A black and white image of a yin yang symbol

Description automatically generatedA black rectangular sign with a map and wings

Description automatically generated

**Universidad Autónoma de Chiapas**

**Facultad de contaduría y administración C-I**

**Carrera:**

Lic. En ing en desarrollo y tecnologías de software.

**Materia:**

Compiladores.

**Catedrático:**

Mtro. Luis Gutiérrez Alfaro.

**Nombre del alumno:**

González Aguilar Eduardo - A211154

**Semestre:** 6.  **Grupo:** M.

**Nombre de la actividad:**

Ejercicios léxico de expresiones regulares.

**Fecha de entrega:**

25/08/2023.

import tkinter as tk

class Lexer:

    def \_\_init\_\_(self):

        self.reservadas = ['for','if','do','while','else']

        self.apertura = ['(']

        self.cierre = [')']

    def tokenize (self, text):

        self.tokens = self.reservadas + self.apertura + self.cierre

        arreglo = []

        current\_token = ""

        for char in text:

            if char in self.tokens:

                if current\_token != "":

                    arreglo.append(current\_token)

                current\_token = ""

                arreglo.append(char)

            else:

                if char.isspace():

                    if current\_token != "":

                        arreglo.append(current\_token)

                    current\_token = ""

                else:

                    current\_token += char

        if current\_token != "":

            arreglo.append(current\_token)

        return arreglo

//En esta parte del código declaramos nuestras palabras reservadas y nuestros paréntesis y le decimos al programa que los defina como token para así posteriormente los reconozca con sus respectivos nombres

def analyze(self, text):

        arreglo = self.tokenize(text)

        result = ""

        for token in arreglo:

            if token in self.reservadas:

                result += f"{token} | Palabra reservada\n"

            elif token in self.apertura:

                result += f"{token} | Parentecis de apertura\n"

            elif token in self.cierre:

                result += f"{token} | Parentecis de cierre\n"

            else:

                result += f"{token} | Error lexico x\_x\n"

        return result

//En esta parte del programa le decimos que reconozca cada uno de los nombres que le pusimos con anterioridad a nuestros tokens y si en dado caso colocamos una palabra que no declaramos somo nuestros tokens que me mande que es un “Error Léxico”

class LexerApp:

    def \_\_init\_\_(self):

        self.window = tk.Tk()

        self.window.title("ANALIZADOR LEXICO 😎")

        self.text\_input = tk.Text(self.window, height=10, width=50)

        self.text\_input.pack()

        self.analyze\_button = tk.Button(self.window, text="Analizar", command=self.analyze\_text)

        self.analyze\_button.pack()

        self.clean\_button = tk.Button(self.window, text="Limpiar", command=self.clean\_text)

        self.clean\_button.pack()

//En esta parte iniciamos otra clase para nuestros botones y titulo del programa.

En el caso del titulo ajustamos la altura y la anchura para que no nos ocupe tanto espacio, en el caso del botón “Analizar”, declaramos analyze\_button/text para que pueda analizar el texto y en caso de “Limpiar” declaramos clean\_button/text para que puedo eliminar los que hay dentro de nuestro programa

frame = tk.Frame(self.window)

        frame.pack()

        self.text\_label = tk.Label(frame, text="Linea", anchor='w')

        self.text\_label.pack(side="left")

        self.text\_label = tk.Label(frame, text="| Lexema |", anchor='center')

        self.text\_label.pack(side="left")

        self.text\_label = tk.Label(frame, text="Token", anchor='e')

        self.text\_label.pack(side="left")

        self.result\_label = tk.Label(self.window, text=" ", height=25, width=50)

        self.result\_label.pack()

//En este caso decalramos “Frame” para poder colocar los subtítulos debajo de los botones simétricamente. En el caso de “Linea” colocamos el frame hacia la izquierda, en el caso de “Lexema” puse dos separadores para reconocer que los resultados ya tiene un lugar asignado y le puse center para que este valla en medio y se pueda visualizar los operadores. En el caso de “Token” se coloco hacia la izquierda para poner reconocer el valor de los lexema y por ultimo en el label del resultado le puse una altura de “25” para que se pudiera ver bien el programa.

 def analyze\_text(self):

        lexer = Lexer()

        text = self.text\_input.get("1.0", "end-1c")  # Corregido para evitar el último salto de línea

        lines = text.split('\n')

        results = []

        for line\_number, line in enumerate(lines, start=1):

            result\_line = lexer.analyze(line)

            results.append(f"{line\_number} | {result\_line}")

        final\_result = "\n".join(results)

        self.result\_label.config(text=final\_result)

//En este apartado es para reconocer el que posición se encuentra cada palabra y al momento de dar el resultado final separe las palabras por orden que en cada línea se encuentra

 def clean\_text(self):

        self.text\_input.delete("1.0", "end")

        self.result\_label.config(text="")

    def run(self):

        self.window.mainloop()

    def run(self):

        self.window.mainloop()

app = LexerApp()

app.run()

//Y como último paso tenemos lo que es el clean para que pueda eliminar correctamente cada palabra y al final tenemos el código que nos sirve para que peda correr el programa.

Adjunto pruebas de escritorio

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated