Nombre: Estiben Yair Lopez Leveron **Carnet:** 202204578

Manual técnico

Este manual técnico proporciona una guía detallada sobre el código utilizado en el programa [nombre del programa]. El objetivo principal es explicar el uso de un arreglo que contiene las palabras reservadas necesarias para el funcionamiento correcto del software.

El código presentado a continuación se divide en secciones que explican paso a paso cómo se define y utiliza este arreglo, crucial para el reconocimiento y almacenamiento de las palabras reservadas.

Interfaz

1. **Clase interfaz**:
    - Esta clase representa la interfaz gráfica de la aplicación.
    - El método **__init__(self)** inicializa la ventana principal (**self.ventana**) con un título, dimensiones y un color de fondo definido como azul marino (**#0B3954**).
    - Se crea un menú con opciones como "Nuevo", "Abrir archivo", "Guardar", "Guardar como" y "Salir". Cada opción tiene asociada una función específica que se ejecuta al ser seleccionada.
    - Se crean botones para realizar acciones como "Análisis", "Tabla Tokens" y "Tabla Errores". Estos botones tienen un diseño específico en cuanto a tamaño, color de fondo y texto.
    - Se crean dos áreas de texto desplazables (**scrolledtext.ScrolledText**) para mostrar contenido o resultados.
    - Finalmente, se inicia el bucle principal (**mainloop()**) para que la interfaz gráfica sea interactiva.
2. **Métodos dentro de la clase interfaz**:
    - **agregar_menu(self)**: Agrega un menú similar al del método **__init__**, pero aquí se define como un método separado para mantener el código organizado.
    - **nuevo(self)**: Borra el contenido de ambas áreas de texto cuando se selecciona la opción "Nuevo" en el menú.
    - **abrir_archivo(self)**: Permite al usuario seleccionar y abrir un archivo de texto, mostrando su contenido en el área de texto principal.

- **guardar_como(self)**: Permite al usuario guardar el contenido del área de texto en un archivo seleccionado por él.
- **salir(self)**: Cierra la aplicación al seleccionar la opción "Salir" en el menú.
- **analizar(self)**: Realiza un análisis sintáctico del texto ingresado en el área principal y muestra los resultados en el área secundaria.
- **tabla_errores(self)** y **tabla_tokens(self)**: Abren archivos HTML (presumiblemente tablas de errores y tokens) en el navegador web predeterminado.

3. **Instancia de la clase**:
    - Al final del código, se crea una instancia de la clase **interfaz** llamada **aplicacion**, lo que inicia la interfaz y muestra la ventana principal.

```
class interfaz:

    def __init__(self):
        self.ventana = Tk()
        self.ventana.title("Aplicación de Análisis")
        self.ventana.geometry("800x600")

        # Fondo azul marino
        self.ventana.configure(bg="#0B3954")

        self.agregar_menu()

        # Frame para botones y menú
        self.frame_menu = Frame(self.ventana, bg="#0B3954")  # Fondo
azul marino para el frame
        self.frame_menu.pack(side=TOP, fill=X)

        # Menú
        menubar = Menu(self.frame_menu, bg="#0B3954", fg="white")  #
Fondo azul marino y texto blanco para el menú
        self.ventana.config(menu=menubar)
        opciones = Menu(menubar, tearoff=0)
        opciones.add_command(label="Nuevo", command=self.nuevo)
        opciones.add_command(label="Abrir archivo",
command=self.abrir_archivo)
        opciones.add_command(label="Guardar")
        opciones.add_command(label="Guardar como",
command=self.guardar_como)
```

```
        opciones.add_command(label="Salir", command=self.salir)
        menubar.add_cascade(label="Menu", menu=opciones)

        # Botones en el frame
        btn_analisis = Button(self.frame_menu, text='Análisis',
width=15, height=2, command=self.analizar, bg="#1B4F72",
fg="white")  # Fondo azul oscuro y texto blanco para el botón
        btn_analisis.pack(side=LEFT, padx=10, pady=10)
        btn_analisis.configure(font=('Arial', 12, 'bold'))

        btn_tokens = Button(self.frame_menu, text='Tabla Tokens',
width=15, height=2, command=self.tabla_tokens, bg="#1B4F72",
fg="white")  # Fondo azul oscuro y texto blanco para el botón
        btn_tokens.pack(side=LEFT, padx=10, pady=10)
        btn_tokens.configure(font=('Arial', 12, 'bold'))

        btn_errores = Button(self.frame_menu, text='Tabla Errores',
width=15, height=2, command=self.tabla_errores, bg="#1B4F72",
fg="white")  # Fondo azul oscuro y texto blanco para el botón
        btn_errores.pack(side=LEFT, padx=10, pady=10)
        btn_errores.configure(font=('Arial', 12, 'bold'))

        # Elementos de texto
        self.scrolledtext = scrolledtext.ScrolledText(self.ventana,
width=80, height=20, bg="#154360", fg="white")  # Fondo azul marino
oscuro y texto blanco para el área de texto
        self.scrolledtext.pack(expand=YES, fill=BOTH, padx=10,
pady=10)

        self.scrolledtext1 = scrolledtext.ScrolledText(self.ventana,
width=80, height=10, bg="#154360", fg="white")  # Fondo azul marino
oscuro y texto blanco para el área de texto
        self.scrolledtext1.pack(expand=YES, fill=BOTH, padx=10,
pady=10)

        self.ventana.mainloop()

    def agregar_menu(self):
        menubar = Menu(self.ventana)
        self.ventana.config(menu=menubar)
        opciones = Menu(menubar, tearoff=0)
        opciones.add_command(label="Nuevo", command=self.nuevo)
```

```
        opciones.add_command(label="Abrir archivo",
command=self.abrir_archivo)
        opciones.add_command(label="Guardar")
        opciones.add_command(label="Guardar como",
command=self.guardar_como)
        opciones.add_command(label="Salir", command=self.salir)
        menubar.add_cascade(label="Menu", menu=opciones)

    def nuevo(self):
        self.scrolledtext.delete("1.0", END)
        self.scrolledtext1.delete("1.0", END)

    def abrir_archivo(self):
        nombre_archivo =
filedialog.askopenfilename(initialdir="c:/pythonya",
title="Seleccione archivo",
                                                   filetypes=(("todo
s los archivos", "*.*"),))
        if nombre_archivo != '':
            with open(nombre_archivo, "r", encoding="utf-8") as
archivo:
                contenido = archivo.read()
                self.scrolledtext.delete("1.0", END)
                self.scrolledtext.insert("1.0", contenido)
                self.texto = contenido

    def guardar_como(self):
        nombre_archivo =
filedialog.asksaveasfilename(initialdir="c:/pythonya", title="Guardar
como",filetypes=(("txt files", "*.txt"), ("todos los archivos",
"*.*")))
        if nombre_archivo != '':
            with open(nombre_archivo, "w", encoding="utf-8") as
archivo:
                archivo.write(self.scrolledtext.get("1.0", END))
            messagebox.showinfo("Información", "Los datos fueron
guardados en el archivo.")

    def salir(self):
        self.ventana.quit()

    def analizar(self):
        respuestas = []
```

```
        self.texto = self.scrolledtext.get("1.0", END)
        respuestas = analizador_sintactico(self.texto)
        self.scrolledtext1.delete('1.0', END)
        for respuesta in respuestas:
            self.scrolledtext1.insert(END, f'\n {respuesta}')

    def tabla_errores(self):
        path = 'TablaErrores.html'
        os.system(path)

    def tabla_tokens(self):
        path = 'TablaTokens.html'
        os.system(path)

aplicacion = interfaz()
```

