PROYECTO 1

MANUAL TÉCNICO

2022

Lenguajes Formales y de Programación

Damaris Julizza Muralles Véliz

202100953

Detalles de desarrollo

Este software se desarrollo en el lenguaje python, utilizando como interfaz gráfica la librería de Tkinter.

Descripción general

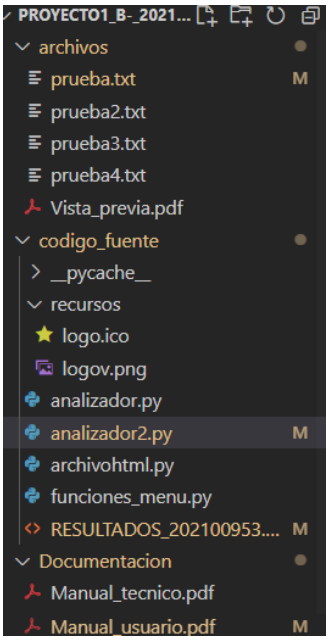
El programa es un software que permite analizar léxicamente un archivo de texto el cual contiene un lenguaje formal que da las instrucciones necesarias para realizar una calculadora.

Cuenta con distintas opciones para manejar el archivo, así como también opciones de ayuda. Todo el desarrollo de la interfaz como se a mencionado con anterioridad se realizó mediante Tkinter y se utilizaron conceptos de AFD y expresiones regulares dentro de la lógica del programa para su funcionalidad.

Lógica del programa

Recursos Utilizados:

En el folder “Proyecto1_B-_202100953” encontraremos todos los recursos y archivos utilizados, como lo son los iconos usados para la interfaz, documentos de apoyo que incluye el presente manual y manual de usuario, algunos archivos de prueba y los archivos de las clases utilizadas, estos ultimas son “interfaz.py”, “funciones_menu.py”, “analizador.py” y “archivohtml.py”.



Interfaz:

```
def ventanaanalizador():···

def abrir(texto):···

def guardar(texto):···

def guardar_como(texto):···

def ejecutar(texto):···

def salir(q):···

def manualu():···

def manualt():···

def info():···


if __name__ == "__main__":···
```

Este archivo contiene las funciones presentadas en la imagen anterior las cuales se encargan de controlar la interfaz gráfica de nuestro programa y lo enlaza con la lógica correspondiente para su funcionamiento, entre estas funciones tenemos:

➢ Método __Main__:

Este coloca prioridad a la ejecución de lo que contenga.

En este método se escribió código para la creación de una pequeña ventana de inicio para cargar el programa, agregando una barra dinámica de carga en la parte inferior que se inicializa al pulsar el botón comenzar ejecutando así el comando para el despliegue de la ventana principal.

➢ Función ventanaanalizador:

En esta función se estructura toda nuestra interfaz y se configura los atributos de cada componente según el aspecto buscado.

Para esto se usó la librería tkinter y se creó una ventana se le coloco un título, se agregó un icono y se configuro las dimensiones, posterior a esto de le enlazaron frames para crear las divisiones que se necesitaban en la interfaz, uno para el menú y otro para el editor y visualizador pdf, a estos frame se les enlazaron labels y bottons, también se agrego una caja de texto text para el editor y para el visualizador se enlazo a un frame y se llama a la una función de la librería de funciones_menu.

A los bottons se les agrego un command para que ejecuten las funciones correspondientes que se explican a continuación.

➢ Función abrir, guardar, guardar_como y ejecutar:

Estos métodos son llamados por los bottons y recibe como atributo la cadena de texto que esta contenida en la caja de texto, el editor, esto posteriormente se extrae por medio de un método get y esta información se pasa a las funciones de la librería funciones_menu que se creó para este proyecto.

La función abrir en específico extrae el contenido del archivo mediante una función de funciones_menu y luego agrega esa información a la caja de texto por medio de un insert.

Las funciones de guardar, si esta acción se realiza correctamente mostraran un messagebox.showinfo para informar de esto.

