UNIVESIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE CIENCAS Y SISTEMAS
LANGUAJES FORMALES Y DE PROGRAMACION
SECCIÓN B+
PRIMER SEMESTRE 2023
AUX. DIEGO ANDRES OBIN ROSALES

MANUAL TECNICO

BRANDON EDUARDO PABLO GARCIA
202112092
Guatemala, febrero del 2023

CONTENIDO

INTRODUCCION

Este manual describe los pasos necesarios para cualquier persona que tenga ciertas bases de sistemas pueda realizar el código implementado en Python donde se crea un código para un sistema de muestras de películas utilizando POO (Programación Orientada a Objetos) de la misma manera Tkinter y así poder implementarlo de la mejor manera. El siguiente código se explicó de la manera más detalla posible para la mejor compresión de la persona.

OBJETIVOS

- Brindar la información necesaria para poder representar la funcionalidad técnica de la estructura, diseño y definición del aplicativo.

- Describir las herramientas utilizadas para el diseño y desarrollo del prototipo

CONTENIDO TECNICO

Para comenzar creamos un listado de nuestras palabras reservadas como los son palabras de “Operación” entre otras y se importaron todas las cuales se guardaron en e diccionario llamado “reserverd”

analizadorlexico.py

```python
from Instrucciones.aritmeticas import *
from Instrucciones.trigonometricas import *
from Abstract.lexema import *
from Abstract.numero import *


#Este es un listado de palabras reservadas como lo son multiplicacion y asi, RTEXTO es un token
reserved = {
    'OPERACION'         : 'Operacion',
    'RVALOR1'            : 'Valor1',
    'RVALOR2'           : 'Valor2',
    'RSUMA'             : 'Suma',
    'RMULTIPLICACION'   : 'Multiplicacion',
    'RDIVISION'         : 'Division',
    'RPOTENCIA'         : 'Potencia',
    'RRAIZ'             : 'Raiz',
    'RINVERSO'          : 'Inverso',
    'RSENO'             : 'Seno',
    'RCOSENO'           : 'Coseno',
    'RTANGENTE'         : 'Tangente',
    'RMODULO'           : 'Modulo',
    'RTEXTO'            : 'Texto',
    'RCOLORFONDONODO'   : 'Color-Fondo-Nodo',
    'RCOLORFUENTENODO'  : 'Color-Fuente-Nodo',
    'RFORMANODO'        : 'Forma-Nodo',
    'COMA'              : ',',
    'PUNTO'             : '.',
    'DPUNTO'            : ':',
    'CORI'              : '[',
    'CORD'              : ']',
    'LLAVEI'            : '{',
    'LLAVED'            : '}',
}
```

Se convirtió el diccionario a una lista de lexemas y llevamos la lista de líneas, columnas, instrucciones y “lista_lexemas”

```python
#Convertimos el diccionario de arriba en una lista con el nombre lexemas

lexemas = list(reserved.values())

#Llevamos la lsita de lineas, columnas, instrucciones y lista_lexemas
global n_lineas
global n_columnas
global instrucciones
global lista_lexemas

n_lineas = 1
n_columnas = 1
lista_lexemas = []
instrucciones = []

#Metodo que recibe una cadea donde mando a llamar a mi liena y columna

```

Se definido un método llamado “instrucciones y recibe una cada dónde se manda a llamar a la línea y columna, por lo mismo en esta parte se va leyendo partes del archivo JSON donde se usaron condicionales para ir viendo que cuando entra una palabra reservada e ir armando cada lexema. Esto fue posible por un “while” para tener ciclos y así seguir dentro de la función mientras se cumplan ciertas condiciones, como notan se usa un puntero el cual se iguala a cero para no acumular demasiada información.

```
49
50  #Metodo que recibe una cadea donde mando a llamar a mi liena y columna
51
52  def instruccion(cadena):
53      global n_lineas
54      global n_columnas
55      global lista_lexemas
```

```
70              l = Lexema(lexema, n_lineas, n_columnas)
71
72              lista_lexemas.append(l)    #Aqui armamos lexema como clase
73              n_columnas += len(lexema) +1
74              puntero = 0        #Reiniciamos porque la cadena recibida fue actualizada
75          elif char.isdigit():        #Si es un numero mandaremos a traer la cadena
76              token, cadena = armar_numero(cadena) #Mandamos la cedena como tal para no cortar nada
77              if token and cadena:
78                  n_columnas +=1
79
80                  n = Numero(token, n_lineas, n_columnas)
81
82                  lista_lexemas.append(n)
83                  n_columnas += len(str(n)) +1    #El toquen lo convertimos en un string y despues a cadena
84                  puntero = 0
85
86          elif char == '[' or char == ']':        #Si el char es igual a un corchete que cierra o abre
87
88              c = Lexema(char, n_lineas, n_columnas)
89
90              lista_lexemas.append(c)
91              cadena = cadena[1:]
92              n_columnas +=1
93              puntero = 0
94          elif char == '\t':        #En este if ingnoramos los saltos de linea
95              n_columnas +=4
96              cadena = cadena[4:]       #Cortamos la cadena con esos espacions
97              puntero = 0             #Reiniciamos el puntero
98          elif char == '\n':        #Este if es por si el char es un salto de linea
99              cadena = cadena[1:]
100             puntero = 0
101             n_lineas += 1
102             n_columnas = 1
103         else:
104             cadena = cadena[1:]
105             puntero = 0
106             n_columnas += 1
```

Se creo un método llamado “armar_lexema” aquí recorremos nuestra cadena y si leímos unas comillas se arma el lexema, para que no falle se retorna un None.

```
110   #metodo armar lexema
111   def armar_lexema(cadena):
112       global n_lineas
113       global n_columnas
114       global lista_lexemas
115       lexema = ''
116       puntero = ''
117       for char in cadena:    #Aqui recorremos nuestra cadena
118           puntero += char
119           if char == '\"':         #Ya leimos la comilla
120               return lexema, cadena[len(puntero):]
121           else:
122               lexema += char
123       return None, None #Para que no falle se retorna None None
124
```

Con el método para armar los números y sus operaciones leíamos el numero con un puntero, todo se recorrido por un “for” si eran decimales entraban a un “if” y se igual el char a ciertas circunstancias como si venia un salto de línea entre otras cosas.

```
126   #Metodo para armar los numeros y sus operaciones
127   def armar_numero(cadena):
128       numero = ''
129       puntero = ''
130       is_decimal = False      #Numeros decimales
131       for char in cadena:
132           puntero += char
133           if char == '.':
134               is_decimal = True
135           if char == '"' or char == ' ' or char == '\n' or char == '\t' or char== ']' or char== "}]":
136               if is_decimal:
137                   return float(numero), cadena[len(puntero)-1:]
138               else:
139                   return int(numero), cadena[len(puntero)-1:]
140           else:
141               numero += char
142       return None, None
```

Aquí en operaciones se define las operaciones a realizar todo esto fue posible con el métodos abstractos

```python
def operar():
    global lista_lexemas
    global instrucciones
    operacion = ''
    n1 = ''
    n2 = ''
    while lista_lexemas:
        lexema = lista_lexemas.pop(0)
        if lexema.operar(None) == 'Operacion':
            operacion = lista_lexemas.pop(0)
        elif lexema.operar(None) == 'Valor1':
            n1 = lista_lexemas.pop(0)
            if n1.operar(None) == '[':
                n1 = operar()
        elif lexema.operar(None) == 'Valor2':
            n2 = lista_lexemas.pop(0)
            if n2.operar(None) == '[':
                n2 = operar()
        #Aqui ya armamos la funcion aritmetica y nos recibe lado dercho e izquierdo osea fila y columna
        if operacion and n1 and n2:
            return Aritmeticas(n1, n2, operacion, f'Inicio: {operacion.getFila()}:{operacion.getColumna()}', f'Fin: {n2.getFila()}:{

        elif operacion and n1 and operacion.operar(None) == ('Seno' or 'Coseno' or 'Tangente'):
            return Trigonometricas(n1, operacion, f'Inicio: {operacion.getFila()}:{operacion.getColumna()}', f'Fin: {n1.getFila()}:{
    return None
```

analizadorlexico.py | abstract.py

Abstract > abstract.py > Expression > getColumna

```python
from abc import ABC, abstractmethod

class Expression(ABC):

    def __init__(self, fila, columna):
        self.fila = fila
        self.columna = columna

    @abstractmethod
    def operar(self, arbol):
        pass

    @abstractmethod
    def getFila(self):
        return self.fila

    @abstractmethod
    def getColumna(self):
        return self.columna
```