Analizador léxico

1.

    - **s**: Importa la clase **Errores** del módulo **Error**. Ambos probablemente son clases relacionadas con el manejo de tokens y errores en un análisis léxico o similar.

2. **Inicialización de listas y variables globales**:

    - **inst**: Una lista que contiene palabras clave o instrucciones que podrían ser utilizadas en un lenguaje o programa específico.
    - Variables globales:
        - **n_linea**, **n_columna**, **puntero**: Variables numéricas para rastrear la posición actual en el código o texto que se está analizando.
        - **lista_lexemas**: Una lista vacía que probablemente se utilizará para almacenar los lexemas (unidades léxicas como palabras o símbolos) identificados durante el análisis.
        - **instrucciones**: Otra lista vacía que puede ser utilizada para almacenar instrucciones o comandos identificados durante el análisis.
        - **lista_errores**: Una lista vacía que se usará para almacenar los errores encontrados durante el análisis.

- **letras_digitos**, **digitos**, **letras**: Listas que contienen caracteres específicos como letras, dígitos y letras combinadas con dígitos. Estas listas se utilizan para realizar comprobaciones y filtrar caracteres durante el análisis léxico.

3. **Inicialización de las listas de caracteres**:

   - **letras_digitos**: Se inicializa con letras mayúsculas y minúsculas (**ascii_letters**), dígitos (**digits**) y el carácter de guion bajo (_).
   - **digitos**: Se inicializa solo con dígitos.
   - **letras**: Se inicializa solo con letras mayúsculas y minúsculas.

```
1
2  inst = ['CrearBD', 'EliminarBD', 'CrearColeccion', 'EliminarColeccion',
3      'InsertarUnico', 'ActualizarUnico', 'EliminarUnico', 'BuscarTodo', 'BuscarUnico', 'nueva']
4
5  global n_linea
6  global n_columna
7  global instrucciones
8  global lista_lexemas
9  global lista_errores
10 global cadena
11 global ultima_posicion
12 global letras_digitos
13 global digitos
14 global letras
15
16
17 n_linea = 1
18 n_columna = 0
19 puntero = 0
20
21 lista_lexemas = []
22 instrucciones = []
23 lista_errores = []
24 letras_digitos = []
25 digitos = []
26 letras = []
27 letras_digitos += list(string.ascii_letters)
28 letras_digitos += list(string.digits)
29 letras_digitos += ['_']
30 digitos += list(string.digits)
31 letras += list(string.ascii_letters)
```

Modificar archivo

la función **modificar_archivo** intenta abrir el archivo especificado en modo de escritura binaria y escribir el contenido nuevo en él. Si la operación es exitosa, imprime un mensaje de éxito. Si ocurre algún error al abrir o manipular el archivo, imprime un mensaje de error indicando el problema.

```python
def modificar_archivo(nombre_archivo, contenido_nuevo):
    try:
        with open(nombre_archivo, 'wb') as archivo:
            archivo.write(contenido_nuevo)
        print('Contenido modificado')
    except IOError:
        print(f'Error al abrir el archivo con el nombre {nombre_archivo}')
```

Función modificar nuevo

Esta función genera dinámicamente el contenido de una tabla HTML con información sobre tokens o errores, según el valor del parámetro **tipo**, y luego lo escribe en un archivo HTML especificado.

```python
def contenidoNew(tokens, tipo):
        titulo = ''
        if tipo == "tokens":
            nombre_archivo = "TablaTokens.html"
            titulo = "Tabla Tokens"
        else:
            nombre_archivo = "TablaErrores.html"
            titulo = "Tabla Errpres"
        contenido_tabla = '''<!DOCTYPE html>
        <html lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <title>Tabla Errores</title>
            <style>
                .centrado {
                    margin: 0 auto;
                    width: 50%;
                }
            </style>'''
```

```
        contenido_tabla += f'''</head>
        <body>
        <div class="centrado">
            <table border="1">
                <caption>{titulo}</caption>
                <thead>
                    <tr>
                        <th>Token</th>
                        <th>Linea</th>
                        <th>Columna</th>
                    </tr>
                </thead>
                <tbody>'''
        for token in tokens:
            contenido_tabla += f'''
                        <tr>
                            <td>{token.token}</td>
                            <td>{token.linea}</td>
                            <td>{token.columna}</td>
                        </tr>'''
        contenido_tabla += '''
                    </tbody>
                </table>
            </div>
        </body>
        </html>'''
        modificar_archivo(nombre_archivo, contenido_tabla)
```

Función Leer

esta función lee el siguiente carácter de una cadena de entrada, actualiza las variables de posición (**n_linea** y **n_columna**), y retorna información sobre si hay más caracteres por leer y el carácter leído en sí.

```
def leer_siguiente():
    global n_linea
    global n_columna
    global puntero
    char = entrada[puntero]
    # print(char)
    puntero += 1
    if puntero < ultima_posicion:
        resultado = True
    else:
        resultado = False
    if char == '\n':
        n_linea += 1
        n_columna = 0
    else:
        n_columna += 1
    return resultado, char
```

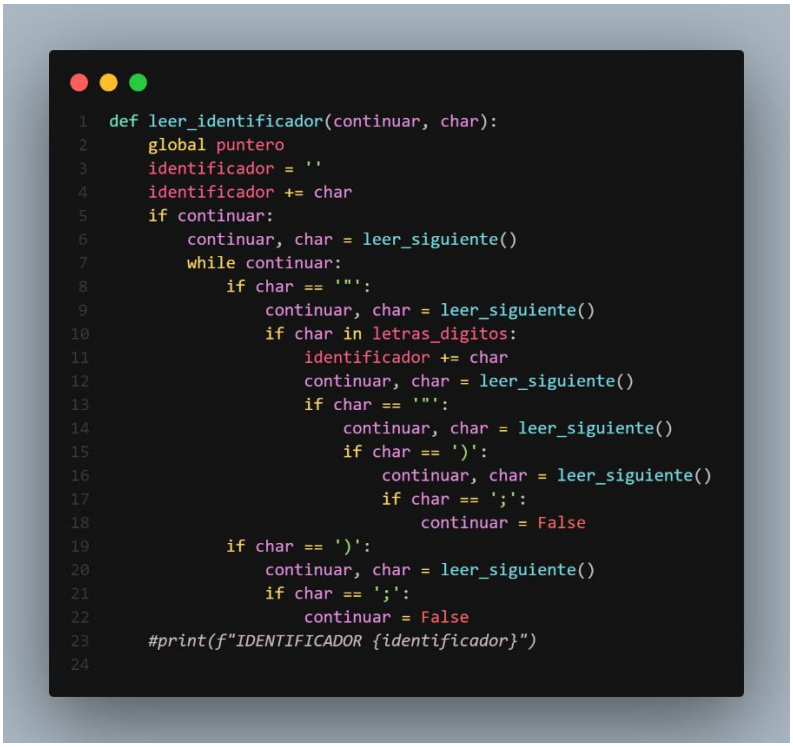
Función instrucción

la función **instruccion()** se encarga de leer caracteres de la cadena de entrada y determinar cómo procesarlos según ciertas reglas definidas en las llamadas a funciones como **leer_comentario()**, **leer_comentario_en_linea()** y **leer_funcion()**.

```python
def instruccion():
    global entrada
    global ultima_posicion
    ultima_posicion = len(entrada)
    #print(letras_digitos)
    continuar = True
    inicio_lexema = False
    #while cadena:
    while continuar:
        continuar, char = leer_siguiente()
        # PUNTO DE INGRESO DE UN COMENTARIO
        if char == '/':
            print("inicio comentario")
            continuar = leer_comentario()
        # PUNTO DE INGRESO DE UN COMENTARIO DE UNA SOLA LÍNEA
        elif char == '-':
            continuar = leer_comentario_en_linea()
        # PUNTO DE INGRESO DE PALABRAS RESERVADAS
        elif char in letras:
            continuar = leer_funcion(continuar, char)

```

Función Leer identificador



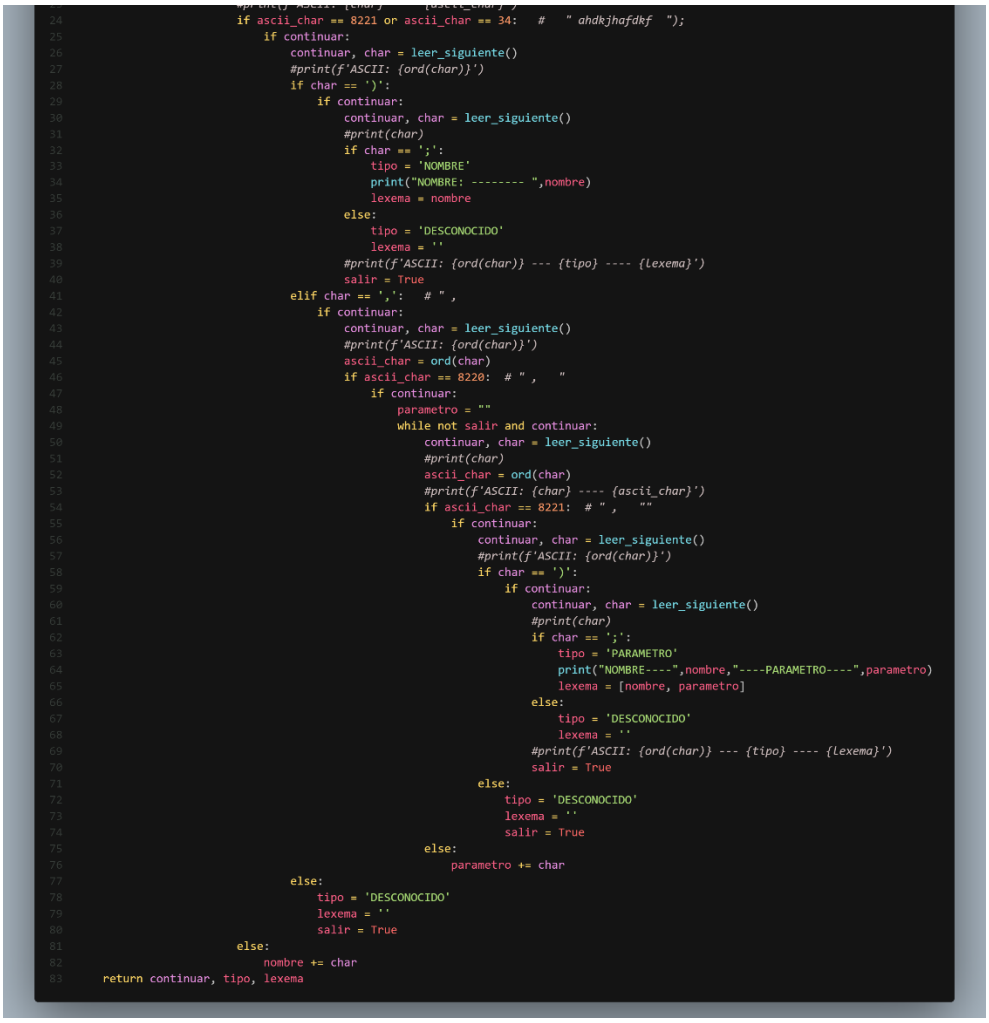
```
def leer_identificador(continuar, char):
    global puntero
    identificador = ''
    identificador += char
    if continuar:
        continuar, char = leer_siguiente()
        while continuar:
            if char == '"':
                continuar, char = leer_siguiente()
                if char in letras_digitos:
                    identificador += char
                    continuar, char = leer_siguiente()
                    if char == '"':
                        continuar, char = leer_siguiente()
                        if char == ')':
                            continuar, char = leer_siguiente()
                            if char == ';':
                                continuar = False
            if char == ')':
                continuar, char = leer_siguiente()
                if char == ';':
                    continuar = False
    #print(f"IDENTIFICADOR {identificador}")

```

esta función está diseñada para leer un identificador en la cadena de entrada, siguiendo ciertas reglas y finalizando cuando se encuentra un carácter específico o secuencia de caracteres que indica el final del identificador.

Función leer parámetros

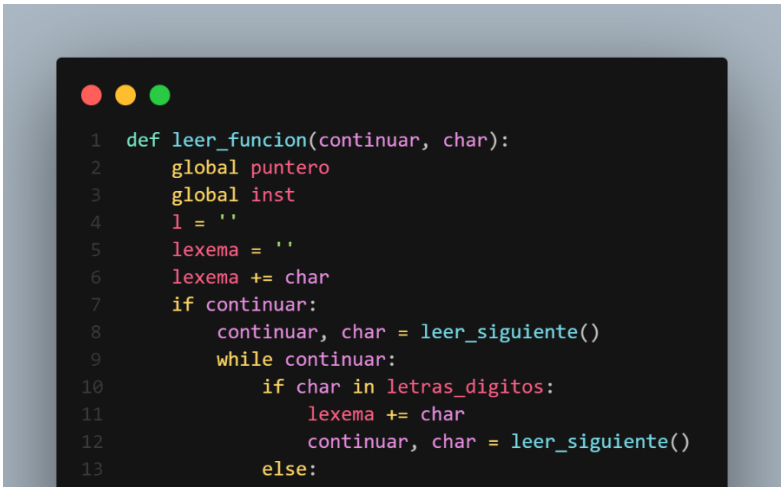
```
def leer_parametros(continuar, char):
    lexema = ''
    tipo = 'DESCONOCIDO'
    salir = False
    if continuar:
        continuar, char = leer_siguiente()
        if char == ')':
            if continuar:
                continuar, char = leer_siguiente()
                if char == ';':
                    tipo = 'CERO PARAMETROS'
                    lexema = '();'
        else:
            #print(f'ASCII: {ord(char)}')
            ascii_char = ord(char)
            if ascii_char == 8220 or ascii_char == 34:
                if continuar:
                    nombre = ""
                    while not salir and continuar:
                        continuar, char = leer_siguiente()
                        #print(char)
                        ascii_char = ord(char)
                        #print(f'ASCII: {char} ---- {ascii_char}')
```



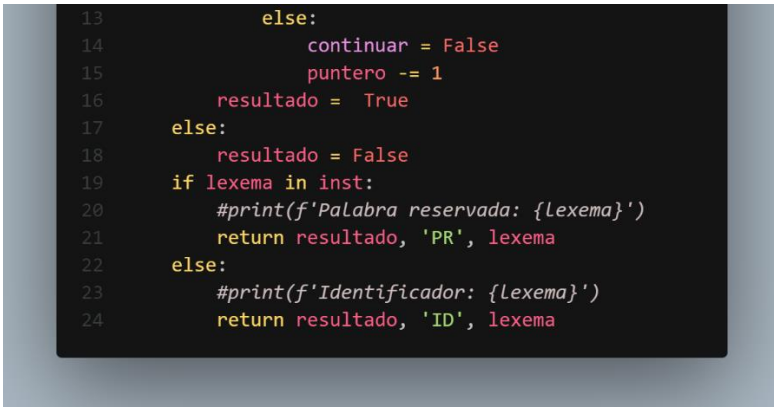
```
                    if ascii_char == 8221 or ascii_char == 34:   #   " ahdkjhafdkf  ");
                        if continuar:
                            continuar, char = leer_siguiente()
                            #print(f'ASCII: {ord(char)}')
                            if char == ')':
                                if continuar:
                                    continuar, char = leer_siguiente()
                                    #print(char)
                                    if char == ';':
                                        tipo = 'NOMBRE'
                                        print("NOMBRE: -------- ",nombre)
                                        lexema = nombre
                                    else:
                                        tipo = 'DESCONOCIDO'
                                        lexema = ''
                                    #print(f'ASCII: {ord(char)} --- {tipo} ---- {lexema}')
                                    salir = True
                            elif char == ',':   # " ,
                                if continuar:
                                    continuar, char = leer_siguiente()
                                    #print(f'ASCII: {ord(char)}')
                                    ascii_char = ord(char)
                                    if ascii_char == 8220:  # " ,   "
                                        if continuar:
                                            parametro = ""
                                            while not salir and continuar:
                                                continuar, char = leer_siguiente()
                                                #print(char)
                                                ascii_char = ord(char)
                                                #print(f'ASCII: {char} ---- {ascii_char}')
                                                if ascii_char == 8221:  # " ,   ""
                                                    if continuar:
                                                        continuar, char = leer_siguiente()
                                                        #print(f'ASCII: {ord(char)}')
                                                        if char == ')':
                                                            if continuar:
                                                                continuar, char = leer_siguiente()
                                                                #print(char)
                                                                if char == ';':
                                                                    tipo = 'PARAMETRO'
                                                                    print("NOMBRE----",nombre,"----PARAMETRO----",parametro)
                                                                    lexema = [nombre, parametro]
                                                                else:
                                                                    tipo = 'DESCONOCIDO'
                                                                    lexema = ''
                                                                #print(f'ASCII: {ord(char)} --- {tipo} ---- {lexema}')
                                                                salir = True
                                                        else:
                                                            tipo = 'DESCONOCIDO'
                                                            lexema = ''
                                                            salir = True
                                                    else:
                                                        parametro += char
                            else:
                                tipo = 'DESCONOCIDO'
                                lexema = ''
                                salir = True
                    else:
                        nombre += char
    return continuar, tipo, lexema
```

esta función está diseñada para leer y procesar parámetros de una instrucción, identificando diferentes tipos de parámetros y almacenando información sobre ellos en tipo y lexema.

Leer Función



```
def leer_funcion(continuar, char):
    global puntero
    global inst
    l = ''
    lexema = ''
    lexema += char
    if continuar:
        continuar, char = leer_siguiente()
        while continuar:
            if char in letras_digitos:
                lexema += char
                continuar, char = leer_siguiente()
            else:
```



```
            else:
                continuar = False
                puntero -= 1
        resultado =  True
    else:
        resultado = False
    if lexema in inst:
        #print(f'Palabra reservada: {lexema}')
        return resultado, 'PR', lexema
    else:
        #print(f'Identificador: {lexema}')
        return resultado, 'ID', lexema
```

esta función está diseñada para identificar y procesar palabras reservadas o identificadores en una cadena de entrada, almacenando información sobre ellos en **resultado**, **tipo** y **lexema**, y retrocediendo en el análisis si se encuentra un carácter no válido.

Leer comentario

```
def leer_comentario_en_linea():
    errores = []
    continuar, char = leer_siguiente()
    if char == '-':
        if continuar:
            continuar, char = leer_siguiente()
            if char == '-':
                while continuar:
                    continuar, char = leer_siguiente()
                    if char == '\n':
                        #print(f'comentario en linea leido en columna {n_columna} y fila {n_linea}')
                        return True
            else:
                errores.append(Errores(char, n_linea, n_columna))
    else:
        errores.append(Errores(char, n_linea, n_columna))
    contenidoNew(errores, errores)
    return False
```

esta función verifica si hay un comentario de una sola línea en la cadena de entrada, identificando el inicio (--) y el final (salto de línea), y manejando los errores si la estructura del comentario no es válida.

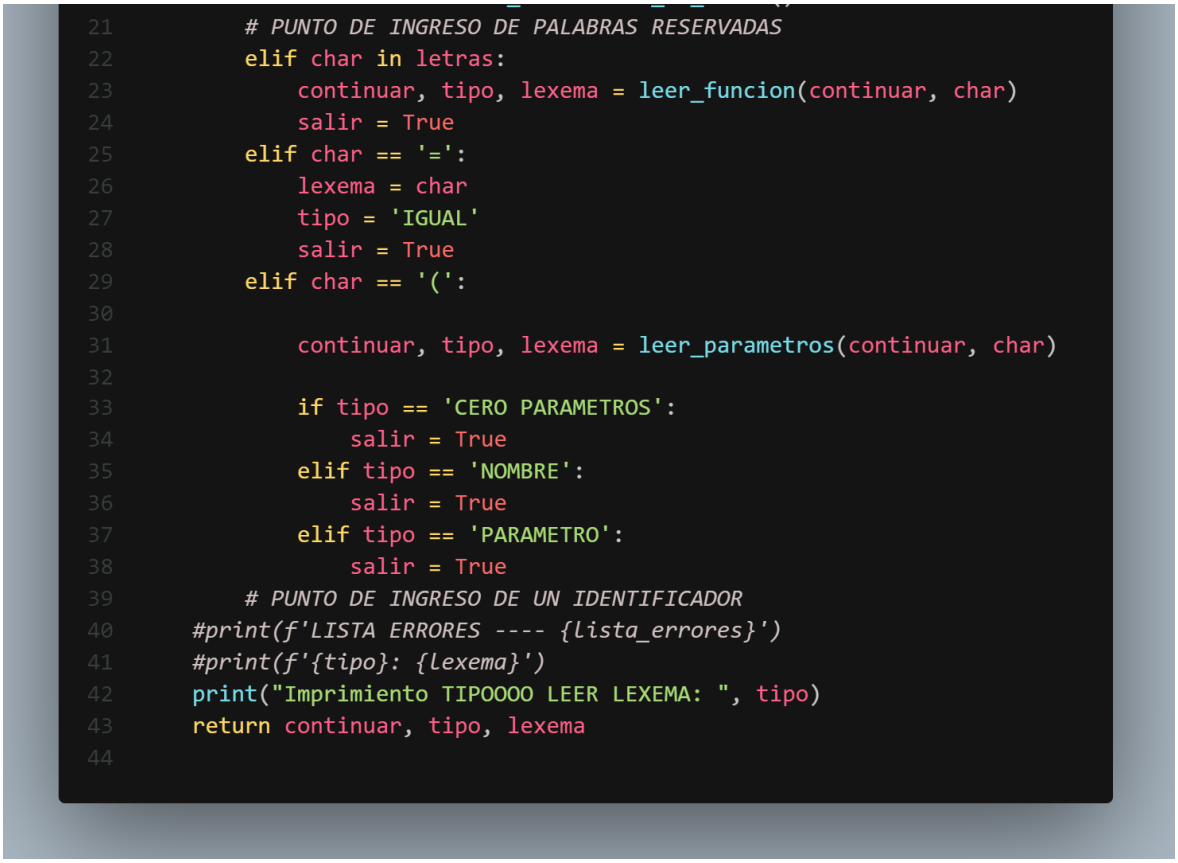
Leer Comentario

```python
def leer_comentario():
    continuar, char = leer_siguiente()
    if char == '*':
        #inicio de comentario
        while continuar:
            continuar, char = leer_siguiente()
            if char == '*':
                if continuar:
                    continuar, char = leer_siguiente()
                    if char == '/':
                        #no se almacena el comentario
                        #print(f'comentario leido en columna {n_columna} y fila {n_linea}')
                        return True
    return False
```

esta función verifica si hay un comentario multilineal en la cadena de entrada, identificando el inicio (*) y el final (*/) del comentario y retornando **True** si se ha leído correctamente el comentario multilineal, y **False** si no se ha leído correctamente.

Leer Lexema

```python
def leer_lexema():
    global entrada
    global ultima_posicion
    ultima_posicion = len(entrada)
    #print(letras_digitos)
    continuar = True
    salir = False
    inicio_lexema = False
    tipo = 'ND'
    lexema = ''
    while not salir and continuar:
        continuar, char = leer_siguiente()
        # PUNTO DE INGRESO DE UN COMENTARIO
        #print("continuar ------" ,continuar, "   char --------" ,char)
        if char == '/':
            #print("inicio comentario")
            continuar = leer_comentario()
        # PUNTO DE INGRESO DE UN COMENTARIO DE UNA SOLA LÍNEA
        elif char == '-':
            continuar = leer_comentario_en_linea()
        # PUNTO DE INGRESO DE PALABRAS RESERVADAS
```



```
        # PUNTO DE INGRESO DE PALABRAS RESERVADAS
        elif char in letras:
            continuar, tipo, lexema = leer_funcion(continuar, char)
            salir = True
        elif char == '=':
            lexema = char
            tipo = 'IGUAL'
            salir = True
        elif char == '(':

            continuar, tipo, lexema = leer_parametros(continuar, char)

            if tipo == 'CERO PARAMETROS':
                salir = True
            elif tipo == 'NOMBRE':
                salir = True
            elif tipo == 'PARAMETRO':
                salir = True
        # PUNTO DE INGRESO DE UN IDENTIFICADOR
    #print(f'LISTA ERRORES ---- {lista_errores}')
    #print(f'{tipo}: {lexema}')
    print("Imprimiento TIPOOOO LEER LEXEMA: ", tipo)
    return continuar, tipo, lexema

```

esta función es responsable de analizar y procesar los lexemas de la cadena de entrada, identificando palabras reservadas, identificadores, operadores y otros elementos importantes para el análisis léxico de un lenguaje de programación o un formato específico.

Analizador sintáctico

```
def analizador_sintactico(cadena):
    global entrada
    global n_linea
    global n_columna
    global puntero
    entrada = cadena
    instrucciones = []
    tokens = []
    n_linea = 1
    n_columna = 0
    puntero = 0
```

```
    continuar = True
    nombre = ''
    salida = ''
    while continuar:
        continuar, tipo, lexema = leer_lexema()
        #print(f'{tipo}: {lexema}')
        if continuar:
            if lexema == 'CrearBD':
                tokens.append(Tokens("CrearBD", n_linea, n_columna))
                continuar, tipo, lexema = leer_lexema()
                if tipo == 'ID':
                    nombre = lexema
                    if continuar:
                        continuar, tipo, lexema = leer_lexema()
                        if lexema == '=':
                            if continuar:
                                continuar, tipo, lexema =
leer_lexema()
                                if lexema == 'nueva':
                                    tokens.append(Tokens("nueva",
n_linea, n_columna))
                                    if continuar:
                                        continuar, tipo, lexema =
leer_lexema()
                                        if lexema == 'CrearBD':
                                            tokens.append(Tokens("Cre
arBD", n_linea, n_columna))
                                            if continuar:
                                                continuar, tipo,
lexema = leer_lexema()
                                                if tipo == 'CERO
PARAMETROS':
                                                    print(f"use('{nom
bre}');")
                                                    instrucciones.app
end(f"use('{nombre}');")
            if lexema == 'EliminarBD':
                tokens.append(Tokens("EliminarBD", n_linea,
n_columna))
                continuar, tipo, lexema = leer_lexema()
                if tipo == 'ID':
                    nombre = lexema
                    if continuar:
```

```
                        continuar, tipo, lexema = leer_lexema()
                        if lexema == '=':
                            if continuar:
                                continuar, tipo, lexema =
leer_lexema()
                                if lexema == 'nueva':
                                    tokens.append(Tokens("nueva",
n_linea, n_columna))
                                    if continuar:
                                        continuar, tipo, lexema =
leer_lexema()
                                        if lexema == 'EliminarBD':
                                            tokens.append(Tokens("Eli
minarBD", n_linea, n_columna))
                                            if continuar:
                                                continuar, tipo,
lexema = leer_lexema()
                                                if tipo == 'CERO
PARAMETROS':
                                                    print(f'db.dropDa
tabase()')
                                                    instrucciones.app
end(f'db.dropDatabase()')
            if lexema == 'CrearColeccion':
                tokens.append(Tokens("CrearColeccion", n_linea,
n_columna))
                continuar, tipo, lexema = leer_lexema()
                if tipo == 'ID':
                    nombre = lexema
                    if continuar:
                        continuar, tipo, lexema = leer_lexema()
                        if lexema == '=':
                            if continuar:
                                continuar, tipo, lexema =
leer_lexema()
                                if lexema == 'nueva':
                                    tokens.append(Tokens("nueva",
n_linea, n_columna))
                                    if continuar:
                                        continuar, tipo, lexema =
leer_lexema()
                                        if lexema ==
'CrearColeccion':
```

```
                                                tokens.append(Tokens("Cre
arColeccion", n_linea, n_columna))
                                                if continuar:
                                                    continuar, tipo,
lexema = leer_lexema()
                                                    print('PRUEBA   ',
tipo, lexema)
                                                    if tipo == 'NOMBRE':
                                                        print(f'db.create
Collection({lexema})')
                                                        instrucciones.app
end(f'db.createCollection({lexema})')
            if lexema == 'EliminarColeccion':
                tokens.append(Tokens("EliminarColeccion", n_linea,
n_columna))
                continuar, tipo, lexema = leer_lexema()
                if tipo == 'ID':
                    nombre = lexema
                    if continuar:
                        continuar, tipo, lexema = leer_lexema()
                        if lexema == '=':
                            if continuar:
                                continuar, tipo, lexema =
leer_lexema()
                                if lexema == 'nueva':
                                    tokens.append(Tokens("nueva",
n_linea, n_columna))
                                    if continuar:
                                        continuar, tipo, lexema =
leer_lexema()
                                        if lexema ==
'EliminarColeccion':
                                            tokens.append(Tokens("Eli
minarColeccion", n_linea, n_columna))
                                            if continuar:
                                                continuar, tipo,
lexema = leer_lexema()
                                                if tipo == 'NOMBRE':
                                                    print(f'db.{lexem
a}.drop()')
                                                    instrucciones.app
end(f'db.{lexema}.drop()')
            if lexema == 'InsertarUnico':
```

```
                tokens.append(Tokens("InsertarUnico", n_linea,
n_columna))
                continuar, tipo, lexema = leer_lexema()
                if tipo == 'ID':
                    nombre = lexema
                    if continuar:
                        continuar, tipo, lexema = leer_lexema()
                        if lexema == '=':
                            if continuar:
                                continuar, tipo, lexema =
leer_lexema()
                                if lexema == 'nueva':
                                    tokens.append(Tokens("EliminarCol
eccion", n_linea, n_columna))
                                    if continuar:
                                        continuar, tipo, lexema =
leer_lexema()
                                        if lexema == 'InsertarUnico':
                                            tokens.append(Tokens("Ins
ertarUnico", n_linea, n_columna))
                                            if continuar:
                                                continuar, tipo,
lexema = leer_lexema()
                                                print("TIPO DE
RETORNO InsertarUnico ---- ",tipo)
                                                if tipo ==
'PARAMETRO':
                                                    print(f'db.{lexem
a[0]}.insertOne({lexema[1]})')
                                                    instrucciones.app
end(f'db.{lexema[0]}.insertOne({lexema[1]})')
            if lexema == 'ActualizarUnico':
                tokens.append(Tokens("ActualizarUnico", n_linea,
n_columna))
                continuar, tipo, lexema = leer_lexema()
                if tipo == 'ID':
                    nombre = lexema
                    if continuar:
                        continuar, tipo, lexema = leer_lexema()
                        if lexema == '=':
                            if continuar:
                                continuar, tipo, lexema =
leer_lexema()
```

```
                                if lexema == 'nueva':
                                    tokens.append(Tokens("nueva",
n_linea, n_columna))
                                    if continuar:
                                        continuar, tipo, lexema =
leer_lexema()
                                        if lexema ==
'ActualizarUnico':
                                            tokens.append(Tokens("Act
ualizarUnico", n_linea, n_columna))
                                            if continuar:
                                                continuar, tipo,
lexema = leer_lexema()
                                                print("TIPO DE
RETORNO ActualizarUnico ---- ",tipo)
                                                if tipo ==
'PARAMETRO':
                                                    print(f'db.{lexem
a[0]}.updateOne({lexema[1]})')
                                                    instrucciones.app
end(f'db.{lexema[0]}.updateOne({lexema[1]})')
            if lexema == 'EliminarUnico':
                tokens.append(Tokens("EliminarUnico", n_linea,
n_columna))
                continuar, tipo, lexema = leer_lexema()
                if tipo == 'ID':
                    nombre = lexema
                    if continuar:
                        continuar, tipo, lexema = leer_lexema()
                        if lexema == '=':
                            if continuar:
                                continuar, tipo, lexema =
leer_lexema()
                                if lexema == 'nueva':
                                    tokens.append(Tokens("nueva",
n_linea, n_columna))
                                    if continuar:
                                        continuar, tipo, lexema =
leer_lexema()
                                        if lexema == 'EliminarUnico':
                                            tokens.append(Tokens("Eli
minarUnico", n_linea, n_columna))
                                            if continuar:
```

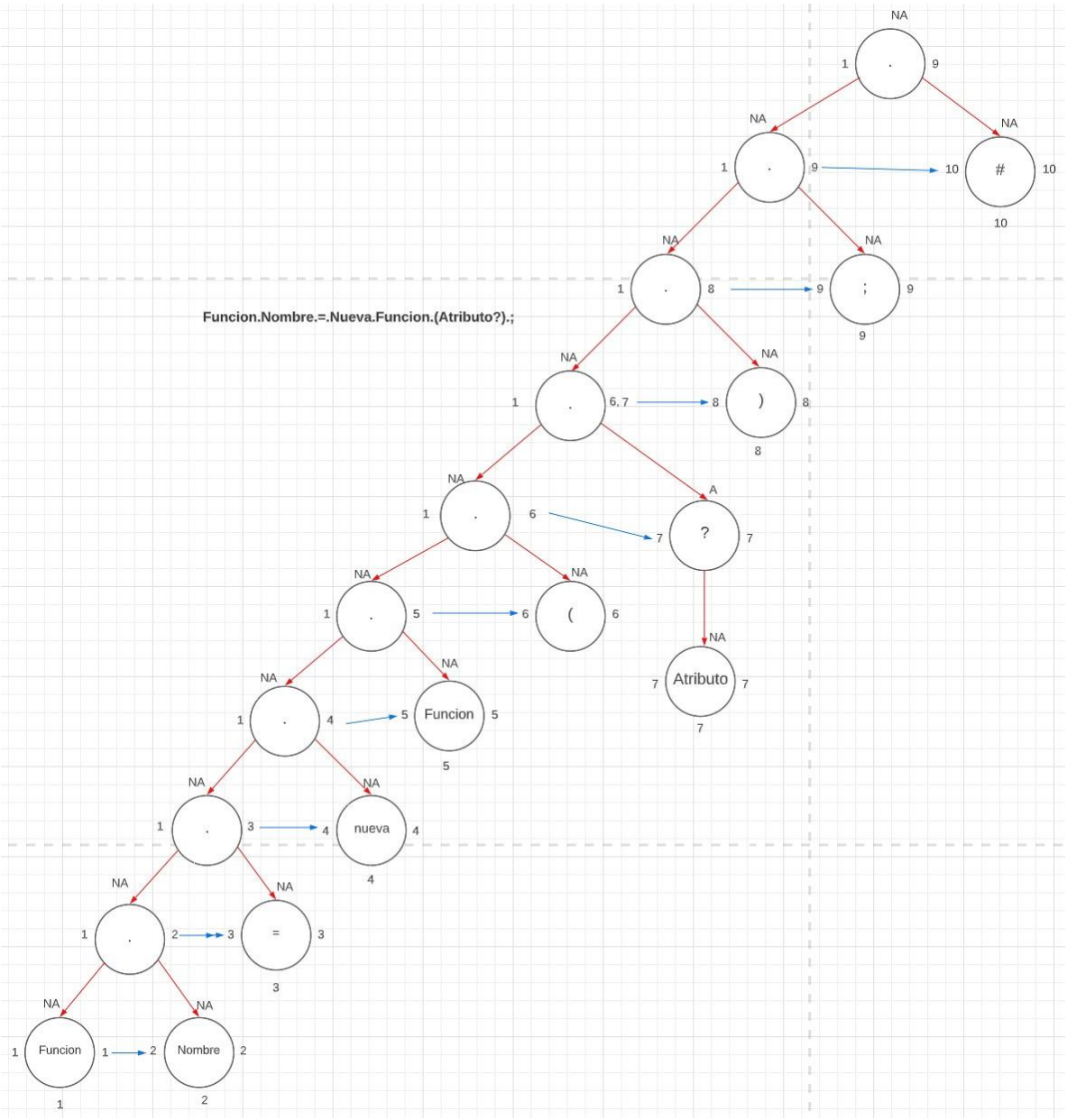
```
                                                    continuar, tipo,
lexema = leer_lexema()
                                                    if tipo != 'CERO
PARAMETROS':
                                                        print(f'db.{lexem
a[0]}.deleteOne({lexema[1]})')
                                                        instrucciones.app
end(f'db.{lexema[0]}.deleteOne({lexema[1]})')
            if lexema == 'BuscarTodo':
                tokens.append(Tokens("BuscarTodo", n_linea,
n_columna))
                continuar, tipo, lexema = leer_lexema()
                if tipo == 'ID':
                    nombre = lexema
                    if continuar:
                        continuar, tipo, lexema = leer_lexema()
                        if lexema == '=':
                            if continuar:
                                continuar, tipo, lexema =
leer_lexema()
                                if lexema == 'nueva':
                                    tokens.append(Tokens("nueva",
n_linea, n_columna))
                                    if continuar:
                                        continuar, tipo, lexema =
leer_lexema()
                                        if lexema == 'BuscarTodo':
                                            tokens.append(Tokens("Bus
carTodo", n_linea, n_columna))
                                            if continuar:
                                                continuar, tipo,
lexema = leer_lexema()
                                                if tipo != 'CERO
PARAMETROS':
                                                    print(f'db.{nombr
e}.find()')
                                                    instrucciones.app
end(f'db.{nombre}.find()')
            if lexema == 'BuscarUnico':
                tokens.append(Tokens("BuscarUnico", n_linea,
n_columna))
                continuar, tipo, lexema = leer_lexema()
                if tipo == 'ID':
```

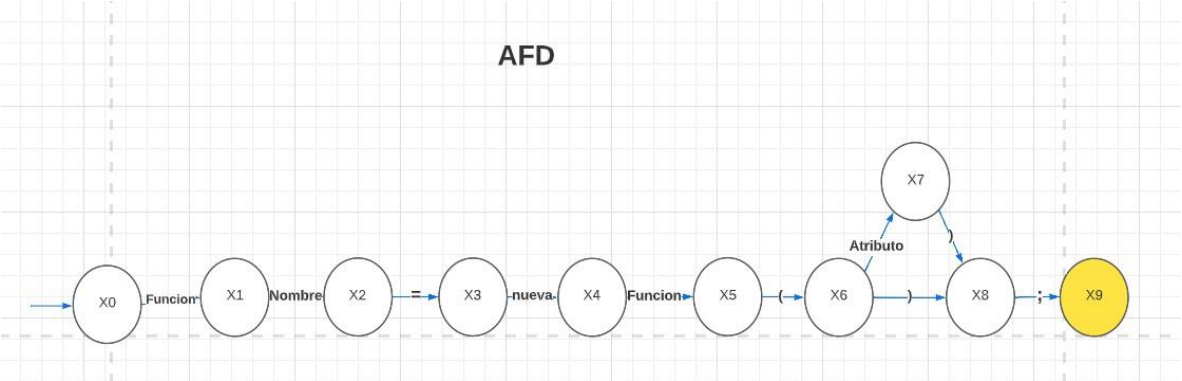
```
                    nombre = lexema
                    if continuar:
                        continuar, tipo, lexema = leer_lexema()
                        if lexema == '=':
                            if continuar:
                                continuar, tipo, lexema =
leer_lexema()
                                if lexema == 'nueva':
                                    tokens.append(Tokens("nueva",
n_linea, n_columna))
                                    if continuar:
                                        continuar, tipo, lexema =
leer_lexema()
                                        if lexema == 'BuscarUnico':
                                            tokens.append(Tokens("Bus
carUnico", n_linea, n_columna))
                                            if continuar:
                                                continuar, tipo,
lexema = leer_lexema()
                                                if tipo != 'CERO
PARAMETROS':
                                                    print(f'db.{nombr
e}.findOne()')
                                                    instrucciones.app
end(f"db.{nombre}.findOne()")
    contenidoNew(tokens, "tokens")

    return instrucciones
```

este analizador sintáctico se encarga de identificar y analizar las estructuras y operaciones de un lenguaje específico relacionado con bases de datos, generando instrucciones que podrían ser utilizadas para interactuar con una base de datos en un sistema o aplicación.

Método del árbol





TABLA

Estado	Funcion	Nombre	=	Nueva	(	Atributo	)	;	#
X0 = {1}	X1	*	*	*	*	*	*	*	
X1 = {2}	*	X2	*	*	*	*	*	*	
X2 = {3}	*	*	X3	*	*	*	*	*	
X3 = {4}	*	*	*	X4	*	*	*	*	
X4 = {5}	X5	*	*	*	*	*	*	*	
X5 = {6}	*	*	*	*	X6	*	*	*	
X6 = {7, 8}	*	*	*	*	*	X7	X8	*	
X7 = {8}	*	*	*	*	*	*	X8	*	
X8 = {9}	*	*	*	*	*	*	*	X9	
X9 = {10}									X10

X9 estado de aceptación

Hoja	Σ	Follow Pos
1	Funcion	2
2	Nombre	3
3	=	4
4	Nueva	5
5	Funcion	6
6	(	7, 8
7	Atributo	8
8	)	9
9	;	10
10	#	

GRAMATICA

G = {Terminales, No Terminales, Inicio, Producciones}

Terminales = {Función, Nombre, =, nueva, atributo, (,), ; }

No Terminales = {X0

Inicio = X0

Producciones:

X0 → Función X1

X1 →Nombre X2

X2 →= X3

X3 →nueva X4

X4 →Función X5

X5 → (X6

X6 →Atributo X7 |) X8

X7 →) X8

X8 →; X9

X9 →épsilon