in self.letras:
        token += char
        self.estado_actual = 1
        self.estado_anterior = 1
    elif char == '>':
        self.guardar_token(token)
        self.guardar_token(char)
```

Cuando nos topamos con un estado de aceptación, procedemos a guardar los tokens recuperados y la entrada de la cadena:

```
if cierre:
    etiqueta_cierra = token
    cierre = False

    if etiqueta_cierra.lower() == 'operacion':
        operacion.operandos = operaciones
        return [cadena, operacion]

if etiqueta_abre.lower() == 'operacion':
    op = Operacion(token)
    valor = self.analizar(cadena[1:], op)
    cadena = valor[0]
    operaciones.append(valor[1])

etiqueta_abre = token
token = ''

self.estado_actual = 2
self.estado_anterior = 1
```


Según nuestro autómata se configuran los demás condicionales, para dirigirnos a nuestros diferentes estados, así como regresar a otros estados, para obtener un error léxico agregamos un else al final de cada condicional para guardarlo en la tabla de errores.

```
elif self.estado_actual == 2:
    if char in '<':
        self.guardar_token(char)

        self.estado_actual = 1
        self.estado_anterior = 2
    elif char in self.numeros:
        token += char
        self.estado_actual = 4
        self.estado_anterior = 2
```

Función Guardar Tokens

Dentro de esta función guardamos los tokens recuperados de nuestro autómata, así como la fila, columna y el lexema recuperado, estos tokens se guardan en un arreglo.

```
def guardar_token(self, lexema):
    nuevo_token = Token(self.fila, self.columna, lexema)
    #self.tabla_tokens.clear()
    self.tabla_tokens.append(nuevo_token)
```

Función AbrirArchi

Esta función abre el archivo ingresado y devuelve su contenido junto con la ruta donde se encuentra.

```
def AbrirArchi(self):
    with open(direccion, 'r') as f:
        global cont1
        cont1 = f.read()
    moverInfo(cont1, direccion)
    abrirArch()
```

Función GuardarCom

Esta función es la encargada de guardar el archivo de la forma “Guardar Como”, utiliza el filedialog de Python.

```
def guardarCom(self):
    files = [('Text Document', '*.txt'),
            ('All Files', '*.*'),
            ('Python Files', '*.py')]
    file = filedialog.asksaveasfile(filetypes = files, defaultextension = files)
    with open(direccion, 'r') as f:
        cont2 = f.read()
    file.write(cont2)
    messagebox.showwarning(message="Se ha guardado el Archivo", title="FELICIDADES")
```

Función EscribirHTML

Esta función se encarga de obtener las variables ya convertidas en cadena y agregarlas a un formato HTML ya preestablecido, la función toma un archivo HTML, lo abre, borra su contenido e inserta el nuevo contenido, cierra el archivo y procede a abrirlo con el lector de pdf preestablecido por el sistema. Tanto para escribir el HTML de resultados como el de errores tienen la misma lógica, el de error lo hace en forma de tabla.

```
def escribirHtml(cont):
    with open("RESULTADOS_202110509.html", "w") as archivo:
        antes = ("<!DOCTYPE html>\n"+
            "<html>\n"+
            "<head>\n"+
            "<title>"+ "RESULTADOS" + "</title>\n"+
            "</head>\n"+
            "<body>\n"
            "<h1>RESULTADOS</h1>\n")
        despues = ("</body>\n"+
            "</html>")

        archivo.write(antes+cont+despues)
    archivo.close()
    directorio = os.getcwd()
    os.startfile(directorio+"\RESULTADOS_202110509.html")
```

Función abrirManualUsuario

Esta función es la encargada de abrir el manual de usuario directamente utilizando un botón dentro de la aplicación, la función obtiene la ruta donde se encuentra el manual y lo abre en el lector de PDF preestablecido por el sistema. Tanto para abrir el manual de usuario como el manual técnico las funciones tienen la misma lógica, solo cambia la ruta del archivo.

```
def abrirManualUsuario(self):
    directorio = os.getcwd()
    os.startfile(directorio+"\Manuales\ManualUsuPrueba.pdf")

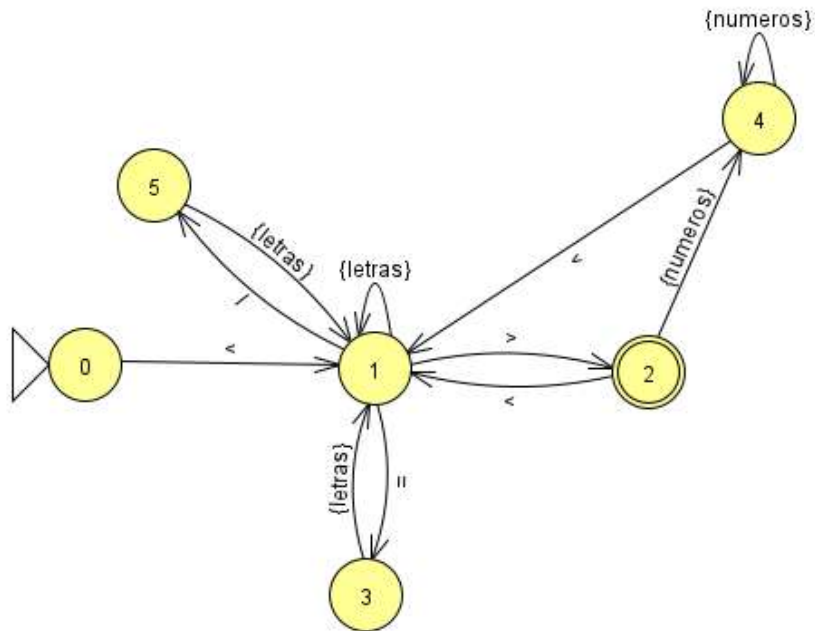
def abrirManualTec(self):
    directorio = os.getcwd()
    os.startfile(directorio+"\Manuales\ManualTecPrueba.pdf")
```

AUTÓMATA

El Autómata Finito Determinista (AFD por sus siglas), diseñado para esta aplicación es el siguiente:

{Letras} = [a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]

{numeros} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]



TOKEN	PATRÓN
<	<
{letras}	Letra seguida de letras
{numeros}	Números seguidos de numeros
>	>
=	=
/	/