➢ Función manualu y manualt:

Esta función simplemente llama a la función de viwer en funciones_menu, pasando como parámetro un numero que representa a que manual se refiere, para manual de usuario el 1 y para el técnico el 2.

➢ Función info:

Se creo un messagebox.showinfo que despliega la informacion del desarollador.

También se hace una instancia de la función viwer en funciones_menu para poner un pdf creado como vista previa con el atributo 0.

➢ Función salir:

Esta función recibe como a tributo a la ventana y todo lo contenido en ella para posteriormente usar un destroy() para detener la ejecución de esta.

Funciones_menu:

```
class funciones:
    def __init__(self):···

    def pdfviwer(self,menuop):···

    def viewer(self):···

    def salve(self, contenido):···

    def salve_c(self,contenido):···

    def run(self):···
```

Aquí se encuentra la clase funciones que a su vez está compuesta por varias funciones las cuales como vimos antes son llamadas en la librería interfaz, a qui se hizo uso de la librerías

de Tkinter, filedialog de tkinter y tkPDFViewer también se importó el archivo del analizador creado y que se explicara después.

➢ Método __init__:

Como este método sirve para inicializar lo que utilizaremos, aquí se colocó solo la variable ruta que nos guardara la direccion en donde se encuentra el archivo a manejar para poder realizar las funciones de guardar, etc.

➢ Función pdfviwer:

Este recibe como atributo un número que representa al archivo que se debe abrir, 0 el predeterminado, 1 el manual de usuario y 2 este manual.

Se uso un frame para contener al visor y se hicieron configuraciones para el tamaño y la posición que ocupa en la ventana principal. Luego se llamo a tkPDFViewer.ShowPdf() para inicializar el visor y para asegurarnos de que no se sigan agregando mas pdf aparte del que deseamos después de esta línea se colocó .img_object_li.clear().

Luego se usaron condicionales if para abrir el archivo que se requiere y para abrirlo se usó la función pdf_view a la cual se le pasaron varios atributos, entre ellos la dirección del documento requerido.

➢ Función viewer:

Esta función será la encargada de abrir el archivo que se requiere, para ello se usó FileDialog.askopenfilename, esta función despliega un navegador de archivos y proporciona como valor la dirección de este, esta dirección se la asignamos a la variable ruta que se colocó en el método de inicio.

Al obtener la ruta se abre el archivo con la función open() pasando como uno de sus atributos r para indicar que se está leyendo, luego se lee con la función read() y el valor obtenido se pasa a una variable denominada contenido , luego cerramos el archivo con la función close(). Hecho esto retornamos la variable de contenido

➢ Función save:

Recibe como atributo a lo contenido en la caja de texto de la interfaz y mientras nuestra variable ruta no este vacía, es decir mientras ya se haya abierto un archivo sobre escribir el contenido de dicho archivo, en esto hacemos lo mismo que para abrir, open()para abrir el archivo solo que en esta ocasión usaremos el atributo w+ para indicar que se estar escribiendo, luego se usa la función write() que tiene como atributo el contenido que se obtuvo de la caja de texto y luego se cierra el archivo con close().

En caso de que la ruta este vacía se llama a la función de save_c.

- ## Función save_c:

  Esta función tiene la misma estructura que sabe, solo que antes del mismo código de sabe agregamos FileDialog.asksaveasfile que nos abrirá un navegador de archivos para que se escoja la dirección en donde se guardara y el nuevo nombre que se quiera para el archivo.

- ## Función run:

  Se instancia al archivo de analizador y a su función de compilador() .

analizador:

```
class token(Enum):···

class analizador:
    def __init__(self):···

    def compilador(self,dir):···

    def numero (self,cadena:str):···

    def operador(self, cadena:str):···

    def tipo(self,cadena:str):···

    def texto_f(self,cadena:str):···

    def funcion(self,cadena:str):···

    def estilo(self,cadena:str):···

    def lineas(self):···

    def quitar_L (self,cadena:str,column:int):···

    def quitar_E (self,cadena:str,etiqueta:str,column:int):···

    def etiqueta(self, cadena:str,etiquet:str):···
```

Este archivo contiene las funciones presentadas en la imagen anterior las cuales se encargan de toda la lógica correspondiente al analizador léxico.

Las librerías utilizadas son las siguientes:

```
from enum import Enum
import math
import os
import re
from archivohtml import constructorHTML
```

En donde el archivonntml es creado especificamente para el programa. A continuacion se detalla informacion de las clases y funciones usados en ellas.

➢ Clase token:

Esta clase contiene una lista de todos los tokens usados en el analizador, para crear esta lista se hizo uso de la librería Enum. Los tokens utilizados son los siguientes:

```
t_menor = "<"
t_mayor = ">"
t_slash = "/"
t_igual = "="
t_e_tipo ="Tipo"
t_e_operacion ="Operacion"
t_e_num = "Numero"
t_e_texto ="Texto"
t_e_funcion ="Funcion"
t_e_titulo ="Titulo"
t_e_Descrip ="Descripcion"
t_e_contenido ="Contenido"
t_e_estilo ="Estilo"
t_a_color ="Color"
t_a_tam ="Tamanio"
t_o_suma ="SUMA"
t_o_resta = "RESTA"
t_o_multiplicacion = "MULTIPLICACION"
t_o_division = "DIVISION"
t_o_potencia = "POTENCIA"
t_o_raiz = "RAIZ"
t_o_inverso = "INVERSO"
t_o_seno = "SENO"
t_o_coseno = "COSENO"
t_o_tangente = "TANGENTE"
t_o_mod = "MOD"
t_num = "([0-9]+)(.[0-9]+)?"
t_text = "[A-Za-zÀ-ÿ\u00f1\u00d10-9\%\*\+\/=√_,-.:;\(\)\[\]\^\s]*"
```

Como se puede ver para los tokens t_num y t_text se hizo uso de una expresión regular como patrón, para los números se consideró que se acepten tanto número enteros como decimales y para el texto se especifico que puede incluir todas las letras del abecedario tanto en minúsculas como en mayúsculas, las

letras tildadas, ñ minúscula y mayúscula, dígitos y otros signos de puntuación o para operaciones.

➢ Clase analizador:

Esta clase está formada por varias funciones que controlan todo lo relacionado al analizador. Las clases son las siguientes:

• Método __init__:

Se inicializaron varias listas que contendrán la información relevante e importante del analizador, como una lista para errores, lista para resultados, etc. también se inicializaron variables para guardar el valor de la fila y columna en la que se encuentra la lectura, token o el error.

• Función compilador:

Recibe un atributo con la dirección del archivo a leer, luego de realizar una lectura se coloca el contenido de este en una variable, para facilitar el análisis mediante un ciclo for para leer cada carácter en la variable contenido se usara el método replace para quitar espacios y saltos de línea.

Al tener limpia la cadena se llaman a las funciones correspondientes, función tipo, función texto, función función y función estilo. Estas funciones realizaran el trabajo de analizar si la cadena esta correcta o tiene error y despues de esto se procede a instanciar a nuestro archivo para construir el html de respuestas y errores con los resultados obtenidos en el análisis para esto se usa la función constructorHTML() de archivohtml en el cual se pasan varios atributos incluida la lista de resultados, y la apariencia que se requiere para el html.

Para los errores encontrados primero se crea una tabla con la librería Graphviz y esta se convierte a una imagen que se usara después en el html, esto se hace mediante condiciones y ciclos for, creando un archivo .dot que contendrá el código de la gráfica y luego haciendo uso de os.system('dot -Tpng name. dot -o name.png') se convierte a png

• Función numero:

Recibe como atributo a la cadena, se define una lista con el patrón que deberá de tener y luego mediante ciclos se va buscando la coincidencia de este haciendo uso de la re.compile() y seach().

Si el token/ patron analizado en ese momento coincide se obtendrá ira quitando dicho patron y se amentara la fila que se esta trabajando, si el troken analizado es t_num que corresponde al valor numérico este valor sera guardado en una variable haciendo uso de group()

Se debe de mencionar que lo anterior descrito esta encerrado en un try-except, esto quiere decir que mientras lo anterior no se pueda realizar generara una excepción el cual se considera un error y por lo tanto este error debe ser guardado en una lista para su uso futuro.

El patron usado es el siguiente:

```
tokens = [
    token.t_menor.value,
    token.t_e_num.value,
    token.t_mayor.value,
    token.t_num.value,
    token.t_menor.value,
    token.t_slash.value,
    token.t_e_num.value,
    token.t_mayor.value,
]
```

- ## Función operador:

Sigue la misma lógica que el número, solo cambiando algunas cosas con respecto a ese y también cambiando su patrón por el mostrado a continuación:

```
tokens = [
    token.t_menor.value,
    token.t_e_operacion.value,
    token.t_igual.value,
    "OPERADOR",
    token.t_mayor.value,
    "NUMERO",
    "NUMERO",
    token.t_menor.value,
    token.t_slash.value,
    token.t_e_operacion.value,
    token.t_mayor.value,
]
```

Los cambios que se deben de tomar en cuenta es que mediante condiciones if se especificó que si el token leído en ese momento es operador se debe de buscar de que operador se trata, si es suma, resta, etc. y esto se guardara para posteriormente hacer las operaciones correspondientes y guardar los resultados en una lista, en caso de que no haya se genera un error que debe de guardarse en la lista de errores, la otra condición es cuando se lea el token de número, en este caso se mandara a llamar a la función número y también se añade la condición de que si en lugar de una etiqueta numero se encuentra un etiqueta de operador vuelva a ingresar a esta misma función.

- ## Función tipo:

Esta función es la principal y la única que es llamada en la función compilador y eso es porque esta contiene a la función operador y numero, llegando a tener muchas de estas, por lo tanto, el patrón de esta función es el siguiente:

```
tokens = [
    token.t_menor.value,
    token.t_e_tipo.value,
    token.t_mayor.value,
    "OPERACIONES",
    token.t_menor.value,
    token.t_slash.value,
    token.t_e_tipo.value,
    token.t_mayor.value,
]
```

La función posee la misma lógica que las anteriores, pero se usa una condición para que, al momento de leer el token de OPERACIONES, mediante un ciclo while ejecute una y otra vez a la función operador según sea necesario, esto es porque dentro de tipo puede haber muchas operaciones.

- ## Función texto_f:

Con la misma lógica anterior esta función leerá los tokens que forman su patrón y en caso de no coincidir generara un error que debe de guardarse, de igual manera se guarda lo contenido dentro de sus etiquetas. El patrón pare esta función es el siguiente:

```
tokens = [
    token.t_menor.value,
    token.t_e_texto.value,
    token.t_mayor.value,
    token.t_text.value,
    token.t_menor.value,
    token.t_slash.value,
    token.t_e_texto.value,
    token.t_mayor.value,
]
```

- ## Función funcion:

Esta función tiene la funcionalidad de contener la información necesitada para crear el archivo html de salida, y por lo tanto posee un título una descripción que es lo obtenido en la función texto y el contenido que se obtuvo de la función tipo.

Siempre llevando la misma lógica, el patrón utilizado es:

```
tokens = [
    token.t_menor.value,
    token.t_e_funcion.value,
    token.t_igual.value,
    "ESCRIBIR",
    token.t_mayor.value,
    token.t_menor.value,
    token.t_e_titulo.value,
    token.t_mayor.value,
    token.t_text.value,
    token.t_menor.value,
    token.t_slash.value,
    token.t_e_titulo.value,
    token.t_mayor.value,

    token.t_menor.value,
    token.t_e_Descrip.value,
    token.t_mayor.value,
    "\[TEXTO\]",
    token.t_menor.value,
    token.t_slash.value,
    token.t_e_Descrip.value,
    token.t_mayor.value,

    token.t_menor.value,
    token.t_e_contenido.value,
    token.t_mayor.value,
    "\[TIPO\]",
    token.t_menor.value,
    token.t_slash.value,
    token.t_e_contenido.value,
    token.t_mayor.value,

    token.t_menor.value,
    token.t_slash.value,
    token.t_e_funcion.value,
    token.t_mayor.value,
]
```

- ## Función estilo:

Tiene la misma lógica que las otras funciones, pero en esta se obtendrá la apariencia del html que se creará con los resultados y errores, por lo tanto, también se crea una lista de colores permitidos y el resultado obtenido se debe de ir guardando en una lista especial para esto.

El patrón usado, incluyendo la lista de colores permitidos es el siguiente:

```
colores = [
    "NEGRO",
    "AZUL",
    "AMARILLO",
    "ROJO",
    "VERDE",
    "MORADO",
    "ANARANJADO"
    ]
tokens = [
    token.t_menor.value,
    token.t_e_estilo.value,
    token.t_mayor.value,
    token.t_menor.value,
    token.t_e_titulo.value,
    token.t_a_color.value,
    token.t_igual.value,
    "COLOR",
    token.t_a_tam.value,
    token.t_igual.value,
    token.t_num.value,
    token.t_slash.value,
    token.t_mayor.value,

    token.t_menor.value,
    token.t_e_Descrip.value,
    token.t_a_color.value,
    token.t_igual.value,
    "COLOR",
    token.t_a_tam.value,
    token.t_igual.value,
    token.t_num.value,
    token.t_slash.value,
    token.t_mayor.value,

    token.t_menor.value,
    token.t_e_contenido.value,
    token.t_a_color.value,
    token.t_igual.value,
    "COLOR",
    token.t_a_tam.value,
    token.t_igual.value,
    token.t_num.value,
    token.t_slash.value,
    token.t_mayor.value,

    token.t_menor.value,
    token.t_slash.value,
    token.t_e_estilo.value,
    token.t_mayor.value,
]
```

- ## Funciones líneas, quitar_L, quitar_E, etiqueta:

Estas funciones servirán como apoyo en las funciones anteriormente descritas, siendo su función principal la siguiente:

La función líneas va incrementando el valor de la fila, es decir que esta será la que controlara en que fila se está analizando.

La función quitar_L tiene como función quitar de la cadena principal todos los caracteres que ya fueron analizados y coincidieron.

La función quitar_E tiene el mismo propósito que quitar_L pero este quitara aquellos caracteres que provocaron error.

La función etiqueta busca hacer una comparación entre la etiqueta que se requiere y la etiqueta en la cadena si estas coinciden se retorna un true en caso contrario un false.

archivohtml:

librerías utilizadas y la función creada son las siguientes:

```
import webbrowser

def constructorHTML(listaresultado,listaapariencia,titulop,descrip, archivo):
```

La funcion constructorHTML recibe varios atributos, etre ellos lista de resultados, atributos para el aspecto, tiulo del archivo, descripcion, etc.

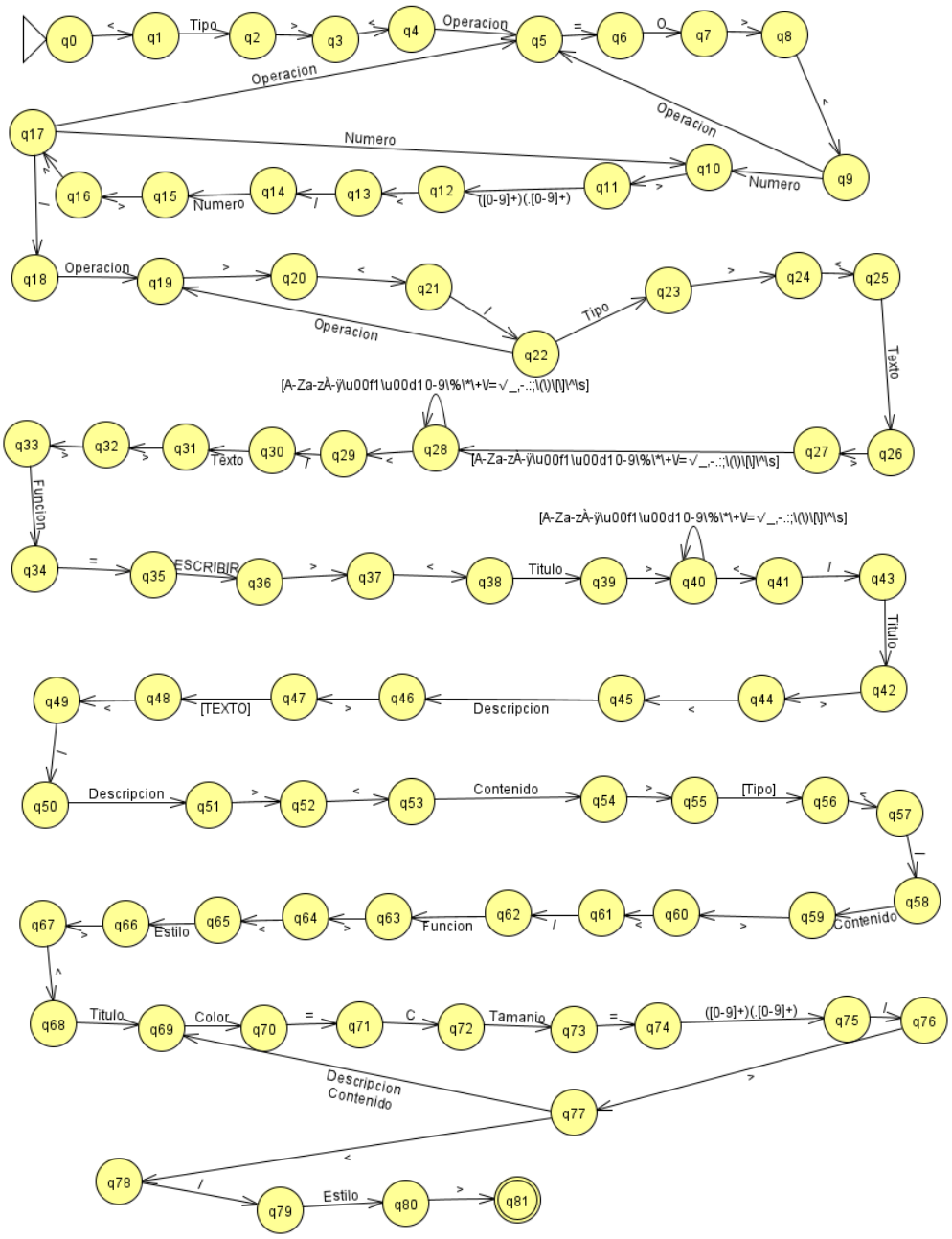
La logica detrás de esta funcion es simple, solo se realiza una concion if al inicio para determinar si se esta creando un archivo para resultados o para error y posteriormente usando la funcion open () con el atributo w+ para escribir se comienza a concatenar una serie de lineas como texto que contienen el codigo que modela la pagina html ingresando los valores de aspecto, color y tamaño obtenidos en el analisis.

Para incluir todos los resultados que se encuentran en una lista se usa un ciclo for y se van concatenando las cadenas, en caso de tratarce de la pagina de errores se tiene una condicional en donde si se trata de esta pagina se debe de concatener en lugar de los resultados una imagen png, esta ultima la creamos en la funcion de compiolador del analizador.

Definido el contenido de la pagina lo escribiremos en el archivo con un write() y cerramos el archivo con un close(), para ver estos archivos inmediantemente en un navegador se utilizo webbrowser.open_new_tab() al que se le paso como atributo el nombre del archivo.

AFD

Tras conocer un poco de la lógica detrás del analizador, para comprender de mejor manera su funcionalidad, se representa en el siguiente AFD:



Donde:
O={SUMA, RESTA, MULTIPLICACION, DIVISION, POTENCIA, RAIZ, MOD, INVERSO, SENO,COSENO,TANGENTE}

C={AZUL,ROJO,AMARILLO,VERDE,MORADO,ANARANJADO,NEGRO}

Mientras que **([0-9]+)(.[0-9]+),** es la expresión regular utilizada para los números y **[A-Za-zÀ-ÿ\u00f1\u00d10-9\%*\+\/=v_,-.:;\(\)\[\]\^\s]** para las cadenas de texto en general.