UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

ING. EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

6 “M”

COMPILADORES

SUBCOMPETENCIA I.- ANÁLISIS LÉXICO

ACT. 1.3 PRÁCTICA. UNIDAD 1. REALIZA UN ANALIZADOR LÉXICO EN PYTHON

ALUMNO: MARCO ANTONIO ZÚÑIGA MORALES – A211121

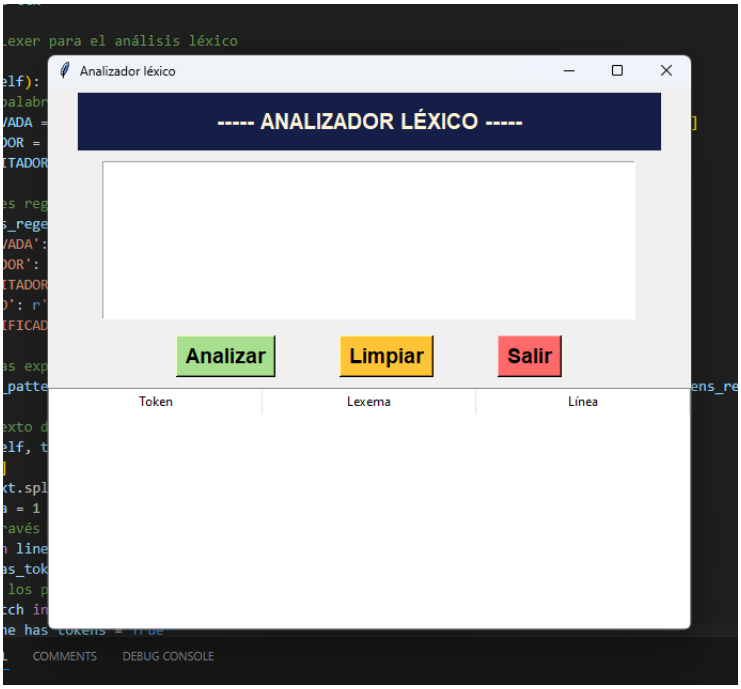
DOCENTE: DR. LUIS GUTIÉRREZ ALFARO

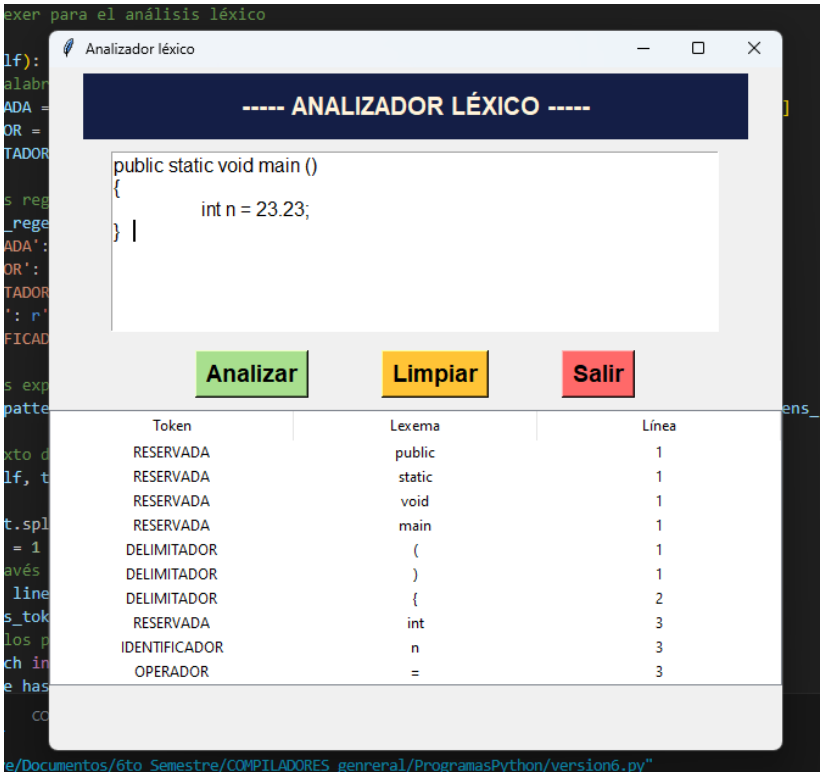
TUXTLA GUTIÉRREZ, CHIAPAS
JUEVES, 24 DE AGOSTO DE 2023

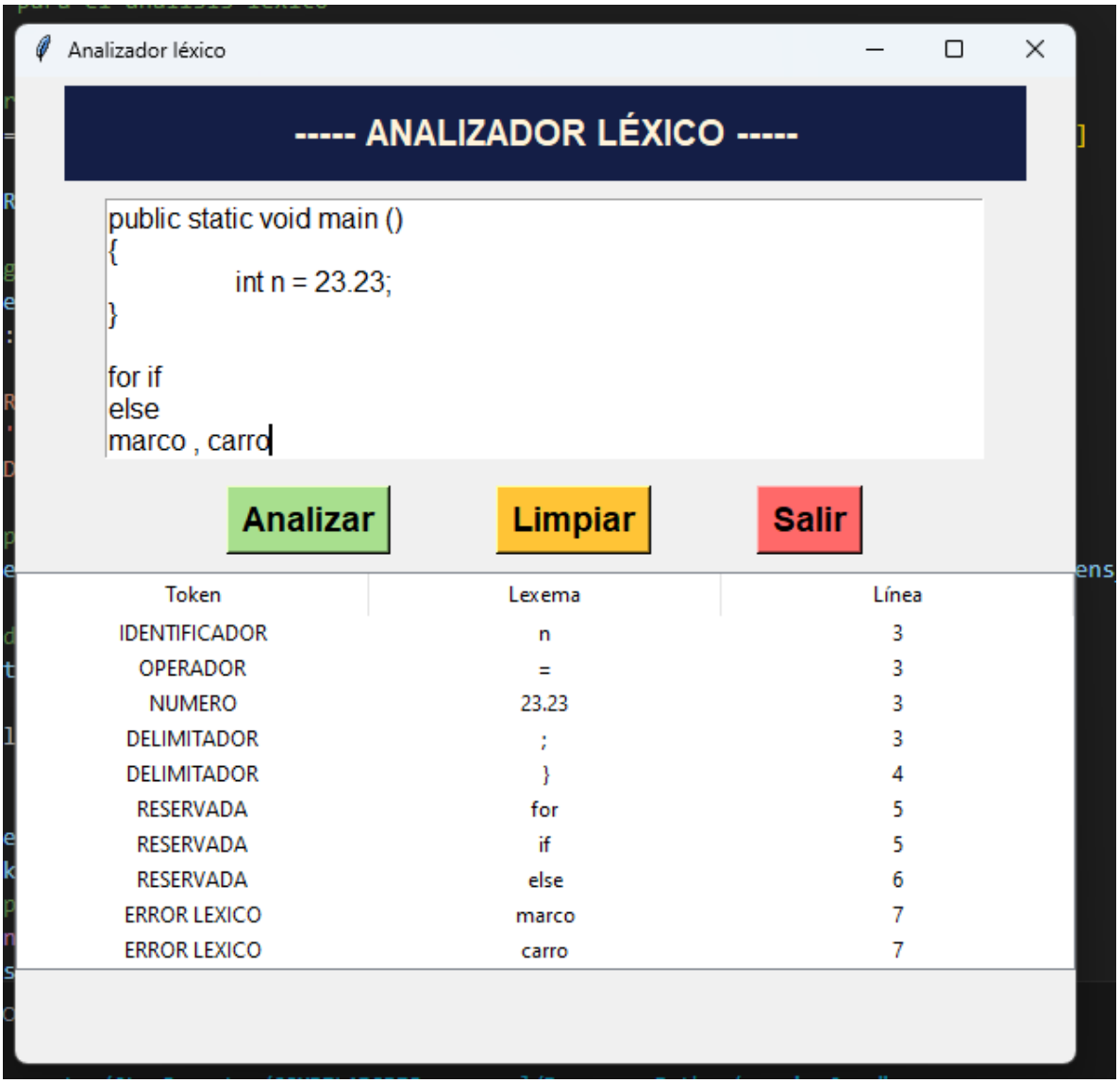
Act. 1.3 Práctica. Unidad 1. Realiza un analizador Léxico en python

```python
#Librerias
import tkinter as tk
import re
from tkinter import ttk

#Define una clase Lexer para el análisis léxico
class Lexer:
    def __init__(self):
        #Lista de palabras reservadas, operadoes y delimitadores
        self.RESERVADA = ['for', 'do', 'while', 'if', 'else', 'public', 'static', 'void', 'int','main']
        self.OPERADOR = ['=', '+', '-', '*', '/']
        self.DELIMITADOR = ['(', ')', '{', '}', ';']

        #Expresiones regulares para patrones de tokens
        self.tokens_regex = {
            'RESERVADA': '|'.join(r'\b' + re.escape(keyword) + r'\b' for keyword in self.RESERVADA),
            'OPERADOR': '|'.join(map(re.escape, self.OPERADOR)),
            'DELIMITADOR': '|'.join(map(re.escape, self.DELIMITADOR)),
            'NUMERO': r'\d+(\.\d+)?',   #Patrón para números enteros y decimales
            'IDENTIFICADOR': r'[A-Za-z_]+'  #Patrón para identificadores
        }
        #Compila las expresiones regulares en un patrón combinado
        self.token_patterns = re.compile('|'.join(f'(?P<{t}>{self.tokens_regex[t]})' for t in self.tokens_regex))

    #Tokeniza el texto de entrada
    def tokenize(self, text):
        tokens = []     #Lista para almacenar los tokens
        lines = text.split('\n') #Divide el texto de entrada en líneas
        NumeroLinea = 1  #Inicializa el número de línea
        #Itera a través de cada línea del texto
        for line in lines:
            line_has_tokens = False  #Bandera para verificar si la línea tiene tokens
            #Busca los patrons de tokens en la línea actual
            for match in self.token_patterns.finditer(line):
                line_has_tokens = True
                #Itera a través de los grupos coincidentes de la expresión regular
                for token_type, token_value in match.groupdict().items():
                    #Verifica el tipo de token y su longitud
                    if token_type == 'IDENTIFICADOR' and token_value and len(token_value) > 1:
                        tokens.append((NumeroLinea, 'ERROR LEXICO', token_value))    #Agrega un token de error léxico
                    elif token_type == 'IDENTIFICADOR' and token_value and len(token_value) == 1:
                        tokens.append((NumeroLinea, 'IDENTIFICADOR', token_value))  #Agrega un token identificador
                    elif token_value:
                        tokens.append((NumeroLinea, token_type, token_value))  #Agrega un token con su tipo y valor
            #Si la linea tiene tokens, incrementa el número de línea
            if line_has_tokens:
                NumeroLinea += 1
        return tokens  #Devuelve la lista de tokens resultante

    #Realiza el análisis de los tokens y genera una cadena de resultados
    def analyze(self, text):
        tokens = self.tokenize(text) #Tokeniza el texto de entrada
        result = "Token\t\tLexema\t\tLinea\n"  #Crea una cadena para almacenar los resultados formateados
        for line_number, token_type, token_value in tokens:  #Itera a través de la lista de tokens generados
            result += f"{token_type}\t\t{token_value}\t\t{line_number}\n"   #Agrega una línea formateada al resultado
        return result  #Devuelve la cadena de resultados
```

```python
#Define una clase LexerApp para la interfaz gráfica de usurio
class LexerApp:
    def __init__(self):  #Define una clase llamda LexerApp para la interfaz grafica de usuario
        self.windows = tk.Tk() #Crea una ventana de la clase
        self.windows.title("Analizador léxico") #Establece el titulo de la ventana

        #Crea una etiqueta para el titulo de la aplicacion
        self.text_label = tk.Label(text="----- ANALIZADOR LÉXICO -----", height=2, width=45, font=("Arial", 15, 'bold'), fg="#FFF3DA",
bg="#141E46")
        self.text_label.pack(pady=5)

        #Crea un cuadro de texto para la entrada de texto
        self.text_input = tk.Text(self.windows, height=8, width=55, font=("Arial", 12))
        self.text_input.pack(pady=5)

        #Crea un marco para los botones
        self.button_frame = tk.Frame(self.windows)
        self.button_frame.pack()

        #crea un boton para realizar el analisis lexico del texto de entrada
        self.analyze_button = tk.Button(self.button_frame, text="Analizar", command=self.analyze_text, bg="#A8DF8E", font=("Arial",
14, 'bold'))
        self.analyze_button.grid(row=0, column=0, padx=30, pady=10)

        #crea un boton para limpiar el cuadro de texto
        self.clean_button = tk.Button(self.button_frame, text="Limpiar", command=self.clean_text, bg="#FFC436", font=("Arial", 14,
'bold'))
        self.clean_button.grid(row=0, column=1, padx=30, pady=10)

        #Crea un boton para salir del programa
        self.exit_button = tk.Button(self.button_frame, text="Salir", command=self.exit_app, bg="#FF6969", font=("Arial", 14, 'bold'))
        self.exit_button.grid(row=0, column=2, padx=30, pady=10)

        #Crea un widget TreeView para mostrar los resultados de análisis
        self.treeview = ttk.Treeview(self.windows, columns=("Token", "Lexema", "Línea"), show="headings")
        self.treeview.heading("Token", text="Token")
        self.treeview.heading("Lexema", text="Lexema")
        self.treeview.heading("Línea", text="Línea")
        self.treeview.pack()

        # Configura la alineación de las columnas para centrar el contenido
        self.treeview.column("Token", anchor="center")
        self.treeview.column("Lexema", anchor="center")
        self.treeview.column("Línea", anchor="center")

    #Define un metodo para realizar el analisis lexico del texto de entrada y mostrar los resultados
    def analyze_text(self):
        lexer = Lexer()  #Crea una instancia de la clase Lexer
        text = self.text_input.get("1.0", "end")
        result = lexer.tokenize(text)

        # Limpia las entradas existentes en el Treeview
        self.treeview.delete(*self.treeview.get_children())

        # Inserta los resultados en el Treeview
        for line_number, token_type, token_value in result:
            self.treeview.insert("", "end", values=(token_type, token_value, line_number))

    #Define un metodo para limpiar el cuadro de texto y la etiqueta de resultados
    def clean_text(self):
        self.text_input.delete("1.0", "end") #Borra el contenido del cuadro del texto
        self.treeview.delete(*self.treeview.get_children()) #Limpia el contenido del ttk

    def exit_app(self): #Define un metodo para salir del programa
        self.windows.destroy()

    def run(self): #Define un metodo para ejecutar la aplicacion de la interfaz grafica
        self.windows.mainloop() #inicia el bucle de la interfaz grafica

app = LexerApp() #Crea una instancia de la clase Lexer app
app.run()   #Ejecuta la aplicacion de la interfaz grafica llamando al metodo "Run"
```


Analizador léxico
----- ANALIZADOR LÉXICO -----
Analizar
Limpiar
Salir
Token
Lexema
Línea


Analizador léxico
----- ANALIZADOR LÉXICO -----
public static void main ()
{
int n = 23.23;
}
Analizar
Limpiar
Salir
Token Lexema Línea
RESERVADA public 1
RESERVADA static 1
RESERVADA void 1
RESERVADA main 1
DELIMITADOR (1
DELIMITADOR) 1
DELIMITADOR { 2
RESERVADA int 3
IDENTIFICADOR n 3
OPERADOR = 3


Analizador léxico
----- ANALIZADOR LÉXICO -----
public static void main ()
{
int n = 23.23;
}
for if
else
marco , carro
Analizar
Limpiar
Salir
Token
Lexema
Línea
IDENTIFICADOR
n
3
OPERADOR
=
3
NUMERO
23.23
3
DELIMITADOR
;
3
DELIMITADOR
}
4
RESERVADA
for
5
RESERVADA
if
5
RESERVADA
else
6
ERROR LEXICO
marco
7
ERROR LEXICO
carro
7