Este método realiza las operaciones donde entra al lado derecho e ingresa después y regrese por el lado izquierdo de valor, aquí fue posible por recursividad, de la misma manera se implementó para las operaciones geométricas como lo son el seno y el coseno, donde se importó la lista “math” que nos ayuda realizar dichas operaciones.

```
14      def operar(self, arbol):
15          leftValue = ''
16          rightValue = ''
17          if self.left != None:
18              leftValue = self.left.operar(arbol)   #aqui me devuelve el valor de un numero ya sea entero o decimal
19          if self.right != None:
20              rightValue = self.right.operar(arbol) #aqui me devuelve el valor de un numero ya sea entero o decimal
21
22          if self.tipo.operar(arbol) == 'Suma':
23              return leftValue + rightValue
24          elif self.tipo.operar(arbol)  == 'Resta':
25              return leftValue - rightValue
26          elif self.tipo.operar(arbol)  == 'Multiplicacion':
27              return leftValue * rightValue
28          elif self.tipo.operar(arbol)  == 'Division':
29              return leftValue / rightValue
30          elif self.tipo.operar(arbol)  == 'Modulo':
31              return leftValue % rightValue
32          elif self.tipo.operar(arbol)  == 'Potencia':
33              return leftValue ** rightValue
34          elif self.tipo.operar(arbol)  == 'Raiz':
35              return leftValue ** (1/rightValue)
36          elif self.tipo.operar(arbol)  == 'Inverso':
37              return 1/leftValue
38          else:
39              return None
40
41      def getFila(self):
42          return super().getFila()
43
44      def getColumna(self):
45          return super().getColumna()
```

Interfaz:

Para el uso de una interfaz se importaron todas las librerías de tkinter posibles para definir un método llamado “Pantalla_Inicial” donde nos despiegla nuestra interfaz hacia el usuario en cuestión. Se coloco el nombre y el tamaño de la pantalla

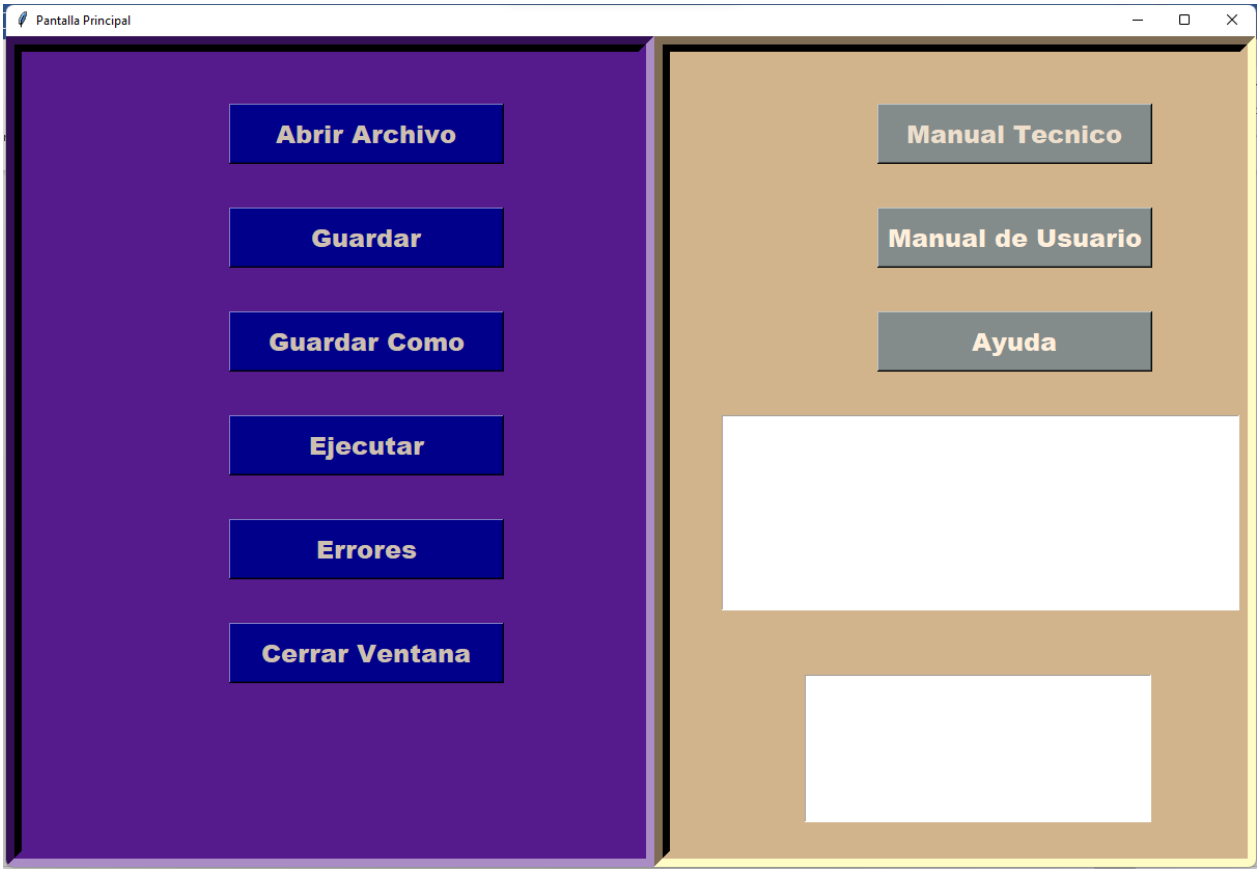
```
15  class Pantalla_Principal():
16
17      def __init__(self):
18          self.PP = Tk()
19          self.PP.title("Pantalla Principal")
20          self.PP.geometry("1000x700")
21          self.PP.configure(bg = "#102027")
22          self.PP
23          self.pantalla_1()
24
25
```

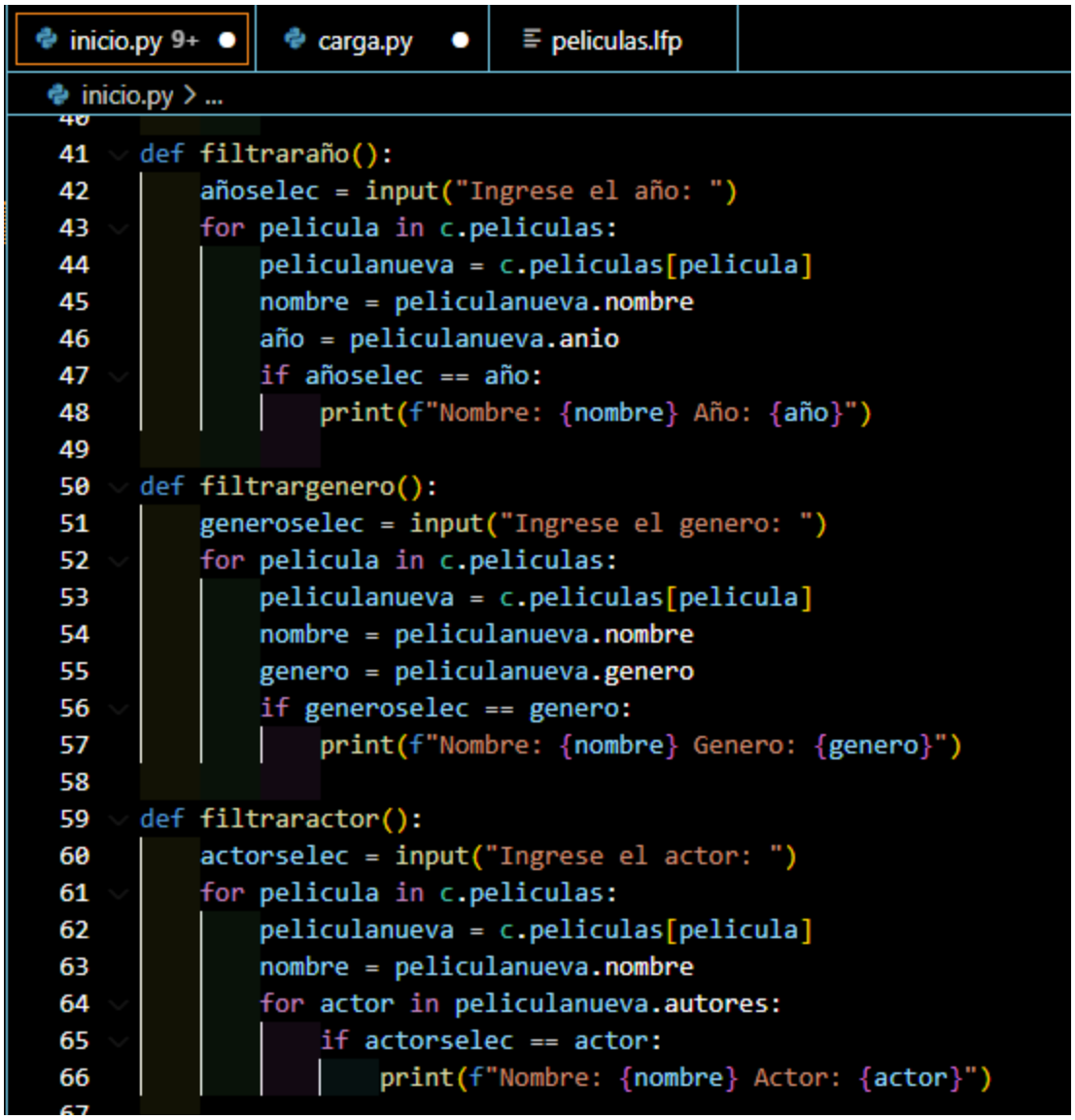
Para los Frames se necesitaron botones donde cada uno cuenta con un comando que les ayuda a realizar dichas operaciones y mostrar de manera mas cómoda al usuario

