MANUAL TÉCNICO

Traductor de archivo tipo JSON a HTML

04 DE ABRIL DE 2024

ANGEL ENRIQUE ALVARADO RUIZ - 202209714

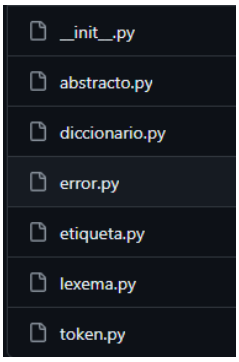
Práctica Única de Lenguajes Formales y de Programación

Contenido

Paradigma utilizado 2
Paquete Elementos 2
Abstracto 2
Lexema, Errores y Tokens 2
Clase Etiqueta 3
Archivos Diccionarios 5
Analizador Léxico 6
Interfaz 15

Paradigma utilizado

El paradigma utilizado durante la practica única fue el paradigma orientado a objetos POO.



Paquete Elementos

En la carpeta elementos podemos ver 7 archivos. El primero y más importante, que sirve para que Python reconozca como un paquete dicha carpeta, es "__init__"; este contiene "enlaces" hacia los otros archivos de la carpeta, para que pueda accederse a ellos desde fuera del paquete.

Abstracto

Es la clase base desde la cual hacemos un esbozo de las clases lexema, token y error, ya que, como tal, las 3 clases van a heredar desde esta clase llamada "Expression".

```python
from abc import ABC, abstractmethod

class Expression(ABC):

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

    @abstractmethod
    def execute(self, enviroment):
        pass

    @abstractmethod
    def getFila(self) -> int:
        return self.fila

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

Lexema, Errores y Tokens

La clase "Lexema", se encarga de compactar en un objeto el lexema encontrado, su número de fila y columna. La clase "Errores" tiene un funcionamiento similar, a excepción del método execute ya que en error crea una fila de una tabla con toda la información relacionada al error encontrado, la clase "Token" funciona igual que error, pero agregando el atributo "id" a su constructor y estructura en el método execute.

```python
from elementos.abstracto import Expression

class Lexema(Expression):
    def __init__(self, lexema, fila, columna):
        self.lexema = lexema
        super().__init__(fila, columna)

    def execute(self, enviroment):
        return self.lexema

    def getFila(self):
        return super().getFila()

    def getColumna(self):
        return super().getColumna()
```

```python
from elementos.abstracto import Expression

class Errores(Expression):

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

    def execute(self, num):
        numero = f'<td align="center">{num}</td>\n'
        lexema = f'<td align="center">{self.caracter}</td>\n'
        fila = f'<td align="center">{self.fila}</td>\n'
        columna = f'<td align="center">{self.columna}</td>\n'
        return f'{numero + lexema + columna + fila}'

    def getFila(self):
        return super().getFila()

    def getColumna(self):
        return super().getColumna()
```

```python
from elementos.abstracto import Expression

class Token(Expression):
    def __init__(self, id, lexema, fila, columna):
        self.id = id
        self.lexema = lexema
        super().__init__(fila, columna)

    def execute(self, enviroment=None):
        id = f'<td align="center">{self.id}</td>\n'
        lexema = f'<td align="center">{self.lexema}</td>\n'
        fila = f'<td align="center">{self.fila}</td>\n'
        columna = f'<td align="center">{self.columna}</td>\n'
        return f'{id + lexema + columna + fila}'

    def getFila(self):
        return super().getFila()

    def getColumna(self):
        return super().getColumna()
```

Clase Etiqueta

Es la clase que realiza el trabajo de traducción directamente, ya que el analizador léxico fue orientado a la extracción de datos y su "limpieza" para que luego instanciarse la información de las etiquetas en esta clase. La clase analiza la instancia en sí, su nombre y su diccionario de atributos. Es un código algo largo, ya que literalmente traduce toda la información, palabras reservadas y valores del diccionario, en etiquetas y atributos HTML.

```
from elementos.diccionario import traduccion
class Etiqueta():
        def __init__(self, nombre):
                self.nombre = nombre    # el nombre por el que voy a identificar mi diccionario
                self.atributos = {}       # el diccionario que voy a llenar con los valores que me lleguen

        def crearEtiquetahtml(self) -> str:
                etiqueta = ""
                contenido = ""
                if self.nombre == "Titulo":
                        etiqueta = "<ht "
                        for i in self.atributos:
                                if i == "tamaÃ±o":
                                        etiquetafinal = etiqueta[3:]
                                        etiqueta = f'<h{self.atributos.get(i,None)[1]}' + etiquetafinal
                                elif i == "texto":
                                        contenido = self.atributos[i]
                                else:
                                        valor = traduccion.get(self.atributos[i].upper(),None)
                                        atr = traduccion.get(i.upper(),None)
                                        if valor != None:
                                                etiqueta += atr + '="' + valor + '" '
                                        else:
                                                etiqueta += atr + '="' + self.atributos[i] + '" '
                        etiqueta += ">" + contenido + "</" + etiqueta[1:3] + ">"
                elif self.nombre == "Negrita":
                        etiqueta = "<b>"
                        contenido = self.atributos["texto"]
                        etiqueta += contenido + "</b>"
                elif self.nombre == "Cursiva":
                        etiqueta = "<u>"
                        contenido = self.atributos["texto"]
                        etiqueta += contenido + "</u>"
                elif self.nombre == "Subrayado":
                        etiqueta = "<u>"
                        contenido = self.atributos["texto"]
                        etiqueta += contenido + "</u>"
                elif self.nombre == "Tachado":
                        etiqueta = "<del>"
                        contenido = self.atributos["texto"]
                        etiqueta += contenido + "</del>"
```

```
elif self.nombre == "Codigo":
        etiqueta = f'<code class="codigo" '
        for i in self.atributos:
                if i == "texto":
                        contenido = self.atributos[i]
                else:
                        valor = traduccion.get(self.atributos[i].upper(),None)
                        atr = traduccion.get(i.upper(),None)
                        if valor != None:
                                etiqueta += atr + '="' + valor + '" '
                        else:
                                etiqueta += atr + '="' + self.atributos[i] + '" '
        etiqueta += ">" + '<font color="white" face="Hurmit Nerd font light">' + contenido + "</font>" +"</code>"
elif self.nombre == "Texto":
        etiqueta = "<" + traduccion.get(self.nombre.upper(),None) + "> "
        contador = 0
        for atributo in self.atributos:
                if contador == 0:
                        etiqueta += "<font" + ' '
                        contador += 1

                if atributo == "texto":
                        contenido = self.atributos[atributo]
                elif atributo == "fuente":
                        etiqueta += 'face'+ '="' + self.atributos[atributo] + '" '
                elif atributo == "color":
                        etiqueta += 'color'+ '="' + self.atributos[atributo] + '" '
                else:
                        lista = list(self.atributos.keys())
                        value = traduccion.get(self.atributos[atributo].upper(),None)
                        if value != None:
                                etiqueta +=  atributo + '="' + value + '" '
                        else:
                                atr = traduccion.get(atributo.upper(),None)
                                etiqueta += atr + '="' + self.atributos[atributo] + '" '
        etiqueta += ">" + contenido + "</font></p>"
else:
       etiqueta = traduccion.get(self.nombre.upper(),None)
       if etiqueta != "table":
                    etiqueta = "<" + etiqueta + " "
                    if self.nombre == "Fondo":
                            return f'background-color: {self.atributos["color"]};'
                    else:
                            for atributo in self.atributos:
                                    if atributo == "texto":
                                            contenido = self.atributos[atributo]
                                    else:
                                            lista = list(self.atributos.keys())
                                            value = traduccion.get(self.atributos[atributo].upper(),None)
                                            if value != None:
                                                    etiqueta +=  atributo + '="' + value + '" '
                                            else:
                                                    atr = traduccion.get(atributo.upper(),None)
                                                    etiqueta += atr + '="' + self.atributos[atributo] + '" '
                                            if atributo == lista[-1]:
                                                    etiqueta += ">" + contenido + "</" + traduccion.get(self.nombre.upper(),None) + ">"
                                                    break
       else:
              # Leer el número de filas y columnas
              num_filas = int(self.atributos['filas'])
              num_columnas = int(self.atributos['columnas'])

              # Iniciar el código HTML de la tabla
              tabla_html = '<table align="center" border="black" height="300" width="85%">\n'
```

```
                # Iniciar el código HTML de la tabla
                tabla_html = '<table align="center" border="black" height="300" width="85%">\n'

                # Recorrer las filas y columnas para construir la tabla
                for i in range(num_filas):
                        tabla_html += '<tr>\n'
                        for j in range(num_columnas):
                                encontrado = False
                                for k in range(num_filas * num_columnas):
                                        clave_elemento = f'elemento{k}'
                                        celda = self.atributos.get(clave_elemento, {})
                                        fila_celda = int(celda.get('fila', '0')) - 1
                                        columna_celda = int(celda.get('columna', '0')) - 1
                                        if fila_celda == i and columna_celda == j:
                                                contenido = celda.get('contenido', '')
                                                tabla_html += f'<td>{contenido}</td>\n'
                                                encontrado = True
                                                break
                                if not encontrado:
                                        tabla_html += '<td></td>\n'
                        tabla_html += '</tr>\n'

                tabla_html += '</table>'
                etiqueta = tabla_html
    return etiqueta
```

Archivos Diccionarios

Este archivo almacena mis dos diccionarios principales que son el de palabras reservadas y el de traducción.

```
traduccion = {
    'TITULO'            :   'h',
    'FONDO'             :   'body',
    'PARRAFO'           :   'p',
    'TEXTO'             :   'p',
    'CODIGO'            :   'code',
    'SALTO'             :   'br',
    'TABLA'             :   'table',
    'NEGRITA'           :   'br',
    'CURSIVA'           :   'em',
    'TACHADO'           :   'del',
    'SUBRAYADO'         :   'u',
    # elementos internos
    'POSICION'          :   'align',
    'TAMAÃ±O'           :   'size',
    'COLOR'             :   'color',
    'FUENTE'            :   'font',
    'IZQUIERDA'         :   'left',
    'CENTRO'            :   'center',
    'DERECHA'           :   'right',
    'AZUL'          :   'blue',
    'ROJO'          :   'red',
    'AMARILLO'      :   'yellow',
    # elementos de tabla
    'FILA'              :   'row',
    'COLUMNA'           :   'column',
    'ELEMENTO'          :   'elemento',
    'CANTIDAD'          :   'cantidad',
    'FILAS'             :   'filas',
    'COLUMNAS'          :   'columnas',
}
```

```
reservadas = {
    # elementos externos
    'INICIO'            :   'Inicio',
    'ENCABEZADO'        :   'Encabezado',
    'CUERPO'            :   'Cuerpo',
    # Elementos medios
    'TITULOPAGINA'      :   'TituloPagina',
    'TITULO'            :   'Titulo',
    'FONDO'             :   'Fondo',
    'PARRAFO'           :   'Parrafo',
    'TEXTO'             :   'Texto',
    'CODIGO'            :   'Codigo',
    'SALTO'             :   'Salto',
    'TABLA'             :   'Tabla',
    'NEGRITA'           :   'Negrita',
    'CURSIVA'           :   'Cursiva',
    'TACHADO'           :   'Tachado',
    'SUBRAYADO'         :   'Subrayado',
    # elementos internos
    'texto'             :   'texto',
    'POSICION'          :   'posicion',
    'TAMAÃ±O'           :   'tamaÃ±o',
    'COLOR'             :   'color',
    'FUENTE'            :   'fuente',
    'FILAS'             :   'filas',
    'COLUMNAS'          :   'columnas',
    'FILA'              :   'fila',
    'COLUMNA'           :   'columna',
    'ELEMENTO'          :   'elemento',
    'CANTIDAD'          :   'cantidad',
    # signos
    '"'                 :   '"',
    ';'                 :   ';',
    ':'                 :   ':',
    '{'                 :   '{',
    '}'                 :   '}',
    '['                 :   '[',
    ']'                 :   ']',
    ','                 :   ','
}
```

Analizador Léxico

```python
# import para denotar que una función tiene diferentes tipos de retornos
from typing import List, Union, Tuple

# importando clases principales
from elementos.error import Errores
from elementos.lexema import Lexema
from elementos.token import Token
from elementos.etiqueta import Etiqueta

# importando diccionario de palabras clave y de traduccion
from elementos.diccionario import *

class Analizador():

    def __init__(self):
        # variables globales
        self.nLinea = 0
        self.nColumna = 0
        self.listadoLexemas = []
        self.listadoLexemas_sinRepetir = []
        self.listadoErrores = []
        self.listadoTokens = []
        self.etiquetas = []
        self.html = ""
```

Inicialmente comenzamos realizando las importaciones de los paquetes; de la clase "Errores", "Lexema", "Token" y "Etiqueta", además de realizar el constructor de la clase "Analizador" e importar el archivo de diccionarios.

adicionalmente se agrega otra importación diferente del proyecto que se llama "typing" que lo único que hace es permitirme denotar que tipo de variables me puede retornar un método o función en Python, por ejemplo:

Es solamente una forma de mantener el orden y tener claridad al momento de tener más de un tipo de retorno en una función, además es muy útil cuando el código resulta bastante largo como en este proyecto.

```python
from typing import Union

# tira un error porque no es posible denotar normalmente en python dos tipos de retornos o más
def convertir_a_string(parametro) -> str, bool:
    if type(parametro) != str:
        return str(parametro), True
    else:
        return f'El parametro "{parametro}" es un string', False

# alternativa por la que se usa typing
def convertir_a_string(parametro) -> Union[str, bool]:
    if type(parametro) != str:
        return [str(parametro), True]
    else:
        return [f'El parametro "{parametro}" es un string', False]
```

```python
def armar_lexema(self, cadena, analisis=1) -> tuple:
    lexema = ''
    puntero = ''
    if analisis == 1:
        for caracter in cadena:
            puntero += caracter
            if caracter == '\"':
                return lexema, cadena[len(puntero):]
            elif caracter == ':' or caracter == '=':
                return lexema, cadena[len(puntero)-1:]
            else:
                lexema += caracter
    else:
        for caracter in cadena:
            puntero += caracter
            if caracter == '\"':
                return lexema, cadena[len(puntero)-1:]
            else:
                lexema += caracter
    return None, None
```

El código del analizador Léxico es el más largo, pero luego sus funciones se ejecutan de la siguiente forma:

Se conforma de las funciones: "Analizar_entrada" y "armarLexema". A grandes rasgos, la primera función es el analizador léxico que va llamando a la función armar lexema que se utilizar para ocasiones en las que el lexema tiene una estructura definida, por ejemplo, los valores de los atributos vienen entre comillas dobles, por lo que la función armar lexema se llama y comienza a armar un lexema que va desde la primer comilla, agarrando todos los caracteres intermedios sin importarle ninguno, hasta la comilla que cierra.

```
def analizar_entrada(self,cadena):
    lexema = ""
    puntero = 0

    while cadena:
        caracter = cadena[puntero]
        puntero += 1

        if caracter == '\"':
            l = Lexema(caracter, self.nLinea, self.nColumna)
            for i in self.listadoLexemas_sinRepetir:
                if l.lexema == i.lexema:
                    break
                if i == self.listadoLexemas_sinRepetir[-1]:
                    if l.lexema != i.lexema:
                        self.listadoLexemas_sinRepetir.append(l)
            lexema, cadena = self.armar_lexema(cadena[puntero:])
            if lexema and cadena:
                lex = Lexema(f'"{lexema}"', self.nLinea, self.nColumna)
                self.listadoLexemas.append(lex)
                if self.existe(lex) == False:
                    self.listadoLexemas_sinRepetir.append(lex)
                self.nColumna += len(lexema) + 1
                puntero = 0
        elif caracter.isupper() or caracter.islower():
            lexema, cadena = self.armar_lexema(cadena[puntero-1:])
            if lexema and cadena:
                lex = Lexema(lexema, self.nLinea, self.nColumna)
                self.listadoLexemas.append(lex)
                if self.existe(lex) == False:
                    self.listadoLexemas_sinRepetir.append(lex)
                self.nColumna += len(lexema) + 1
                puntero = 0
        elif caracter == '{' or caracter == '}' or caracter == '[' or caracter == ']' or caracter == ';' or caracter == ',' or caracter == ':' or caracter == '=':
            lex = Lexema(caracter, self.nLinea, self.nColumna)
            self.listadoLexemas.append(lex)
            if self.existe(lex) == False:
                self.listadoLexemas_sinRepetir.append(lex)
            cadena = cadena[1:]
            puntero = 0
            self.nColumna += 1
        elif caracter =="\t":
            self.nColumna += 4
            cadena = cadena[4:]
            puntero = 0
        elif caracter == "\n":
            cadena = cadena[1:]
            puntero = 0
            self.nLinea += 1
            self.nColumna = 1
        elif caracter == ' ' or caracter == '\r':
            self.nColumna += 1
            cadena = cadena[1:]
            puntero = 0
        else:
            error = Errores(caracter, self.nLinea, self.nColumna)
            self.listadoErrores.append(error)
            self.nColumna += 1
            cadena = cadena[1:]
            puntero = 0
```

Luego analizamos Errores; primero verificamos que no hayan errores léxicos, en caso de ser así vamos a exportar los errores con la función self.exportarErrores(), en la cual vamos a utilizar la función getErrores() que nos dará un formato para exportar los errores en documento tipo html.

En caso de no ser así vamos a verificar que no haya errores en que denoten que la etiqueta inicio venga dos veces o no venga, así como el cuerpo y el encabezado. Para esto nos ayudamos de las funciones getErrores2(), cuyo objetivo es verificar las repitencias de las etiquetas INICIO, ENCABEZADO Y CUERPO.

Después vemos que hay un ciclo for, que cuenta la cantidad de llaves de apertura y cerradura que hay, así como los corchetes, esto porque así vamos a poder detectar si falta un corchete o una llave, la cantidad de corchetes debe ser igual en la apertura y cerradura, de la misma forma en las llaves. En caso de no cumplirse alguna de estas condiciones para errores, entonces podemos seguir la ejecución en caso contrario vamos a retornar un string o una lista, para saber que hubo un error y parar la ejecución.

```python
def analizarErrores(self) -> Union[List, str]:
    erroresLexicos = len(self.listadoErrores)
    if erroresLexicos > 0:
        self.exportarErrores()
        return "errores generados en archivo ListaErrores.html"

    # verificando etiqueta encabezado y cuerpo
    atributo, problema = self.getErrores2()
    lista = [atributo, problema]
    if atributo != None:
        return lista

    la, lc, ca, cc = 0, 0, 0, 0
    listadoSignos = ""
    for signo in self.listadoLexemas:
        if signo.lexema == "{":
            la += 1
            listadoSignos += f'{signo.lexema} , {signo.getFila()} , {signo.getColumna()}'+'\n'
        elif signo.lexema == "[":
            ca += 1
            listadoSignos += f'{signo.lexema} , {signo.getFila()} , {signo.getColumna()}'+'\n'
        elif signo.lexema == "}":
            lc += 1
            listadoSignos += f'{signo.lexema} , {signo.getFila()} , {signo.getColumna()}'+'\n'
        elif signo.lexema == "]":
            cc += 1
            listadoSignos += f'{signo.lexema} , {signo.getFila()} , {signo.getColumna()}'+'\n'
        else:
            continue
    llaves = la == lc
    corchetes = ca == cc
    if llaves == True and corchetes == True:
        return None
    else:
        error = f' el simbolo {"{"+" "}aparece {la} veces y el simbolo {"}"+" "} aparece {lc} veces'+'\n'
        error += f' el simbolo "[" aparece {ca} veces y el simbolo "]" aparece {cc} veces'+'\n'*2
        error += "Listado de ocurrencias de llaves y corchetes: \n"
        error += listadoSignos
        return ["error en la cantidad de llaves y corchetes", error]
```

```
def getErrores(self) -> str:
    self.listadoErrores
    formato = '<table align="center" border="black" height="50%" width="50%">\n'
    formato += '<tr bgcolor="black">\n'
    formato += '<th><font color="white">No.</font</th>\n'
    formato += '<th><font color="white">Lexema</font</th>\n'
    formato += '<th><font color="white">Fila</font</th>\n'
    formato += '<th><font color="white">Columna</font</th>\n'
    formato += '</tr>\n'
    for i in range(len(self.listadoErrores)):
        formato += '<tr>\n'
        error = self.listadoErrores[i]
        formato += error.execute(i+1) + '</tr>\n'
    formato += '</table>\n'
    return formato
```

```
def getErrores2(self) -> list:
    c1, c2, c3 = 0, 0, 0
    for lexema in self.listadoLexemas:
        if lexema.lexema == "Encabezado":
            c1 += 1
        elif lexema.lexema == "Cuerpo":
            c2 += 1
        elif lexema.lexema == "Inicio":
            c3 += 1
        else:
            continue
    if c3 == 0 or c3 > 1:
        return ["Inicio", c3]
    elif c1 == 0 or c1 > 1:
        return ["Encabezado", c1]
    elif c2 == 0 or c2 > 1:
        return ["Cuerpo", c2]
    return None, None
```

```
def exportarErrores(self):
    nombre = "ListaErrores"+".html"
    with open(nombre, 'w') as archivo:
        archivo.write('<!DOCTYPE html>\n')
        archivo.write('<html>\n')
        archivo.write('<head>\n')
        archivo.write('<title>Errores</title>\n')
        archivo.write('</head>\n')
        archivo.write('<body font-size="16">\n')
        archivo.write('<font size="6">\n')
        archivo.write(self.getErrores())
        archivo.write('</font>\n')
        archivo.write('</body>\n')
        archivo.write('</html>\n')
```

La función de apoyo “Existe”, solamente me sirve para verificar si un lexema ya se agregó a un listado reservado para lexemas sin repetir (servirá para crear tokens), esta función se utiliza dentro de analizar_entrada() porque desde ahí comienza a guardar los lexemas.

```
# funciones de apoyo
def existe(self, lexema) -> bool:
    listado = self.listadoLexemas_sinRepetir
    for i in listado:
        if i.lexema == lexema.lexema:
            return True
    return False
```

Luego de realizar la depuración de errores, podemos seguir con la creación de tokens, es importante mencionar que la función llamada de esta forma, solamente se basa en el listado de lexemas sin repetir inicializado en el constructor de clase, por lo que solamente hay que convertir a objetos tipo tokens la información relacionada, además es aquí donde usamos el diccionario de palabras reservadas, para saber si es una palabra reservada, un símbolo o una cadena cualquiera, dichas cadenas serán enumeradas, para saber cuántas son.

```
def crearTokens(self, cadenas=True, listado=[]):
    # primero analizamos las palabras reservadas y separamos de strings y numeros
    global reservadas
    contadorCadenas = 1
    if cadenas:
        for lexema in self.listadoLexemas_sinRepetir:
            if len(lexema.lexema) == 1:
                if lexema.lexema == reservadas.get(lexema.lexema,None):
                    self.listadoTokens.append(
                        Token("Simbolo", lexema.lexema, lexema.getFila(), lexema.getColumna())
                    )
                continue
            clave = ""
            if lexema.lexema == "texto":
                clave = lexema.lexema.lower()
            else:
                clave = lexema.lexema.upper()
            valor = reservadas.get(clave, None)
            if lexema.lexema == valor and valor != None:
                self.listadoTokens.append(
                    Token("Palabra Reservada", lexema.lexema, lexema.getFila(), lexema.getColumna())
                )
            elif lexema.lexema.isdigit():
                self.listadoTokens.append(
                    Token("Numero", lexema.lexema, lexema.getFila(), lexema.getColumna())
                )
            else:
                self.listadoTokens.append(
                    Token(f"Cadena no.{contadorCadenas}", lexema.lexema, lexema.getFila(), lexema.getColumna())
                )
                contadorCadenas += 1
    else:
        contadorFilas = 0
        contadorColumnas = 0
        for lexema in listado:
            clave = ""
            if lexema == "fila":
                contadorFilas += 1
                clave = lexema.upper()
            elif lexema == "columna":
                contadorColumnas += 1
                clave = lexema.upper()
            else:
                clave = lexema
            if lexema == reservadas.get(clave,None):
                if contadorFilas == 1 and len(lexema) == 4:
                    objeto = self.getLexema(lexema)
                    tokenAntiguo = None
                    for elem in self.listadoTokens:
                        if elem.lexema == objeto.lexema:
                            tokenAntiguo = elem
                    tokenAntiguo.id = "Palabra reservada"
                    tokenAntiguo.lexema = lexema
                    tokenAntiguo.fila = objeto.getFila() + 1
                elif contadorColumnas == 1 and len(lexema) == 7:
                    objeto = self.getLexema(lexema)
                    tokenAntiguo = None
                    for elem in self.listadoTokens:
                        if elem.lexema == objeto.lexema:
                            tokenAntiguo = elem
                    tokenAntiguo.id = "Palabra reservada"
                    tokenAntiguo.lexema = lexema
                    tokenAntiguo.fila = objeto.getFila() + 1
```

```python
def getLexema(self, lexema):
    for i in self.listadoLexemas_sinRepetir:
        lex = i.lexema[1:len(i.lexema)-1]
        if lex == lexema:
            return i
    return None
```

Además nos auxiliamos de getLexema() para verificar que dentro de las cadenas tipo string no vengan palabras reservadas poque en la etiqueta de tabla en elemento si tenemos dichas palabras, por lo tanto, debemos verificarlo.

Luego vamos a crear un nuevo listado a partir de las cadenas obtenidas durante el método de obtenerCadenas() , este método devolverá todos los “valores” que se pasan como strings y retorna una cadena con todos estos, luego vamos a descomponer dichas cadenas con el método self.descomponerCsdenas() mediante u bucle for en cada lexema del arreglo de las cadenas sin comillas. Teniendo el listado de las cadenas descompuestas vamos a crear los tokens y posteriormente a exportarlos.

```python
def listadoCadenas(self) -> list:
    cadenas = []
    for i in self.listadoLexemas:
        first_char = i.lexema[0]
        last_char = i.lexema[-1]
        condicional = first_char == '\"' and last_char == '\"'
        if condicional:
            cadenas.append(i)
        else:
            continue
    return cadenas
```

```python
def descomponerCadenas(self, cadena):
    valor = ""
    lexema = ""
    puntero = 0
    columna = 0

    while cadena:
        caracter = cadena[puntero]
        puntero += 1

        if caracter == '\"':
            '''
                " f i l a "
                0 1 2 3 4 5
                [puntero:puntero+5] = "fila"
                [puntero+1:puntero+4] = fila

                " c o l u m n a "
                0 1 2 3 4 5 6 7 8
                [puntero:puntero+6] = "columna"
                [puntero+1:puntero+7] = columna
            '''
            if cadena == '"fila"' and len(cadena) == 6:
                cadenaInteres = cadena[puntero:puntero+4]
                valor = cadenaInteres
                cadena = cadena[puntero+4:]
                puntero = 0
                columna += 1
            elif cadena == '"columna"' and len(cadena) == 9:
                cadenaInteres = cadena[puntero:puntero+7]
                valor = cadenaInteres
                cadena = cadena[puntero+7:]
                puntero = 0
                columna += 1
            else:
                cadena = cadena[1:]
                puntero = 0
                columna += 1
        elif caracter.isalpha():
            lexema, cadena = self.armar_lexema(cadena[puntero-1:],2)
            if lexema and cadena:
                valor = f'"{lexema}"'
                columna += len(lexema) + 1
                puntero = 0
        elif caracter.isdigit():
            lexema, cadena = self.armar_lexema(cadena[puntero-1:],2)
            if lexema and cadena:
                if len(lexema) == 6:
                    valor = lexema
                else:
                    valor = int(lexema)
                columna += len(lexema) + 1
                puntero = 0
        else:
            cadena = cadena[1:]
            puntero = 0
            columna += 1
    return valor
```

```
def getTokens(self) -> str:
    self.listadoErrores
    formato = '<table align="center" border="black" height="50%" width="50%">\n'
    formato += '<tr bgcolor="black">\n'
    formato += '<th><font color="white">Token</font</th>\n'
    formato += '<th><font color="white">Lexema</font</th>\n'
    formato += '<th><font color="white">Fila</font</th>\n'
    formato += '<th><font color="white">Columna</font</th>\n'
    formato += '</tr>\n'
    for i in range(len(self.listadoTokens)):
        formato += '<tr>\n'
        token = self.listadoTokens[i]
        formato += token.execute() + '</tr>\n'
    formato += '</table>\n'
    return formato
```

```
def exportarTokens(self) -> None:
    nombre = "ListaTokens"+".html"
    with open(nombre, 'w') as archivo:
        archivo.write('<!DOCTYPE html>\n')
        archivo.write('<html>\n')
        archivo.write('<head>\n')
        archivo.write('<title>Errores</title>\n')
        archivo.write('</head>\n')
        archivo.write('<body font-size="16">\n')
        archivo.write('<font size="6">\n')
        archivo.write(self.getTokens())
        archivo.write('</font>\n')
        archivo.write('</body>\n')
        archivo.write('</html>\n')
```

Ahora para traducir el código, primero debemos crear un arreglo auxiliar que elimine la etiqueta de inicio y sus llaves. Luego vamos a eliminar las comillas (los índices que contengan un string con comillas) y vamos a dejar puramente los valores, sin comillas.

Para poder leer el cuerpo debemos separarlo por bloques, con la función creandoBloques() obtendremos el encabezado y el cuerpo cuando le pasemos el listado sin la etiqueta de inicio, del cual vamos a querer solamente desde ciertos caracteres hasta otros, para ello aplicamos la misma metodología que con el inicio, borramos los primeros caracteres para quedarnos con el contenido importante.

Por último vamos a llamar al método crearEtiquetas() para pasarle el arreglo de lexemas que contienen el cuerpo y esta función es otro analizador, pero esta vez como no hay posibilidad de que haya errores porque ya los depuramos, entonces leerá índice por índice del arreglo de tal forma que vaya leyendo pares dentro de etiquetas mayores entre llaves del tipo:

$$PalabraReservada: \{atributo : valor\ ; atributo : valor\ \ldots\}$$

```
def crearEtiquetas(self,listado):
    diccionario = {}
    nombreEtiqueta = ""
    nombreAtributo = ""
    etiqueta = ""
    css = ""
    posiciones = 0
    numerador = 0
    valores = []
    while listado:
        actual = listado[posiciones]
        posiciones += 1
        siguiente= ""
        if actual.isalpha():
            if actual[0].isupper():
                if actual == reservadas.get(actual.upper(),None):
                    nombreEtiqueta = actual
        if len(listado) > posiciones:
            siguiente = listado[posiciones]

        if actual+siguiente != "},," and actual+siguiente != "}]":
            if actual == "elemento" and siguiente == ":":
                indice1 = listado.index(siguiente)
                listadoAux = listado[indice1+1:]
                posAux = 0
                for j in listadoAux:
                    posiciones += 1
                    posAux += 1
                    j_siguiente = listadoAux[posAux]
                    if j+j_siguiente != "};":
                        if j != ":" and j != "{":
                            valores.append(j)
                        else:
                            continue
                    else:
                        diccionarioAux = {}
                        for k in enumerate(valores):
                            index, elemento = k
                            if elemento != ",":
                                if elemento == "fila":
                                    diccionarioAux[elemento] = valores[index+1]
                                elif elemento == "columna":
                                    diccionarioAux[elemento] = valores[index+1]
                                else:
                                    diccionarioAux["contenido"] = elemento
                            else:
                                continue
                        diccionario[actual+str(numerador)] = diccionarioAux
                        listado = listado[posiciones+2:]
                        posiciones = 0
                        numerador += 1
                        break
            elif siguiente == ":" or siguiente == "=":
                if actual == "texto":
                    nombreAtributo = actual
                    listado = listado[posiciones+1:]
                    posiciones = 0
                elif reservadas.get(actual.upper(),None)[0].isupper():
                    nombreEtiqueta = actual
                    listado = listado[posiciones+2:]
                    posiciones = 0
                elif reservadas.get(actual.upper(),None)[0].islower():

                    nombreAtributo = actual
                    listado = listado[posiciones+1:]
                    posiciones = 0
            elif actual == ";" or actual == ",":
                listado = listado[posiciones+1:]
                posiciones = 0
            else:
                diccionario[nombreAtributo] = actual
                listado = listado[posiciones+1:]
                posiciones = 0
                continue
        else:
            e = Etiqueta(nombreEtiqueta)
            e.atributos = diccionario
            name = ""
            if nombreEtiqueta != "Salto":
                et = e.crearEtiquetahtml()
            else:
                cantidad = int(e.atributos["cantidad"])
                nombreEtiqueta = "<br>"*cantidad
                name = "Salto"

            if name == "Salto":
                self.etiquetas.append(nombreEtiqueta)
                name = ""
            else:
                self.etiquetas.append(et)

            listado = listado[posiciones+1:]
            posiciones = 0
            numerador = 0
            diccionario = {}
            valores = []
```

Para exportar el resultado utilizamos el metodo exportandoEtiquetas() al cual que pasaremos como parametro el header, como se podrá notar en el codigo anterior el metodo interno de la clase Etiqueta hace el trabajo de traducir el codigo desde el JSON hasta el HTML.

```
def exportandoEtiquetas(self, head):
    # for i in self.etiquetas:
    #     print(i)
    self.html += f'<!DOCTYPE html>\n'
    self.html += f'<html>\n'
    self.html += f'<head>\n'
    self.html += f'<title>{head}</title>\n'
    self.html += f'<link rel="stylesheet" type="text/css" href="estilos.css">\n'
    self.html += f'</head>\n'
    self.html += f'<body>\n'

    for i in self.etiquetas:
        if i[0] != "<":
            nombre = "estilos"+".css"
            with open(nombre, 'w') as archivo:
                archivo.write(f'body'+ " {" + "\n\t" + i + "\n" + "}" + "\n")
                archivo.write(f'.codigo'+ " {" + "\n\t" + "background-color: black;" + "\n\t" + "text-align: center;" + "\n\t" + "padding: 3px;" + "\n" +"}")
        else:
            self.html += i + "\n"

    self.html += f'</body>\n'
    self.html += f'</html>\n'
    nombre = "pagina"+".html"
    open(nombre, 'w').write(self.html)
```

Y este metodo de exportación crea directamente la página html y el css asociado al fondo de la página. El detalle del procedimiento a seguir está en la función progreso()

```
def progreso(self, cadena) -> Union[List, str, Tuple[str, List]]:

    # literalmente el analizador lexico

    self.analizar_entrada(cadena)

    # manejo de errores

    detenerse = self.analizarErrores()
    if type(detenerse) == list:
        # retorna una Lista
        return detenerse
    elif type(detenerse) == str:
        # retorna un string
        return detenerse

    # creacion de tokens

    self.crearTokens()
    cadenas = self.listadoCadenas()
    listadoDescomposiciones = []
    for c in cadenas:
        valor = self.descomponerCadenas(c.lexema)
        listadoDescomposiciones.append(valor)
    self.crearTokens(False, listadoDescomposiciones)
    self.exportarTokens()

    # comienzo de traducción de datos

    listadoAux = self.listadoLexemas[3:len(self.listadoLexemas)-1]
    for l in listadoAux:
        if l.lexema[0] == '\"':
            longitud = len(l.lexema) - 1
            cadenaTexto = l.lexema[1:longitud]
            l.lexema = cadenaTexto
```

```
    # obteniendo bloques de datos
    listadoBloques = self.creandoBloques(listadoAux)
    encabezado = listadoBloques[0]
    cuerpo = listadoBloques[1]
    cuerpo.append("]")
    self.crearEtiquetas(cuerpo)

    # traduciendo y creando HTML
    self.exportandoEtiquetas(encabezado[2])

    # retornando mensaje de exito
    return "Archivo generado con exito", self.html
```

Cuando se finalice la función progreso() retornará valores validos hacia la interface del main.

Interfaz

Es una clase que crea una ventana donde se puede escribir codigo o abrir un explorador de archivos para seleccionar el codigo, por lo tanto, también tiene un botón de traducir, el codigo solamente implementa objetos tipo botón, area de texto y labels. Además se configuró una ventana auxiliar para mandar mensajes, por ejemplo de error o de finalización de ejecución y luego muestra el codigo traducido a la derecha, además de exportarlo en html.

```
class Ventana():
    def __init__(self, root=tk.Tk()):
        self.root = root
        self.data = []

        self.root.title("Proyecto 1 - Lenguajes Formales y de Programacion")
        ancho_ventana = self.root.winfo_reqwidth()
        alto_ventana = self.root.winfo_reqheight()
        pos_x = int(self.root.winfo_screenwidth() / 4 - ancho_ventana / 4)
        pos_y = int(self.root.winfo_screenheight() / 4 - alto_ventana / 4)
        self.root.geometry(f"900x550+{pos_x}+{pos_y}")

        self.root.resizable(False, False)

        self.boton_abrir = tk.Button(self.root, text="Abrir archivo", command=self.abrir_explorador)
        self.boton_abrir.grid(row=0, column=0, padx=10, pady=10, sticky="w")

        self.boton_borrar_entrada = tk.Button(self.root, text="Borrar entrada", command=lambda: self.area_entrada.delete("1.0", "end"))
        self.boton_borrar_entrada.grid(row=0, column=1, padx=10, pady=10, sticky="w")

        self.boto_borrar_salida = tk.Button(self.root, text="Borrar salida", command=self.borrar)
        self.boto_borrar_salida.grid(row=1, column=1, padx=10, pady=10, sticky="w")

        self.entrada = tk.Label(self.root, text="Texto de entrada")
        self.entrada.grid(row=2, column=0, padx=10, pady=10, sticky="w")

        self.salida = tk.Label(self.root, text="Traduccion")
        self.salida.grid(row=2, column=1, padx=10, pady=10, sticky="w")

        self.area_entrada = tk.Text(self.root, width=50, height=20)
        self.area_entrada.grid(row=4, column=0, padx=10, pady=10)

        self.area_salida = tk.Text(self.root, width=50, height=20, state="disabled")
        self.area_salida.grid(row=4, column=1, padx=10, pady=10)

        self.boton_traducir = tk.Button(self.root, text="Traducir", command=self.traducir)
        self.boton_traducir.grid(row=6, column=0, padx=10, pady=10, sticky="e")
```

```
def borrar(self):
    self.area_salida.config(state="normal")
    self.area_salida.delete("1.0", "end")
    self.area_salida.config(state="disabled")

def abrir_explorador(self):
    filepath = filedialog.askopenfilename(initialdir="/", title="Selecciona un archivo",
                                          filetypes=(("Archivos de texto", "*.json"), ("Archivos de texto", "*.lfp"),
                                                     ("Todos los archivos", "*.*")))
    self.data = [filepath, filepath.split("/")[-1], open(filepath, "r").read()]
    self.area_entrada.delete("1.0", "end")
    self.area_entrada.insert("1.0", str(self.data[2]))
```

```python
def ventanMensaje(self, contenido, advertencia="error"):
    ventana = tk.Tk()
    if advertencia == "error":
        ventana.title("Errores")
    else:
        ventana.title("Mensaje")

    # Centrar la ventana de errores
    ancho_ventana = ventana.winfo_reqwidth()
    alto_ventana = ventana.winfo_reqheight()

    pos_x = int(ventana.winfo_screenwidth() / 4 - ancho_ventana / 4)
    pos_y = int(ventana.winfo_screenheight() / 4 - alto_ventana / 4)

    ventana.geometry(f"600x200+{pos_x}+{pos_y}")

    ventana.resizable(False, False)

    fuente = font.Font(size=12)
    negrita = font.Font(size=12, weight="bold")

    enunciado = tk.Label(ventana, text=contenido, font=negrita)
    enunciado.pack(expand=True, fill="both")

    if advertencia == "error":
        enunciado2 = tk.Label(ventana, text=f"Verifica el archivo de entrada {self.data[1]}", font=fuente)
        enunciado2.pack(expand=True, fill="both")
    else:
        pass

    ventana.mainloop()
```

```
def traducir(self):
    entrada = len(str(self.area_entrada.get("1.0", "end")))

    if entrada <= 5:
       return self.ventanMensaje("No hay contenido para traducir")
    else:
        # configurando contenido
        aux, nombre, contenidoArchivo = self.data
        ruta = aux.replace(nombre, "")
        contenidoArea = str(self.area_entrada.get("1.0", "end"))
        contenido = ""
        if contenidoArchivo != contenidoArea:
            contenido = contenidoArea
        else:
            contenido = contenidoArchivo
        # analizandor lexico
        ejecucion = Analizador().progreso(contenido)
        if type(ejecucion) == list:
            self.area_entrada.delete("1.0", "end")
            self.area_salida.config(state="normal")
            self.area_salida.delete("1.0", "end")
            self.area_salida.insert("1.0", "Error")
            self.area_salida.config(state="disabled")
            if ejecucion[0] != "Encabezado" and ejecucion[0] != "Cuerpo" and ejecucion[0] != "Inicio":
                er, con = ejecucion
                self.area_salida.config(state="normal")
                self.area_salida.delete("1.0", "end")
                self.area_salida.insert("1.0", con)
                self.area_salida.config(state="disabled")
                self.ventanMensaje(er)
            else:
                self.ventanMensaje(f'Error la etiqueta {ejecucion[0]} se repite {ejecucion[1]} veces en el archivo')
```

```
            elif ejecucion == "errores generados en archivo ListaErrores.html":
                self.area_entrada.delete("1.0", "end")
                self.area_salida.config(state="normal")
                self.area_salida.delete("1.0", "end")
                self.area_salida.insert("1.0", "Errores generados en archivo ListaErrores.html")
                self.area_salida.config(state="disabled")
                self.ventanMensaje(ejecucion)
                del ejecucion
            else:
                men, con = ejecucion
                self.area_salida.config(state="normal")
                self.area_salida.delete("1.0", "end")
                self.area_salida.insert("1.0", con)
                self.area_salida.config(state="disabled")
                self.ventanMensaje(men, "men")
                del ejecucion

if __name__ == "__main__":
    # instanciando la interfaz
    app = Ventana()
    app.root.mainloop()
```