Manual Tecnico

Clase lexer:

En esta clase se tienen las dos principales funciones para que el analizador léxico pueda traducir correctamente el archivo html el cual se carga dentro del programa.

Funcion Tokenizar:

Este método toma un texto como entrada y lo divide en tokens según ciertos patrones definidos. Utiliza expresiones regulares para identificar patrones como identificadores, cadenas de texto, símbolos de puntuación, etc. También maneja errores de formato y sintaxis y los almacena en self.errors.

```
def tokenizar(self, text):
    self.initValues()

    for char in text:
        if self.wait:
            self.wait = False
            if char in self.spaces or char in self.saltos or char in self.limiters:
                self.saveToken(self.temporal)

                if char in self.limiters:
                    self.buffer = char
                    self.saveToken(self.limiters[char])
                continue
            else:
                self.temporal = ""

        self.buffer += char
        self.comment()

        if self.ignore(char):
            self.buffer = self.buffer[:-1]
            continue
```

Funcion Save_Token:

Este método que guarda los strings ingresados en un token si este token pertenece al grupo de palabras.

```
def saveToken(self, tipo):
    self.tokens.append(
        {(variable) buffer: Literal['']o, "x": self.x, "y": self.y}
    )
    self.buffer = ""
```

Funcion Ignore:

Este método sirve para omitir los saltos de línea y los espacios que presenta el archivo a analizar.

```
def ignore(self, char):
    if char in self.spaces:
        self.x += self.spaces[char]
        return True
    if char in self.saltos:
        self.x = 1
        self.y += self.saltos[char]
        return True

    return False
```

Funcion generar_html:

Este método genera un documento HTML basado en el diccionario JSON construido. Define estilos CSS en línea y recorre el JSON para generar etiquetas HTML correspondientes.

```
def generarHtml(self):
    json_data = self.json_data
    if json_data == {}:
        return ""
    help = 0
    html = "<html>\n"

    inicio = json_data["inicio0"]

    html += """<style>
            table {
                font-family: Arial, sans-serif;
                border-collapse: collapse;
                width: 100%;
            }
            th, td {
                border: 1px solid black;
                text-align: left;
                padding: 8px;
            }
            th {
```

Funcion generar_reporte_html:

Este método genera dos informes HTML: uno para los tokens analizados y otro para los errores encontrados durante el análisis. Cada informe muestra detalles como el tipo de token, el lexema, la línea y la columna en la que se encuentra.

```
def generar_reporte_html(self):
    # add utf-8 encoding
    html = '''
    <html lang="es">
    <meta charset="UTF-8">
    <head>
        <style>
            table {
                font-family: Arial, sans-serif;
                border-collapse: collapse;
                width: 100%;
            }
            th, td {
                border: 1px solid #dddddd;
                text-align: left;
                padding: 8px;
            }
            th {
                background-color: #f2f2f2;
            }
        </style>
    </head>
    <body>
        <h2>Reporte de Tokens</h2>
        <table>
            <tr>
```

Clase App:

En esta clase se ubican todos los comandos e instrucciones de posición para que el programa funcione y tenga estética usando los custom themes de la librería autorizada (tkinter).

```
class App(customtkinter.CTk):
    def __init__(self):
        super().__init__()

        self.tokens = []

        self.title("Proyecto 1 - Lenguajes")
        self.geometry(f"{1100}x{580}")

        self.grid_columnconfigure(1, weight=1)
        self.grid_columnconfigure((2, 3), weight=0)
        self.grid_rowconfigure((0, 1, 2), weight=1)

        self.sidebar_frame = customtkinter.CTkFrame(self, width=140, corner_radius=0)
        self.sidebar_frame.grid(row=0, column=0, rowspan=4, sticky="nsew")
        self.sidebar_frame.grid_rowconfigure(4, weight=1)
        self.logo_label = customtkinter.CTkLabel(self.sidebar_frame, text="Traductor", font=customtkinter.CTkFon
        self.logo_label.grid(row=0, column=0, padx=20, pady=(20, 10))
        self.sidebar_button_1 = customtkinter.CTkButton(self.sidebar_frame, text="Abrir archivo", command=self.a
        self.sidebar_button_1.grid(row=1, column=0, padx=20, pady=10)
        self.sidebar_button_2 = customtkinter.CTkButton(self.sidebar_frame, text="Reportes", command=self.report
        self.sidebar_button_2.grid(row=2, column=0, padx=20, pady=10)

        self.entry = customtkinter.CTkTextbox(self, width=250)
        self.entry.grid(row=2, column=1, columnspan=2, padx=(20, 0), pady=(20, 20), sticky="nsew")
```

Función abrir_archivo:

Abre un cuadro de diálogo para seleccionar un archivo y luego lee su contenido y lo muestra en un cuadro de texto.

```
def abrir_archivo(self):
    ruta_archivo = filedialog.askopenfilename(title="Seleccione un archivo")
    if ruta_archivo:
        with open(ruta_archivo, "r", encoding="utf-8") as archivo:
            contenido = archivo.read()
            self.textbox.insert("1.0", contenido)
```

Función Reportes:

Genera informes HTML utilizando el objeto Analizador y los métodos definidos en la clase Analizador.

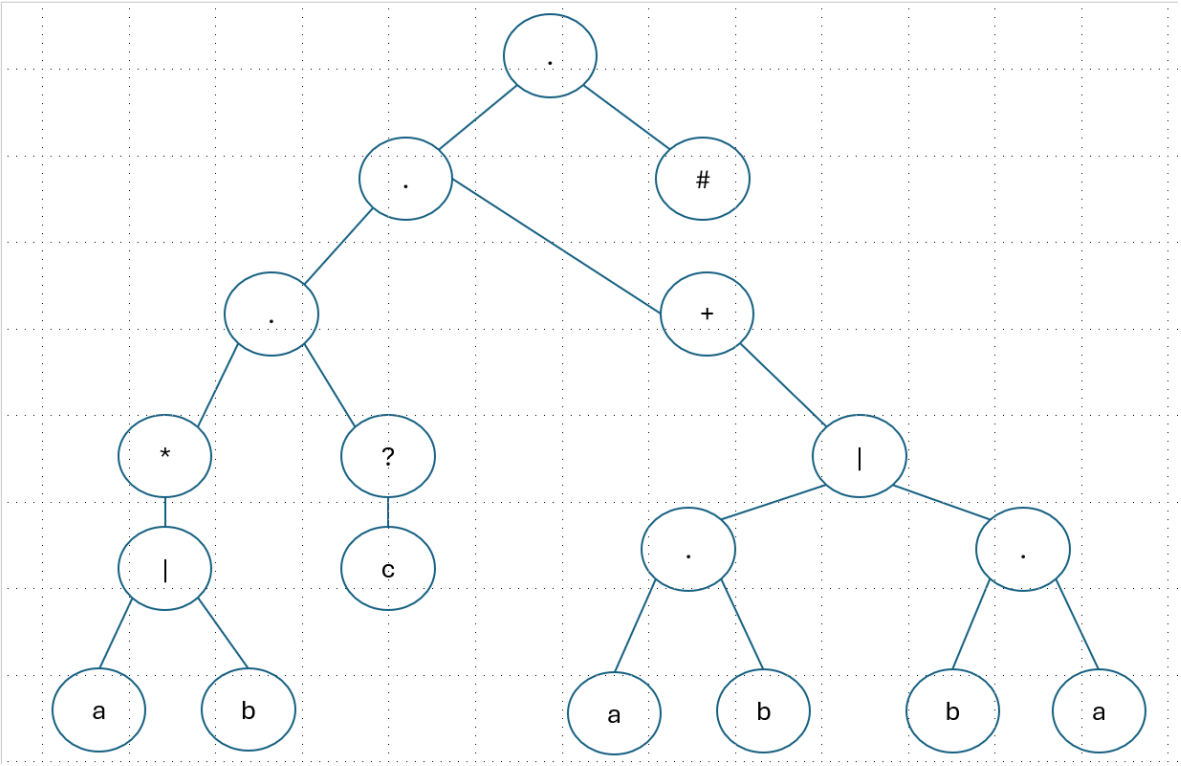
```
def reportes(self):
    self.analizador.generar_reporte_html()
```

Función analizar:

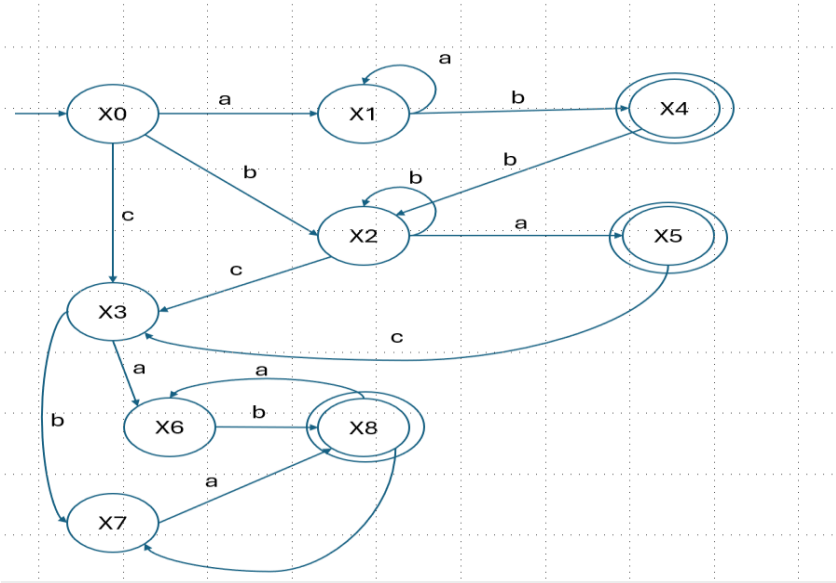
Obtiene el texto del cuadro de texto, lo pasa al objeto Analizador para realizar el análisis y muestra el resultado en otro cuadro de texto.

```
def analizar(self):
    base = ["id", "igual", "nueva"]
    basefinal = ["string", "parentesis_cerrado", "punto_y_coma"]
    grammar = [
        ["crearbd"] + base + ["crearbd"] + ["parentesis_abierto"] + ["parentesis_cerrado", "punt
        ["eliminarbd"] + base + ["eliminarbd"] + ["parentesis_abierto"] + ["parentesis_cerrado",
        ["crearcoleccion"] + base + ["crearcoleccion"] + ["parentesis_abierto"] + basefinal,
        ["eliminarcoleccion"] + base + ["eliminarcoleccion"] + ["parentesis_abierto"] + basefina
        ["insertarunico"] + base + ["insertarunico"] + ["parentesis_abierto"] + ["string"] + ["c
        ["actualizarunico"] + base + ["actualizarunico"] + ["parentesis_abierto"]  + ["string"]
        ["eliminarunico"] + base + ["eliminarunico"] + ["parentesis_abierto"] + ["string"] + ["c
        ["buscartodo"] + base + ["buscartodo"] + ["parentesis_abierto"] + basefinal,
        ["buscarunico"] + base + ["buscarunico"] + ["parentesis_abierto"] + basefinal,
    ]

    sentence = None
    sentenceResult = []
    sentenceSize = 0
    actualSize =1

    self.sentences = []
    self.errors = []
```

Árbol Binario



Autómata Finito Determinista



Expresión Regular

"([^"]*)" Cadenas de caracteres

---.* Comentarios simples

\/[*][\s\S]*?[*]\/ Comentarios de bloque

=|:|,|\{|\}|\[|\]|;|\(|\) Operadores y delimitadores

[\t]+|\n|\r Espacios y saltos de línea

^[a-zA-ZñÑ]+$ Identificadores

Gramática Libre de Contexto

S -> sentencia

sentencia -> "crearbd" base "crearbd" "(" ")" ";"

| "eliminarbd" base "eliminarbd" "(" ")" ";"

| "crearcoleccion" base "crearcoleccion" "(" basefinal

| "eliminarcoleccion" base "eliminarcoleccion" "(" basefinal

| "insertarunico" base "insertarunico" "(" "string" "," basefinal

| "actualizarunico" base "actualizarunico" "(" "string" "," basefinal

| "eliminarunico" base "eliminarunico" "(" "string" "," basefinal

| "buscartodo" base "buscartodo" "(" basefinal

| "buscarunico" base "buscarunico" "(" basefinal

base -> "id" | "igual" | "nueva"

basefinal -> "string" | "parentesis_cerrado" | "punto_y_coma"