como usar la interfaz gráfica en cuestión. Con el método “Button” se crearon los botones y se les dio color junto con su forma.

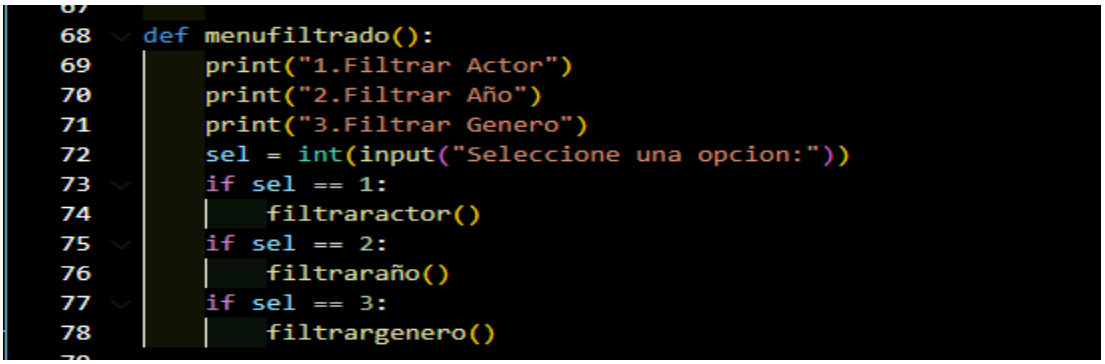
```
36      Button(self.Frame, command=self.abrir_archivo ,text="Abrir Archivo", font=("Arial Black", 18), fg="AntiqueWhite3", bg="blue4"
37
38      Button(self.Frame ,text="Guardar", font=("Arial Black", 18),fg="AntiqueWhite3", bg="blue4", width=15).place(x=200, y=150)
39
40      Button(self.Frame, command=self.guardar_como ,text="Guardar Como", font=("Arial Black", 18),fg="AntiqueWhite3", bg="blue4", w
41
42      Button(self.Frame, command=self.ejecutar ,text="Ejecutar", font=("Arial Black", 18), fg="AntiqueWhite3", bg="blue4", width=15
43
44      Button(self.Frame,text="Errores", font=("Arial Black", 18), fg="AntiqueWhite3", bg="blue4", width=15).place(x=200, y=450)
45
46      Button(self.Frame, text="Cerrar Ventana", command=self.PP.destroy, font=("Arial Black", 18), fg="AntiqueWhite3", bg="blue4",
47
```

La interfaz seria la siguiente:





Y debajo de eso encontramos la función donde definimos como será el filtrado usando prints e ifs para tener una mejor resultado, usado en “int()” en el input para que aceptara caracteres numéricos.



Y debajo de eso encontramos la función donde definimos como será el filtrado usando prints e ifs para tener una mejor resultado, usado en “int()” en el input para que aceptara caracteres numéricos. Y por últimos pero no menos importante tenemos la función “mostraractores()” donde con un for nos recorrerá nuestra lista denominada c.peliculas y así obtener en nombre de dicho actor. La variable elección nos ayuda a escoger una película donde saldrá el actor, pero ojo tiene que colocar en nombre de la película. El resultado lo imprimirá con el nombre de la película y el actor.

```
78          filtrargenero()
79
80      def mostraractores():
81
82          for pelicula in c.peliculas:
83              peliculanueva = c.peliculas[pelicula]
84              nombre = peliculanueva.nombre
85              print(nombre)
86
87          eleccion = input("Seleccione una pelicula: ")
88          if eleccion in c.peliculas:
89              peliculanueva = c.peliculas[eleccion]
90              nombre = peliculanueva.nombre
91              for actor in peliculanueva.autores:
92                  print(f"Nombre: {nombre} Actor: {actor}")
93
```

AFD del analizador léxico: