JAVABRIDGE – PROYECTO 2

MANUAL TÉCNICO

LENGUAJES FORMALES Y DE PROGRAMACIÓN

202405935
Erwin Alejandro Girón Menéndez

Introducción

El presente documento describe el diseño, implementación y funcionamiento del traductor de código JAVA a código Python, una aplicación web que permite convertir código fuente escrito en Java a su equivalente en Python de manera automática.

El sistema implementa un proceso completo de análisis léxico y sintáctico mediante autómatas finitos deterministas (AFD). La aplicación proporciona una interfaz web intuitiva que permite a los usuarios cargar archivos Java, visualizar la traducción a Python, y generar reportes detallados de tokens y errores encontrados durante el proceso de análisis.

El traductor es capaz de reconocer y convertir estructuras fundamentales del lenguaje Java, incluyendo declaraciones de variables, estructuras de control (condicionales y bucles), operadores aritméticos y lógicos, comentarios, y sentencias de impresión, adaptándolas a la sintaxis y convenciones de Python.

Objetivo General

Desarrollar un traductor automático de código fuente que convierta programas escritos en Java a su equivalente funcional en Python, implementando técnicas de compilación que incluyan análisis léxico mediante autómatas finitos deterministas (AFD con el fin de proporcionar una herramienta educativa y práctica que facilite la comprensión de los procesos de traducción entre lenguajes de programación.

Objetivos Específicos

-Implementar un analizador léxico (Lexer) basado en autómatas finitos deterministas que sea capaz de tokenizar código Java, reconociendo palabras reservadas, identificadores, operadores, símbolos, literales (números, cadenas, caracteres) y comentarios.

-Desarrollar un analizador sintáctico (Parser) que valide la estructura gramatical del código Java y construya las representaciones necesarias para la traducción a Python.

Especificaciones Técnicas

- Procesador: Intel Core i3 de 2.0 GHz o equivalente (Core i5 o superior).
- Memoria RAM: 4 GB mínimo (8 GB recomendados)
- Espacio en Disco: 500 MB para descarga de reportes
- 500 MB adicionales para guardar archivos Python

- Pantalla: Resolución mínima de 1280x720 (se recomienda 1366x768 o superior).
- Periféricos: Teclado y mouse (recomendado).

Especificaciones de Software

- HTML5: Estructura de la interfaz web

- CSS3: Estilos y diseño responsivo

- JavaScript: Lógica de la aplicación

Módulos del Programa

- `Lexer.js`: Implementación del analizador léxico

- `Parser.js`: Implementación del analizador sintáctico

- `Token.js`: Clase para representar tokens

- `Error.js`: Clase para errores léxicos

- `ErrorSin.js`: Clase para errores sintácticos

- `main.js`: Controlador principal de la aplicación

Lógica de la aplicación

Nombre del paquete	Descripción	Objetos dentro del paquete
Errores	Se encarga de analizar los dos tipos de errores posibles en el programa, léxicos y sintácticos	ErrorLex.js ErrorSin.js
Lexer	Primera fase de análisis de un compilador, se encarga de analizar los tokens de un lenguaje JAVA	lexer.js

Parser	Segunda fase de análisis de un compilador, este se encarga de revisar la parte sintáctica por medio de los tokens proporcionados por el lexer	Parser.js
Token	Clase que encarga de inicializar los datos de los tokens válidos	Token.js
Fronted	Este se encarga de la interfaz que verá el usuario al momento de usar el programa	Index.html Style.css
----	Se encarga de la lógica del programa conectando la interfaz gráfica con los analizadores, este permite que se lleve a cabo los diferentes procesos correctamente	main.js

Backend

Clase ErrorLex.js

Ruta: ../Error./ErrorLex.js

Esta clase es un constructor prácticamente en donde se inicializan los atributos que tendrán los errores léxicos

```
class Error{
    constructor(tipo, descripcion, linea, columna){
        this.tipo = tipo;
        this.descripcion = descripcion;
        this.linea = linea;
        this.columna = columna;
    }
}


export { Error }
```

Clase ErrorSin.js

Ruta: ../Error/ErrorSin.js

Esta clase tiene una forma similar a la clase ErrorLex, pero este constructor es especialmente para los errores de tipo sintácticos para mantener un mejor orden en los errores

```
class ErroresSin {
    constructor(type, valor, mensaje, linea, columna) {
        this.type = type;
        this.valor = valor;
        this.mensaje = mensaje;
        this.linea = linea;
        this.columna = columna;
    }
}

export { ErroresSin }
```

Clase Lexer

Ruta: ../Lexer/lexer.js

Esta clase se encarga del análisis léxico del programa, primeramente tenemos el constructor:

Este constructor recibe tiene inicializado la posición en la cual empieza el conteo de tokens, la columna donde inicia el conteo, la línea, un arreglo que almacena los tokens, un arreglo que almacena los errores léxicos y el código que va a ser el parámetro del análisis

```
1   class lexer {
2
3     constructor(code) {
4       this.posicion = 0 // Posición actual en el código
5       this.columna = 1 // Columna actual (para reportes de error)
6       this.linea = 1 // Línea actual (para reportes de error)
7       this.token = [] // Array de tokens generados
8       this.error = [] // Array de errores léxicos encontrados
9       this.code = code // Código fuente a analizar
10    }
```

Luego tendremos un método para verificar si el token es una letra, tendrá un parámetro c el cual retornará mientras sea mayor o igual a y menor o igual z, siendo mayúscula o minúscula y a la vez se retornará si es un guion bajo

```
14    es_letra(c) {
15      return (c >= "a" && c <= "z") || (c >= "A" && c <= "Z") || c === "_"
16    }
```

Tenemos un método el cual detectará si el token es un dígito, teniendo también un parámetro c el cual se retornará mientras sea mayor igual a “0” y menos o igual a “9”

```
20    es_digito(c) {
21      return c >= "0" && c <= "9"
22    }
```

El siguiente método sirve para ir avanzado, aumenta en una unidad la posición y la columna, este sirve para avanzar en el código y determinar el análisis léxico

```
26    seguir_avanzando() {
27      this.posicion++
28      this.columna++
29    }
```

Luego tendremos el método principal llamado analizar, el cual hace todo el análisis léxico, y apoyado en los métodos estructurados anteriormente. Primeramente tendremos la inicialización de variables, las cuales también se encuentran en el constructor

```
33    analizar() {
34      // Reiniciar estado del analizador
35      this.token = []
36      this.error = []
37      this.posicion = 0
38      this.linea = 1
39      this.columna = 1
```

Luego tenemos un método el cual se encarga de recorrer carácter por carácter el código, mientras la posición ser menor que la longitud del código, una variable c hará referencia a la posición actual en el código

```
42    // Recorrer todo el código carácter por carácter
43    while (this.posicion < this.code.length) {
44      const c = this.code[this.posicion]
45
46
```

Luego tendremos una sentencia if la cual condiciona que si el parámetro ‘ c ‘ es igual a un espacio o una tabulación, se ignorará y se aumentará en una unidad su línea y columna

```
48      // Ignorar espacios en blanco y tabulaciones
49      if (c === " " || c === "\t") {
50        this.seguir_avanzando()
51        continue
52      }
```

Luego tendremos otra sentencia parecida, pero para los saltos de línea, este nos dice que si el parámetro ‘ c ‘ es igual a un salto de línea, la línea aumente una unidad y la columna vuelve al inicio (columna 1), y se seguirá avanzando en la lectura

```
56      // Manejar saltos de línea
57      if (c === "\n") {
58        this.linea++
59        this.columna = 1
60        this.seguir_avanzando()
61        continue
62      }
```

Luego tendremos un método para tockenizar comentarios de línea, este nos dice que si el parámetro ‘ c ‘ es un ‘ / ‘ y su posición siguiente es otro ‘ / ‘ es un comentario de línea, la columna de inicio del comentario de almacena y le crea una variable comentario vacía la cual recibe el comentario. Mientras la posición sea menor a la longitud del código y en dicha posición sea diferente a un salto de línea de analiza el comentario y se sigue avanzando.

Luego de analizar el comentario se pushea al arreglo de tokens con sus respectivos datos

```
66        // Detectar comentarios de línea (//) y tokenizarlos
67    if (c === "/" && this.code[this.posicion + 1] === "/") {
68      const colInicio = this.columna
69      let comentario = ""
70
71      while (this.posicion < this.code.length && this.code[this.posicion] !== "\n") {
72        comentario += this.code[this.posicion]
73        this.seguir_avanzando()
74      }
75
76      this.token.push({
77        type: "COMMENT_LINE",
78        value: comentario,
79        line: this.linea,
80        column: colInicio
81      })
82      continue
83    }
```

Luego tendremos este método que detecta comentarios de bloque para tockenizarlos básicamente tiene un enfoque parecido al método de comentario de línea ya que analiza la posición actual y la siguiente para chequear ‘ / ‘ y luego ‘ * ‘, determina la columna de inicio y hay una variable que almacena el comentario respectivo

```
87        // Detectar comentarios de bloque (/* */) y tokenizarlos
88    if (c === "/" && this.code[this.posicion + 1] === "*") {
89      const colInicio = this.columna
90      let comentario = ""
91
92      comentario += this.code[this.posicion]
93      this.seguir_avanzando() // Consumir /
94      comentario += this.code[this.posicion]
95      this.seguir_avanzando() // Consumir *
96
97      while (
98        this.posicion < this.code.length &&
99        !(this.code[this.posicion] === "*" && this.code[this.posicion + 1] === "/")
100     ) {
101       comentario += this.code[this.posicion]
102       this.seguir_avanzando()
103     }
104
105     if (this.posicion < this.code.length) {
106       comentario += this.code[this.posicion]
107       this.seguir_avanzando() // Consumir *
108       comentario += this.code[this.posicion]
109       this.seguir_avanzando() // Consumir /
110     }
111
112     this.token.push({
113       type: "COMMENT_BLOCK",
114       value: comentario,
115       line: this.linea,
116       column: colInicio
117     })
118     continue
119   }
```

Luego tendremos el método encargado de reconocer Strings

```
152   // Reconocer caracteres (entre comillas simples)
153   if (c === "'") {
154     const colInicio = this.columna
155     let valor = "'" // <-- INCLUIR comilla de apertura
156     this.seguir_avanzando() // Consumir comilla de apertura
157
158     // Leer hasta encontrar la comilla de cierre
159     while (this.posicion < this.code.length && this.code[this.posicion] !== "'") {
160       valor += this.code[this.posicion]
161       this.seguir_avanzando()
162     }
163
164     // Verificar si se cerró correctamente
165     if (this.posicion < this.code.length) {
166       valor += "'" // <-- INCLUIR comilla de cierre
167       this.seguir_avanzando() // Consumir comilla de cierre
168       this.token.push({ type: "CHAR", value: valor, line: this.linea, column: colInicio })
169     } else {
170       // Error: carácter sin cerrar
171       this.error.push({
172         tipo: "Lexico",
173         descripcion: "Carácter sin cerrar",
174         linea: this.linea,
175         columna: colInicio,
176       })
177     }
178     continue
179   }
```

Continuamos con la sentencia la cual se encarga de reconocer palabras claves o identificadores

```
211        // Reconocer palabras clave e identificadores
212        if (this.es_letra(c)) {
213          const colInicio = this.columna
214          let valor = ""
215
216          // Leer letras y dígitos (identificadores pueden contener números después de la primera letra)
217          while (
218            this.posicion < this.code.length &&
219            (this.es_letra(this.code[this.posicion]) || this.es_digito(this.code[this.posicion]))
220          ) {
221            valor += this.code[this.posicion]
222            this.seguir_avanzando()
223          }
```

Le damos un diccionario de palabras reservadas para que saber cuales son esas palabras

```
226          const palabrasReservadas = {
227            public: "PUBLIC",
228            class: "CLASS",
229            static: "STATIC",
230            void: "VOID",
231            main: "MAIN",
232            String: "STRING_TYPE",
233            args: "ARGS",
234            int: "INT_TYPE",
235            double: "DOUBLE_TYPE",
236            char: "CHAR_TYPE",
237            boolean: "BOOLEAN_TYPE",
238            true: "TRUE",
239            false: "FALSE",
240            if: "IF",
241            else: "ELSE",
242            for: "FOR",
243            while: "WHILE",
244            System: "SYSTEM",
245            out: "OUT",
246            println: "PRINTLN",
247          }
```

Para luego por medio de una variable tipo que será un valor del diccionario, determinará el tipo final, si es palabra reservada o un identificador y se tokenizará y se hará push al arreglo de tokens

```
249         const tipo = palabrasReservadas[valor]
250
251         // Determinar si es palabra reservada o identificador
252         const tipoFinal = tipo || "IDENTIFICADOR"
253         this.token.push({ type: tipoFinal, value: valor, line: this.linea, column: colInicio })
254         continue
255       }
```

Luego tendremos una sentencia para determinar caracteres dobles como se muestra en el comentario, también por medio de un diccionario de posibles operadores dobles

```
257     // Reconocer operadores de dos caracteres (==, !=, <=, >=, ++, --, +=, -=)
258     if (this.posicion < this.code.length - 1) {
259       const doble = c + this.code[this.posicion + 1]
260       const operadoresDobles = {
261         "==": "IGUAL_IGUAL",
262         "!=": "DIFERENTE",
263         "<=": "MENOR_IGUAL",
264         ">=": "MAYOR_IGUAL",
265         "++": "INCREMENTO",
266         "--": "DECREMENTO",
267         "+=": "MAS_IGUAL",
268         "-=": "MENOS_IGUAL",
269       }
270
271       if (operadoresDobles[doble]) {
272         const colInicio = this.columna
273         this.token.push({ type: operadoresDobles[doble], value: doble, line: this.linea, column: colInicio })
274         this.seguir_avanzando()
275         this.seguir_avanzando()
276         continue
277       }
278     }
```

Luego tendremos un diccionario para símbolos de un solo carácter, y una sentencia para reconocerlos

```
280     // Reconocer símbolos de un solo carácter
281     const simbolos = {
282       "{": "LLAVE_ABIERTA",
283       "}": "LLAVE_CERRADA",
284       "(": "PARENTESIS_ABIERTO",
285       ")": "PARENTESIS_CERRADO",
286       "[": "CORCHETE_ABIERTO",
287       "]": "CORCHETE_CERRADO",
288       ";": "PUNTO_Y_COMA",
289       ",": "COMA",
290       ".": "PUNTO",
291       "=": "ASIGNACION",
292       "+": "SUMA",
293       "-": "RESTA",
294       "*": "MULTIPLICACION",
295       "/": "DIVISION",
296       "<": "MENOR_QUE",
297       ">": "MAYOR_QUE",
298     }
299
300     if (simbolos[c]) {
301       const colInicio = this.columna
302       this.token.push({ type: simbolos[c], value: c, line: this.linea, column: colInicio })
303       this.seguir_avanzando()
304       continue
305     }
```

Luego, si no es ninguno de los casos definidos, se pushea un error léxico y se sigue avanzando para seguir analizando

```
307       // Si llegamos aquí, el carácter no es reconocido (error léxico)
308       this.error.push({
309         tipo: "Lexico",
310         descripcion: `Carácter no reconocido: '${c}'`,
311         linea: this.linea,
312         columna: this.columna,
313       })
314       this.seguir_avanzando()
315     }
```

Se retornan los tokens, se cierra la clase lexer y se exporta para tener acceso global

Clase Parser

Ruta: ../Parser/parser.js

La clase Parser funciona como el núcleo del traductor de Java a Python, implementando un analizador sintáctico que procesa tokens y genera código Python equivalente. Su funcionamiento comienza determinando si el código Java tiene una estructura completa de clase (con public class) o solo fragmentos de código, teniendo dos caminos para analiza… en analizarConClase() o analizarSinClase() respectivamente. Cuando detecta una clase completa, valida meticulosamente cada componente de la sintaxis Java - desde la declaración public class NombreClase hasta el método main con su firma exacta public static void main(String[] args). Utilizando un sistema de posición incremental (this.pos), el parser recorre los tokens aplicando métodos especializados para cada estructura: traducirIf() para condicionales, traducirFor() y traducirWhile() para bucles, declaracionVariable() para declaraciones, y traducirPrint() para System.out.println. Maneja la indentación automáticamente, convierte tipos de datos y expresiones booleanas (true→True, false→False), procesa secuencias de escape en strings, y detecta errores sintácticos almacenándolos en un array mediante la clase ErroresSin. El resultado final es código Python correctamente estructurado que preserva la lógica del código Java original.

```
//_____________________________________________________________________________
//clase de errores sintácticos
class ErroresSin {
  constructor(tipo, valor, mensaje, linea, columna) {
    this.tipo = tipo
    this.valor = valor
    this.mensaje = mensaje
    this.linea = linea
    this.columna = columna
  }
}
//_____________________________________________________________________________



class Parser {
//_____________________________________________________________________________

//constructor de clase Parser
  constructor(tokens) {
    this.tokens = tokens // Tokens a analizar
    this.pos = 0 // Posición actual en el array de tokens
    this.errors = [] // Array de errores sintácticos encontrados
    this.pythonCode = "" // Código Python generado
    this.indent = "" // Indentación actual
  }
  //_____________________________________________________________________________
```

```
  //_____________________________________________________________________________
  //función analizar clase (estructura de clase) si hay, de lo contrario, analiza fragmentos de código
  analizar() {
    // Verificar si el código comienza con "public class" (estructura completa)
    if (this.tokens[0] && this.tokens[0].type === "PUBLIC") {
      // Modo con estructura de clase completa
      return this.analizarConClase() //análisis en caso de que sí venga una estructura de clase
    } else {
      // Modo sin estructura de clase (solo sentencias)
      return this.analizarSinClase()
    }
  }
  //_____________________________________________________________________________
```

```
48  //_____________________________________________________________________________
49  //analizando estructura con clase
50    analizarConClase() {
51      // 1. Verificar "public"
52      if (!this.tokens[this.pos] || this.tokens[this.pos].type !== "PUBLIC") {
53        const token = this.tokens[this.pos]
54        this.errors.push(
55          new ErroresSin(
56            "SINTACTICO",
57            token?.value || "EOF",
58            "Se esperaba 'public' al inicio del archivo. El archivo debe comenzar con 'public class NombreClase'",
59            token?.line || 1,
60            token?.column || 1,),)
61        return { errors: this.errors, python: this.pythonCode }
62      }
63      this.pos++ //avanza a "class"
64
65
66
67
68      // 2. Verificar "class"
69      if (!this.tokens[this.pos] || this.tokens[this.pos].type !== "CLASS") {
70        const token = this.tokens[this.pos]
71        this.errors.push(
72          new ErroresSin(
73            "SINTACTICO",
74            token?.value || "EOF",
75            "Se esperaba 'class' después de 'public'",
76            token?.line || 1,
77            token?.column || 1,
78          ),)
79        return { errors: this.errors, python: this.pythonCode }
80      }
81      this.pos++ //avanza al nombre de cla clase
```

```
86      // 3. Verificar nombre de la clase (identificador)
87      if (!this.tokens[this.pos] || this.tokens[this.pos].type !== "IDENTIFICADOR") { //si no existe un token en la pos actual ni su valor es un iden
88        const token = this.tokens[this.pos]
89        this.errors.push(
90          new ErroresSin(
91            "SINTACTICO",
92            token?.value || "EOF",
93            "Se esperaba el nombre de la clase después de 'class'",
94            token?.line || 1,
95            token?.column || 1,
96          ),
97        )
98        return { errors: this.errors, python: this.pythonCode } //se retorna el error
99      }
100     const nombreClase = this.tokens[this.pos].value  //el identificador se almacena en nombreClsae
101     this.pos++ //avanza en los tokens
102
103     this.pythonCode += `class ${nombreClase}:\n` //parte traducida a python + incremento de identación
104     // Incrementar indentación para todo el contenido de la clase
105     this.indent += "    "
```

```
109     // 4. Verificar llave de apertura de la clase
110     if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "{") {
111       const token = this.tokens[this.pos]
112       this.errors.push(
113         new ErroresSin(
114           "SINTACTICO",
115           token?.value || "EOF",
116           `Se esperaba '{' después del nombre de la clase '${nombreClase}'`,
117           token?.line || 1,
118           token?.column || 1,
119         ),
120       )
121       return { errors: this.errors, python: this.pythonCode }
122     }
123     this.pos++ //avanzando al siguiente token
124
125
126
127     // 5. Verificar "public static void main(String[] args) {"
128     if (!this.validarMetodoMain()) {
129       return { errors: this.errors, python: this.pythonCode }
130     }
```

```
132     // 6. Procesar el contenido del método main
133     this.procesarSentencias() //método que contiene un switch and case para varios casos de sentencias
134
135     // 7. Verificar llave de cierre del método main
136     if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "}") {
137       const token = this.tokens[this.pos]
138       this.errors.push(
139         new ErroresSin(
140           "SINTACTICO",
141           token?.value || "EOF",
142           "Se esperaba '}' para cerrar el método main",
143           token?.line || 1,
144           token?.column || 1,
145         ),
146       )
147       return { errors: this.errors, python: this.pythonCode }
148     }
149     this.pos++
150
151     this.indent = this.indent.slice(0, -4)
```

```
    if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "}") {
      const token = this.tokens[this.pos]
      this.errors.push(
        new ErroresSin(
          "SINTACTICO",
          token?.value || "EOF",
          "Se esperaba '}' para cerrar la clase",
          token?.line || 1,
          token?.column || 1,
        ),
      )
      return { errors: this.errors, python: this.pythonCode }
    }
    this.pos++




    if (this.pos < this.tokens.length) { //validando que no hayan instrucciones fuera de la clase (después de la llave de cierre)
      const token = this.tokens[this.pos]
      this.errors.push(
        new ErroresSin(
          "SINTACTICO",
          token.value,
          `Token inesperado '${token.value}' después del cierre de la clase. No se permiten sentencias fuera de la clase`,
          token.line,
          token.column,
        ),
      )
    }

    return { errors: this.errors, python: this.pythonCode }
  }
//_____________________________________________________________________________
```

```
  //_____________________________________________________________________________
  //en caso de solo traducir un fragmento, se usa un método simple que tiene la traducción de las sentencias
    analizarSinClase() {
      // Procesar todas las sentencias directamente
      this.procesarSentencias()
      return { errors: this.errors, python: this.pythonCode }
    }
  //_____________________________________________________________________________
```

```
//_____________________________________________________________________________
//método para procesar sentencias de código por medio de un switch con varios casos de estructuras de control
procesarSentencias() {
  while (this.pos < this.tokens.length && this.tokens[this.pos].value !== "}") {
    const token = this.tokens[this.pos]

    // <CHANGE> Agregar manejo explícito de comentarios al inicio
    if (token.type === "COMMENT_LINE" || token.type === "COMMENT_BLOCK") {
      this.traducirComentario()
      continue
    }
```

```
    switch (token.type) {
      case "INT_TYPE":
      case "DOUBLE_TYPE":
      case "CHAR_TYPE":
      case "STRING_TYPE":
      case "BOOLEAN_TYPE":
        this.declaracionVariable()
        break
      case "IF":
        this.traducirIf()
        break
      case "FOR":
        this.traducirFor()
        break
      case "WHILE":
        this.traducirWhile()
        break
      case "SYSTEM":
        this.traducirPrint()
        break
      case "IDENTIFICADOR":
        this.traducirAsignacionOIncremento()
        break
      default:
        this.errors.push(
          new ErroresSin("SINTACTICO", token.value, `Token inesperado '${token.value}'. Se esperaba una declaración de variable, estructura de cont
        )
        this.pos++
        break
    }
  }
}
  //_____________________________________________________________________________________________
```

```
  //_____________________________________________________________________________________________
  //método encargado de validar la estrucura main de la clase
    validarMetodoMain() {
      const inicioMain = this.pos

      // Verificar "public"
      if (!this.tokens[this.pos] || this.tokens[this.pos].type !== "PUBLIC") {
        const token = this.tokens[this.pos]
        this.errors.push(
          new ErroresSin(
            "SINTACTICO",
            token?.value || "EOF",
            "Se esperaba 'public' al inicio del método main. La firma correcta es: public static void main(String[] args)",
            token?.line || 1,
            token?.column || 1,
          ),
        )
        return false
      }
      this.pos++

      // Verificar "static"
      if (!this.tokens[this.pos] || this.tokens[this.pos].type !== "STATIC") {
        const token = this.tokens[this.pos]
        this.errors.push(
          new ErroresSin(
            "SINTACTICO",
            token?.value || "EOF",
            "Se esperaba 'static' después de 'public' en el método main",
            token?.line || 1,
            token?.column || 1,
          ),
        )
```

```
289         return false
290       }
291       this.pos++
292
293       // Verificar "void"
294       if (!this.tokens[this.pos] || this.tokens[this.pos].type !== "VOID") {
295         const token = this.tokens[this.pos]
296         this.errors.push(
297           new ErroresSin(
298             "SINTACTICO",
299             token?.value || "EOF",
300             "Se esperaba 'void' después de 'static' en el método main",
301             token?.line || 1,
302             token?.column || 1,
303           ),
304         )
305         return false
306       }
307       this.pos++
308
309       // Verificar "main"
310       if (!this.tokens[this.pos] || this.tokens[this.pos].type !== "MAIN") {
311         const token = this.tokens[this.pos]
312         this.errors.push(
313           new ErroresSin(
314             "SINTACTICO",
315             token?.value || "EOF",
316             "Se esperaba 'main' después de 'void'",
317             token?.line || 1,
318             token?.column || 1,
319           ),
320         )
```

```
321         return false
322       }
323       this.pos++
324
325       // Verificar "("
326       if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "(") {
327         const token = this.tokens[this.pos]
328         this.errors.push(
329           new ErroresSin(
330             "SINTACTICO",
331             token?.value || "EOF",
332             "Se esperaba '(' después de 'main'",
333             token?.line || 1,
334             token?.column || 1,
335           ),
336         )
337         return false
338       }
339       this.pos++
340
341       // Verificar "String"
342       if (!this.tokens[this.pos] || this.tokens[this.pos].type !== "STRING_TYPE") {
343         const token = this.tokens[this.pos]
344         this.errors.push(
345           new ErroresSin(
346             "SINTACTICO",
347             token?.value || "EOF",
348             "Se esperaba 'String' en los parámetros del método main",
349             token?.line || 1,
350             token?.column || 1,
351           ),
352         )
```

```
353         return false
354       }
355       this.pos++
356
357       // Verificar "["
358       if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "[") {
359         const token = this.tokens[this.pos]
360         this.errors.push(
361           new ErroresSin(
362             "SINTACTICO",
363             token?.value || "EOF",
364             "Se esperaba '[' después de 'String' en el método main",
365             token?.line || 1,
366             token?.column || 1,
367           ),
368         )
369         return false
370       }
371       this.pos++
372
373       // Verificar "]"
374       if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "]") {
375         const token = this.tokens[this.pos]
376         this.errors.push(
377           new ErroresSin(
378             "SINTACTICO",
379             token?.value || "EOF",
380             "Se esperaba ']' después de '[' en el método main",
381             token?.line || 1,
382             token?.column || 1,
383           ),
384         )
```

```
385         return false
386       }
387       this.pos++
388
389       // Verificar "args" (identificador)
390       if (!this.tokens[this.pos] || this.tokens[this.pos].type !== "ARGS") {
391         const token = this.tokens[this.pos]
392         this.errors.push(
393           new ErroresSin(
394             "SINTACTICO",
395             token?.value || "EOF",
396             "Se esperaba 'args' como nombre del parámetro en el método main",
397             token?.line || 1,
398             token?.column || 1,
399           ),
400         )
401         return false
402       }
403       this.pos++
404
405       // Verificar ")"
406       if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ")") {
407         const token = this.tokens[this.pos]
408         this.errors.push(
409           new ErroresSin(
410             "SINTACTICO",
411             token?.value || "EOF",
412             "Se esperaba ')' para cerrar los parámetros del método main",
413             token?.line || 1,
414             token?.column || 1,
415           ),
```

```
416         )
417         return false
418       }
419       this.pos++
420
421       // Verificar "{"
422       if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "{") {
423         const token = this.tokens[this.pos]
424         this.errors.push(
425           new ErroresSin(
426             "SINTACTICO",
427             token?.value || "EOF",
428             "Se esperaba '{' para iniciar el cuerpo del método main",
429             token?.line || 1,
430             token?.column || 1,
431           ),
432         )
433         return false
434       }
435       this.pos++
436
437       return true
438     }
439   //_____________________________________________________
440
```

```
445  //_____________________________________________________________________
446  //clase encargada de traducir variables de tipo String
447    traducirString(valor, tipo) {
448      // Eliminar comillas si ya las tiene
449      let contenido = valor //el contenido va a ser el valor que almacena la variable de tipo string
450      if (contenido.startsWith('"') && contenido.endsWith('"')) {
451        contenido = contenido.slice(1, -1)
452      } else if (contenido.startsWith("'") && contenido.endsWith("'")) {
453        contenido = contenido.slice(1, -1)
454      }
455
456      // Procesar secuencias de escape carácter por carácter
457      let resultado = ""
458      let i = 0
459      while (i < contenido.length) {
460        if (contenido[i] === "\\" && i + 1 < contenido.length) {
461          // Detectar secuencia de escape
462          const siguienteChar = contenido[i + 1]
463          if (siguienteChar === "n") {
464            resultado += "\n" // Nueva línea
465            i += 2
466          } else if (siguienteChar === "t") {
467            resultado += "\t" // Tabulación
468            i += 2
469          } else if (siguienteChar === "r") {
470            resultado += "\r" // Retorno de carro
471            i += 2
472          } else if (siguienteChar === '"') {
473            resultado += '"' // Comilla doble escapada
474            i += 2
475          } else if (siguienteChar === "'") {
476            resultado += "'" // Comilla simple escapada
```

```
447    traducirString(valor, tipo) {
477            i += 2
478          } else if (siguienteChar === "\\") {
479            resultado += "\\" // Barra invertida escapada
480            i += 2
481          } else {
482            // Si no es una secuencia reconocida, mantener el carácter
483            resultado += contenido[i]
484            i++
485          }
486        } else {
487          // Carácter normal
488          resultado += contenido[i]
489          i++
490        }
491      }
492
493      // Retornar con el formato apropiado para Python
494      if (tipo === "CHAR") {
495        return `'${resultado}'` // Chars usan comillas simples
496      } else {
497        return `"${resultado}"` // Strings usan comillas dobles
498      }
499    }
500    //_____________________________________________________________________
501
```

```
505   //_____________________________________________________________________________
506   //método que traduce y analiza los tokens de las declaraciones de variables
507     declaracionVariable() {
508       const tipo = this.tokens[this.pos].type
509       this.pos++
510
511       // Obtener el identificador (nombre de la variable)
512       const id = this.tokens[this.pos]
513
514       if (!id || id.type !== "IDENTIFICADOR") {
515         this.errors.push(
516           new ErroresSin(
517             "SINTACTICO",
518             id?.value || "EOF",
519             "Se esperaba un identificador",
520             id?.line || 0,
521             id?.column || 0,
522           ),
523         )
524         return
525       }
526       this.pos++
527
528       // Verificar el signo de asignación
529       const igual = this.tokens[this.pos]
530       if (!igual || igual.value !== "=") {
531         this.errors.push(
532           new ErroresSin("SINTACTICO", igual?.value || "EOF", "Se esperaba '='", igual?.line || 0, igual?.column || 0),
533         )
534         return
535       }
536       this.pos++
537
538       let expresion = ""
539
540       while (this.pos < this.tokens.length && this.tokens[this.pos].value !== ";") {
541         const token = this.tokens[this.pos]
542
543         // Traducir valores booleanos
544         if (token.type === "TRUE") {
545           expresion += "True "
546         } else if (token.type === "FALSE") {
547           expresion += "False "
548         }
549         // Traducir strings y chars
550         else if (token.type === "STRING") {
551           expresion += this.traducirString(token.value, "STRING") + " "
552         } else if (token.type === "CHAR") {
553           expresion += this.traducirString(token.value, "CHAR") + " "
554         }
555         // Agregar cualquier otro token (números, identificadores, operadores)
556         else {
557           expresion += token.value + " "
558         }
559
560         this.pos++
561       }
```

```
562
563       // Verificar que se encontró la expresión
564       if (expresion.trim() === "") {
565         this.errors.push(new ErroresSin("SINTACTICO", "EOF", "Falta valor en asignación", id.line, id.column))
566         return
567       }
568
569       // Generación de la traducción de asignación de una variable
570       this.pythonCode += `${this.indent}${id.value} = ${expresion.trim()}\n`
571
572       // Verificar punto y coma final
573       const fin = this.tokens[this.pos]
574       if (fin && fin.value === ";") this.pos++
575       else this.errors.push(new ErroresSin("SINTACTICO", fin?.value || "EOF", "Se esperaba ';'", id.line, id.column))
576     }
577     //_____________________________________________________________________________
```

```
583   //_____________________________________________________________________________
584   //método encargado de traducir una estrucura de control IF
585     traducirIf() {
586       const ifToken = this.tokens[this.pos] //el token actual se almacenará en una variable llamada ifToken
587       this.pos++ //avanza a ' ( '
588
589       // Verificar paréntesis de apertura
590       if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "(") {
591         const currentToken = this.tokens[this.pos]
592         this.errors.push(
593           new ErroresSin("SINTACTICO",currentToken?.value || "EOF",
594               "Se esperaba '(' después de 'if'",
595               currentToken?.line || ifToken.line,
596               currentToken?.column || ifToken.column,),)
597         return
598         }
599       this.pos++
600       let condicion = ""
601
602       // Leer la condición hasta encontrar el paréntesis de cierre
603       while (this.pos < this.tokens.length && this.tokens[this.pos].value !== ")") {
604         const token = this.tokens[this.pos]
605
606         if (token.type === "CHAR") {
607           condicion += this.traducirString(token.value, "CHAR") + " "
608         } else if (token.type === "STRING") {
609           condicion += this.traducirString(token.value, "STRING") + " "
610         } else if (token.type === "TRUE") {
611           condicion += "True "
612         } else if (token.type === "FALSE") {
613           condicion += "False "
```

```
614         } else {
615           const val = token.value
616           // Evitar duplicar el operador ==
617           if (val === "=" && condicion.trim().endsWith("=")) {
618             this.pos++
619             continue
620           }
621           condicion += val + " "
622         }
623         this.pos++
624       }
625
626       // Verificar paréntesis de cierre
627       if (this.tokens[this.pos]?.value !== ")") {
628         const currentToken = this.tokens[this.pos]
629         this.errors.push(
630           new ErroresSin(
631             "SINTACTICO",
632             currentToken?.value || "EOF",
633             "Se esperaba ')' para cerrar la condición del if",
634             currentToken?.line || ifToken.line,
635             currentToken?.column || ifToken.column,
636           ),
637         )
638         return
639       }
640       this.pos++
641
642       // Generar código Python para el if
643       this.pythonCode += `${this.indent}if ${condicion.trim()}:\n`
```

```
644
645       // Verificar llave de apertura del bloque
646       if (this.tokens[this.pos]?.value !== "{") {
647         const currentToken = this.tokens[this.pos]
648         this.errors.push(
649           new ErroresSin("SINTACTICO",currentToken?.value || "EOF",
650               "Se esperaba '{' para iniciar el bloque del if",
651               currentToken?.line || ifToken.line,
652               currentToken?.column || ifToken.column,),)
653         return
654       }
655       this.pos++
656
657       // Incrementar indentación para el bloque
658       this.indent += "    "
659
660       // Procesar el contenido del bloque if
661       while (this.pos < this.tokens.length && this.tokens[this.pos].value !== "}") {
662         this.traducirLineaBloque()
663       }
664
665       // Restaurar indentación
666       this.indent = this.indent.slice(0, -4)
667
668       // Verificar llave de cierre del bloque if
669       if (this.tokens[this.pos]?.value !== "}") {
670         const currentToken = this.tokens[this.pos]
671         this.errors.push(new ErroresSin("SINTACTICO",currentToken?.value || "EOF",
672           "Se esperaba '}' para cerrar el bloque del if",
673           currentToken?.line || ifToken.line,
674           currentToken?.column || ifToken.column,),)
675         return
676       }
```

```
677       this.pos++ //avanza a un ' ELSE ' en caso de que venga
678
679       // Verificar si hay un bloque else
680       if (this.tokens[this.pos]?.type === "ELSE") {
681         const elseToken = this.tokens[this.pos]
682         this.pos++ //avanza a ' { '
683
684         // Verificar llave de apertura del else
685         if (this.tokens[this.pos]?.value !== "{") {
686           const currentToken = this.tokens[this.pos]
687           this.errors.push(
688             new ErroresSin("SINTACTICO",currentToken?.value || "EOF",
689               "Se esperaba '{' después de 'else'",
690               currentToken?.line || elseToken.line,
691               currentToken?.column || elseToken.column,),)
692           return
693         }
694
695         this.pos++
696         this.pythonCode += `${this.indent}else:\n`
697
698         // Incrementar indentación para el bloque else
699         this.indent += "    "
700
701         // Procesar el contenido del bloque else
702         while (this.pos < this.tokens.length && this.tokens[this.pos].value !== "}") { //mientras el token no sea un ' } ' se traduce el bloque
703           this.traducirLineaBloque()
704         }
705
706         // Restaurar indentación
707         this.indent = this.indent.slice(0, -4) //restaurando identación a 0 espacios
708
```

```
709         // Verificar llave de cierre del bloque else
710         if (this.tokens[this.pos]?.value === "}") this.pos++
711         else {
712           const currentToken = this.tokens[this.pos]
713           this.errors.push(new ErroresSin("SINTACTICO",currentToken?.value || "EOF","Se esperaba '}' para cerrar el bloque del else",currentToken?.li
714         }
715       }
716     }
717     //_____________________________________________________________________________________________
718
719
720
721
722   //_____________________________________________________________________________________________
723   //método el cual se encarga de traducir la estructura FOR
724       traducirFor() {
725       const forToken = this.tokens[this.pos]
726       this.pos++
727
728       if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "(") {
729         const currentToken = this.tokens[this.pos]
730         this.errors.push(
731           new ErroresSin(
732             "SINTACTICO",
733             currentToken?.value || "EOF",
734             "Se esperaba '('",
735             currentToken?.line || forToken.line,
736             currentToken?.column || forToken.column,
737           ),
738         )
739         return
740       }
741       this.pos++
```

```
741     this.pos++
742
743     const tipo = this.tokens[this.pos]
744     if (!tipo || (tipo.type !== "INT_TYPE" && tipo.type !== "DOUBLE_TYPE"))
745       this.errors.push(
746         new ErroresSin(
747           "SINTACTICO",
748           tipo?.value || "EOF",
749           "Se esperaba tipo de variable",
750           tipo?.line || forToken.line,
751           tipo?.column || forToken.column,
752         ),
753       )
754       return
755     }
756     this.pos++
757
758     const variable = this.tokens[this.pos]
759     if (!variable || variable.type !== "IDENTIFICADOR") {
760       this.errors.push(
761         new ErroresSin(
762           "SINTACTICO",
763           variable?.value || "EOF",
764           "Se esperaba un identificador",
765           variable?.line || forToken.line,
766           variable?.column || forToken.column,
767         ),
768       )
769       return
770     }
771     this.pos++
772
773     if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "=") {
774       const currentToken = this.tokens[this.pos]
```

```
775         this.errors.push(
776           new ErroresSin(
777             "SINTACTICO",
778             currentToken?.value || "EOF",
779             "Se esperaba '='",
780             currentToken?.line || forToken.line,
781             currentToken?.column || forToken.column,
782           ),
783         )
784         return
785       }
786       this.pos++
787
788       const inicio = this.tokens[this.pos]
789       if (!inicio) {
790         this.errors.push(
791           new ErroresSin("SINTACTICO", "EOF", "Se esperaba un valor inicial", forToken.line, forToken.column),
792         )
793         return
794       }
795       this.pos++
796
797       if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ";") {
798         const currentToken = this.tokens[this.pos]
799         this.errors.push(
800           new ErroresSin(
801             "SINTACTICO",
802             currentToken?.value || "EOF",
803             "Se esperaba ';'",
804             currentToken?.line || forToken.line,
805             currentToken?.column || forToken.column,
806           ),
807         )
808         return
```

```
809     }
810     this.pos++
811
812     const condVar = this.tokens[this.pos]
813     if (!condVar) {
814       this.errors.push(new ErroresSin("SINTACTICO", "EOF", "Se esperaba una variable", forToken.line, forToken.column))
815       return
816     }
817     this.pos++
818
819     const operador = this.tokens[this.pos]
820     if (!operador) {
821       this.errors.push(new ErroresSin("SINTACTICO", "EOF", "Se esperaba un operador", forToken.line, forToken.column))
822       return
823     }
824     this.pos++
825
826     const limite = this.tokens[this.pos]
827     if (!limite) {
828       this.errors.push(
829         new ErroresSin("SINTACTICO", "EOF", "Se esperaba un valor limite", forToken.line, forToken.column),
830       )
831       return
832     }
833     this.pos++
834
835     if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ";") {
836       const currentToken = this.tokens[this.pos]
837       this.errors.push(
838         new ErroresSin(
839           "SINTACTICO",
840           currentToken?.value || "EOF",
```

```
841           "Se esperaba ';'",
842           currentToken?.line || forToken.line,
843           currentToken?.column || forToken.column,
844         ),
845       )
846       return
847     }
848     this.pos++
849
850     const incVar = this.tokens[this.pos]
851     if (!incVar) {
852       this.errors.push(new ErroresSin("SINTACTICO", "EOF", "Se esperaba una variable", forToken.line, forToken.column))
853       return
854     }
855     this.pos++
856
857     if (this.tokens[this.pos]?.type === "INCREMENTO") this.pos++
858
859     if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ")") {
860       const currentToken = this.tokens[this.pos]
861       this.errors.push(
862         new ErroresSin(
863           "SINTACTICO",
864           currentToken?.value || "EOF",
865           "Se esperaba ')'",
866           currentToken?.line || forToken.line,
867           currentToken?.column || forToken.column,
868         ),
869       )
870       return
871     }
872     this.pos++
873
```

```
874     if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "{") {
875       const currentToken = this.tokens[this.pos]
876       this.errors.push(
877         new ErroresSin(
878           "SINTACTICO",
879           currentToken?.value || "EOF",
880           "Se esperaba '{'",
881           currentToken?.line || forToken.line,
882           currentToken?.column || forToken.column,
883         ),
884       )
885       return
886     }
887     this.pos++
888
889     this.pythonCode += `${this.indent}${variable.value} = ${inicio.value}\n`
890     this.pythonCode += `${this.indent}while ${condVar.value} ${operador.value} ${limite.value}:\n`
891     this.indent += "    "
892
893     while (this.pos < this.tokens.length && this.tokens[this.pos].value !== "}") {
894       this.traducirLineaBloque()
895     }
896
897     this.pythonCode += `${this.indent}${incVar.value} += 1\n`
898     this.indent = this.indent.slice(0, -4)
899
900     if (this.tokens[this.pos]?.value === "}") this.pos++
901     else {
902       const currentToken = this.tokens[this.pos]
903       this.errors.push(
904         new ErroresSin(
905           "SINTACTICO",
```

```
            currentToken?.value || "EOF",
            "Se esperaba '}'",
            currentToken?.line || forToken.line,
            currentToken?.column || forToken.column,
          ),
        )
      }
    }
    //_____________________________________________________________________________________________________________________


    //_____________________________________________________________________________________________________________________
    //método para traducir los valores de tipo string
     traducirPrint() {
      const systemToken = this.tokens[this.pos]
      this.pos++

      // Verificar la secuencia completa: System.out.println()
      if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ".") {
        const currentToken = this.tokens[this.pos]
        this.errors.push(
          new ErroresSin(
            "SINTACTICO",
            currentToken?.value || "EOF",
            "Se esperaba '.' después de 'System'",
            currentToken?.line || systemToken.line,
            currentToken?.column || systemToken.column,
          ),
        )
        return
      }
```

```
939         this.pos++
940
941         if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "out") {
942           const currentToken = this.tokens[this.pos]
943           this.errors.push(
944             new ErroresSin(
945               "SINTACTICO",
946               currentToken?.value || "EOF",
947               "Se esperaba 'out' después de 'System.'",
948               currentToken?.line || systemToken.line,
949               currentToken?.column || systemToken.column,
950             ),
951           )
952           return
953         }
954         this.pos++
955
956         if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ".") {
957           const currentToken = this.tokens[this.pos]
958           this.errors.push(
959             new ErroresSin(
960               "SINTACTICO",
961               currentToken?.value || "EOF",
962               "Se esperaba '.' después de 'System.out'",
963               currentToken?.line || systemToken.line,
964               currentToken?.column || systemToken.column,
965             ),
966           )
967           return
968         }
969         this.pos++
970
971         const println = this.tokens[this.pos]
972         if (!println || println.value !== "println") {
973           this.errors.push(
```

```
973       this.errors.push(
974         new ErroresSin(
975           "SINTACTICO",
976           println?.value || "EOF",
977           "Se esperaba 'println' después de 'System.out.'",
978           println?.line || systemToken.line,
979           println?.column || systemToken.column,
980         ),
981       )
982       return
983     }
984     this.pos++
985
986     // Verificar paréntesis de apertura
987     if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "(") {
988       const currentToken = this.tokens[this.pos]
989       this.errors.push(
990         new ErroresSin(
991           "SINTACTICO",
992           currentToken?.value || "EOF",
993           "Se esperaba '(' después de 'println'",
994           currentToken?.line || systemToken.line,
995           currentToken?.column || systemToken.column,
996         ),
997       )
998       return
999     }
1000    this.pos++
1001
1002    // Leer el contenido a imprimir
1003    let contenido = ""
1004    while (this.pos < this.tokens.length && this.tokens[this.pos].value !== ")") {
1005      const tok = this.tokens[this.pos]
1006
1007      // Traducir strings y chars correctamente
```

```
1007        // Traducir strings y chars correctamente
1008        if (tok.type === "STRING") {
1009          contenido += this.traducirString(tok.value, "STRING") + " "
1010        } else if (tok.type === "CHAR") {
1011          contenido += this.traducirString(tok.value, "CHAR") + " "
1012        } else {
1013          contenido += tok.value + " "
1014        }
1015        this.pos++
1016      }
1017
1018      // Verificar paréntesis de cierre
1019      if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ")") {
1020        const currentToken = this.tokens[this.pos]
1021        this.errors.push(
1022          new ErroresSin(
1023            "SINTACTICO",
1024            currentToken?.value || "EOF",
1025            "Se esperaba ')' para cerrar System.out.println",
1026            currentToken?.line || systemToken.line,
1027            currentToken?.column || systemToken.column,
1028          ),
1029        )
1030        return
1031      }
1032      this.pos++
1033
1034      if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ";") {
1035        const currentToken = this.tokens[this.pos]
1036        this.errors.push(
1037          new ErroresSin(
1038            "SINTACTICO",
1039            currentToken?.value || "EOF",
1040            "Se esperaba ';' después de System.out.println()",
```

```
1041            currentToken?.line || systemToken.line,
1042            currentToken?.column || systemToken.column,
1043          ),
1044        )
1045        return
1046      }
1047      this.pos++
1048
1049      // Generar código Python
1050      this.pythonCode += `${this.indent}print(${contenido.trim()})\n`
1051    }
1052
1053
1054
1055
1056
1057
1058  //_____________________________________________________________________________
1059  //método que traduce estructuras while
1060   traducirWhile() {
1061      const whileToken = this.tokens[this.pos]
1062      this.pos++
1063
1064      // Verificar paréntesis de apertura
1065      if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "(") {
1066        const currentToken = this.tokens[this.pos]
1067        this.errors.push(
1068          new ErroresSin(
1069            "SINTACTICO",
1070            currentToken?.value || "EOF",
1071            "Se esperaba '(' después de 'while'",
1072            currentToken?.line || whileToken.line,
1073            currentToken?.column || whileToken.column,),)
```

```
1074            return
1075          }
1076          this.pos++
1077
1078          // Leer la condición hasta encontrar el paréntesis de cierre
1079          let condicion = ""
1080          while (this.pos < this.tokens.length && this.tokens[this.pos].value !== ")") {
1081            const token = this.tokens[this.pos]
1082
1083            // Traducir booleanos
1084            if (token.type === "TRUE") {
1085              condicion += "True "
1086            } else if (token.type === "FALSE") {
1087              condicion += "False "
1088            } else {
1089              condicion += token.value + " "
1090            }
1091            this.pos++
1092          }
1093
1094          // Verificar paréntesis de cierre
1095          if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ")") {
1096            const currentToken = this.tokens[this.pos]
1097            this.errors.push(
1098              new ErroresSin(
1099                "SINTACTICO",
1100                currentToken?.value || "EOF",
1101                "Se esperaba ')' para cerrar la condición del while",
1102                currentToken?.line || whileToken.line,
1103                currentToken?.column || whileToken.column,
1104              ),
```

```
1105        )
1106        return
1107      }
1108      this.pos++
1109
1110      // Generar código Python para el while
1111      this.pythonCode += `${this.indent}while ${condicion.trim()}:\n`
1112
1113      // Verificar llave de apertura del bloque
1114      if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "{") {
1115        const currentToken = this.tokens[this.pos]
1116        this.errors.push(
1117          new ErroresSin(
1118            "SINTACTICO",
1119            currentToken?.value || "EOF",
1120            "Se esperaba '{' para iniciar el bloque del while",
1121            currentToken?.line || whileToken.line,
1122            currentToken?.column || whileToken.column,
1123          ),
1124        )
1125        return
1126      }
1127      this.pos++
1128
1129      // Incrementar indentación para el bloque
1130      this.indent += "    "
1131
1132      // Procesar el contenido del bloque while
1133      while (this.pos < this.tokens.length && this.tokens[this.pos].value !== "}") {
1134        this.traducirLineaBloque()
1135      }
1136
1137      // Restaurar indentación
1138      this.indent = this.indent.slice(0, -4)
```

```
1139
1140      // Verificar llave de cierre del bloque while
1141      if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "}") {
1142        const currentToken = this.tokens[this.pos]
1143        this.errors.push(
1144          new ErroresSin(
1145            "SINTACTICO",
1146            currentToken?.value || "EOF",
1147            "Se esperaba '}' para cerrar el bloque del while",
1148            currentToken?.line || whileToken.line,
1149            currentToken?.column || whileToken.column,
1150          ),
1151        )
1152        return
1153      }
1154      this.pos++
1155    }
1156  //_____________________________________________________________________
1157
1158
1159
1160
1161
1162  //_____________________________________________________________________
1163  //método que traduce asignaciones o incrementos
1164   traducirAsignacionOIncremento() {
1165      const identificador = this.tokens[this.pos]
1166      this.pos++
1167
1168      // Verificar si es un incremento (++)
1169      if (this.tokens[this.pos]?.type === "INCREMENTO") {
```

```
1170        this.pos++
1171        this.pythonCode += `${this.indent}${identificador.value} += 1\n`
1172
1173        if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ";") {
1174          const currentToken = this.tokens[this.pos]
1175          this.errors.push(
1176            new ErroresSin(
1177              "SINTACTICO",
1178              currentToken?.value || "EOF",
1179              `Se esperaba ';' después de '${identificador.value}++'`,
1180              currentToken?.line || identificador.line,
1181              currentToken?.column || identificador.column,
1182            ),
1183          )
1184          return
1185        }
1186        this.pos++
1187        return
1188      }
1189
1190      // Verificar si es un decremento (--)
1191      if (this.tokens[this.pos]?.type === "DECREMENTO") {
1192        this.pos++
1193        this.pythonCode += `${this.indent}${identificador.value} -= 1\n`
1194
1195        if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ";") {
1196          const currentToken = this.tokens[this.pos]
1197          this.errors.push(
1198            new ErroresSin(
1199              "SINTACTICO",
1200              currentToken?.value || "EOF",
1201              `Se esperaba ';' después de '${identificador.value}--'`,
```

```
1202                currentToken?.line || identificador.line,
1203                currentToken?.column || identificador.column,
1204              ),
1205            )
1206            return
1207          }
1208          this.pos++
1209          return
1210        }
1211
1212        // Verificar si es una asignación (=)
1213        if (this.tokens[this.pos]?.value === "=") {
1214          this.pos++
1215
1216          // Leer la expresión hasta el punto y coma
1217          let expresion = ""
1218          while (this.pos < this.tokens.length && this.tokens[this.pos].value !== ";") {
1219            const token = this.tokens[this.pos]
1220
1221            // Traducir valores booleanos
1222            if (token.type === "TRUE") {
1223              expresion += "True "
1224            } else if (token.type === "FALSE") {
1225              expresion += "False "
1226            }
1227            // Traducir strings y chars
1228            else if (token.type === "STRING") {
1229              expresion += this.traducirString(token.value, "STRING") + " "
1230            } else if (token.type === "CHAR") {
1231              expresion += this.traducirString(token.value, "CHAR") + " "
1232            }
1233            // Agregar cualquier otro token
```

```
1234            else {
1235              expresion += token.value + " "
1236            }
1237
1238            this.pos++
1239          }
1240
1241          // Generar código Python
1242          this.pythonCode += `${this.indent}${identificador.value} = ${expresion.trim()}\n`
1243
1244          if (!this.tokens[this.pos] || this.tokens[this.pos].value !== ";") {
1245            const currentToken = this.tokens[this.pos]
1246            this.errors.push(
1247              new ErroresSin(
1248                "SINTACTICO",
1249                currentToken?.value || "EOF",
1250                `Se esperaba ';' después de la asignación '${identificador.value} = ...'`,
1251                currentToken?.line || identificador.line,
1252                currentToken?.column || identificador.column,
1253              ),
1254            )
1255            return
1256          }
1257          this.pos++
1258          return
1259        }
1260
1261        // Si no es ninguno de los casos anteriores, es un error
1262        const currentToken = this.tokens[this.pos]
1263        this.errors.push(
1264          new ErroresSin(
1265            "SINTACTICO",
1266            currentToken?.value || "EOF",
1267            `Se esperaba '=', '++' o '--' después del identificador '${identificador.value}'`,
1268            currentToken?.line || identificador.line,
```

```
1269            currentToken?.column || identificador.column,
1270          ),
1271        )
1272      }
1273      //_____________________________________________________________________________
1274
1275
1276
1277
1278
1279    //_____________________________________________________________________________
1280    //método que se encarga de traducir los comentarios
1281      traducirComentario() {
1282      const comentarioToken = this.tokens[this.pos]
1283
1284      if (!comentarioToken) {
1285        return
1286      }
1287
1288      let contenido = comentarioToken.value
1289
1290      // Comentario de línea: // texto → # texto
1291      if (contenido.startsWith("//")) {
1292        contenido = contenido.substring(2).trim()
1293        this.pythonCode += `${this.indent}# ${contenido}\n`
1294      }
1295      // Comentario de bloque: /* texto */ → ''' texto '''
1296      else if (contenido.startsWith("/*") && contenido.endsWith("*/")) {
1297        contenido = contenido.substring(2, contenido.length - 2).trim()
```

```
1297        contenido = contenido.substring(2, contenido.length - 2).trim()
1298        this.pythonCode += `${this.indent}''' ${contenido} '''\n`
1299      }
1300
1301      this.pos++
1302    }
1303    //_____________________________________________________________________
1304
1305
1306
1307      traducirLineaBloque() {
1308        const actual = this.tokens[this.pos]
1309
1310        switch (actual.type) {
1311          case "INT_TYPE":
1312          case "DOUBLE_TYPE":
1313          case "CHAR_TYPE":
1314          case "STRING_TYPE":
1315          case "BOOLEAN_TYPE":
1316            this.declaracionVariable()
1317            break
1318          case "IF":
1319            this.traducirIf()
1320            break
1321          case "FOR":
1322            this.traducirFor()
1323            break
1324          case "WHILE":
1325            this.traducirWhile()
1326            break
1327          case "SYSTEM":
1328            this.traducirPrint()
1329            break
```

```
1330            case "IDENTIFICADOR":
1331              this.traducirAsignacionOIncremento()
1332              break
1333            default:
1334              // Token no reconocido, avanzar
1335              this.pos++
1336              break
1337          }
1338        }
1339      }
1340
1341      // Exponer la clase Parser globalmente
1342      window.Parser = Parser
1343
```

Clase Token

Ruta: ../Token/Token.js

Esta clase es parecida a la de los tipos de errores, solamente inicializa datos de los tokens

```
1   class Token{
2       constructor(tipo, valor, linea, columna){
3           this.tipo = tipo;
4           this.valor = valor;
5           this.linea = linea;
6           this.columna = columna;
7       }
8   }
9
10
11  export { Token }
```

Clase main

Ruta: ../main.js

Esta clase es la encargada del dinamismo en el navegador, ya que genera las listas de tokens, de errores, abre archivos, descarga reportes, etc. Entonces tiene un rol fundamental dentro del programa:

```
let tokens = [] // Tokens generados por el lexer

let erroresLexicos = [] // Errores léxicos encontrados

let erroresSintacticos = [] // Errores sintácticos encontrados

let codigoPython = "" // Código Python traducido


//_____________________________________________________________________________

//referenciando elementos del DOM

// Áreas de texto

const javaCodeArea = document.getElementById("java-code")

const pythonCodeArea = document.getElementById("python-code")


// Botones de acción

const translateBtn = document.getElementById("translate")

const viewTokensBtn = document.getElementById("view-tokens-btn")

const clearAllBtn = document.getElementById("clear-all-btn")

const openJavaBtn = document.getElementById("open-java-btn")

const saveJavaBtn = document.getElementById("save-java-btn")

const savePythonBtn = document.getElementById("save-python-btn")

const reporteTokensBtn = document.getElementById("reporte-tokens-btn")

const reporteLexicosBtn = document.getElementById("reporte-lexicos-btn")

const reporteSintacticosBtn = document.getElementById("reporte-sintacticos-btn")


// Elementos de estado
```

```
const statusMessage = document.getElementById("status-message")
const tokensCount = document.getElementById("tokens-count")

// Cuerpos de las tablas
const tokensBody = document.getElementById("tokens-body")
const lexicalErrorsBody = document.getElementById("lexical-errors-body")
const syntaxErrorsBody = document.getElementById("syntax-errors-body")

// Pestañas
const tabs = document.querySelectorAll(".tab")
const tabContents = document.querySelectorAll(".tab-content")
console.log(" main.js cargado correctamente")
//_____
___




//_____
// REGISTRO DE EVENT LISTENERS

// Botones principales
translateBtn.addEventListener("click", traducirCodigo)
viewTokensBtn.addEventListener("click", mostrarPestanaTokens)

// Event listener para abrir Java
if (openJavaBtn) {
  openJavaBtn.addEventListener("click", abrirArchivoJava)
  console.log(" Event listener para abrir Java registrado")
}
```

```
if (saveJavaBtn) {
  saveJavaBtn.addEventListener("click", guardarCodigoJava)
  console.log(" Event listener para guardar Java registrado")
}

if (savePythonBtn) {
  savePythonBtn.addEventListener("click", guardarCodigoPython)
  console.log(" Event listener para guardar Python registrado")
}

if (reporteTokensBtn) {
  reporteTokensBtn.addEventListener("click", generarReporteTokens)
  console.log(" Event listener para reporte tokens registrado")
}

if (reporteLexicosBtn) {
  reporteLexicosBtn.addEventListener("click", generarReporteErroresLexicos)
  console.log(" Event listener para reporte léxicos registrado")
}

if (reporteSintacticosBtn) {
  reporteSintacticosBtn.addEventListener("click", generarReporteErroresSintacticos)
  console.log(" Event listener para reporte sintácticos registrado")
}
if (clearAllBtn) {
  clearAllBtn.addEventListener("click", limpiarTodo)
}

// Pestañas de reportes
tabs.forEach((tab) => {
```

```
  tab.addEventListener("click", () => { //haciendo una acción al realizar un click en la pestaña
    const tabName = tab.getAttribute("data-tab")
    cambiarPestana(tabName)
  })
})

console.log(" Event listeners registrados")
//_____________________________________________________________________________________
_____________________


//_____________________________________________________________________________________
_____________________________________________
// FUNCIÓN PRINCIPAL DE TRADUCCIÓN
function traducirCodigo() {
  console.log(" Ejecutando traducción...")

  const codigo = javaCodeArea.value

  // Validar que haya código para traducir
  if (!codigo.trim()) {
    alert("Por favor, ingrese código Java para traducir")
    return
  }

  // Actualizar estado de la interfaz
  statusMessage.textContent = "Analizando..."
```

```
  try {
    //analizando primeramente la parte léxica por medio de una instancia
    console.log(" Iniciando análisis léxico...")
    const lexer = window.lexer //importando lexer.js
    const analizadorLexico = new lexer(codigo) //instancia
    tokens = analizadorLexico.analizar() //analizando tokens por medio de la instancia
    erroresLexicos = analizadorLexico.error || [] //retornando errores o un arreglo vacío en caso de no haber

    console.log(" Tokens encontrados:", tokens ? tokens.length : 0)
    console.log(" Errores léxicos:", erroresLexicos ? erroresLexicos.length : 0)

    if (!tokens || !Array.isArray(tokens)) { //si tokens es inválido o no existe, devuelve un array vacío
      tokens = []
    }
    if (!erroresLexicos || !Array.isArray(erroresLexicos)) {
      erroresLexicos = []
    }

    // Actualizar contador de tokens en la interfaz
    tokensCount.textContent = tokens.length

    // Actualizar tablas de tokens y errores léxicos
    actualizarTablaTokens()
    actualizarTablaErroresLexicos()
//_____________________________________________________________________________________
_____________________________________


//_____________________________________________________________________________________
_____________________________________
```

```
// PASO 2: ANÁLISIS SINTÁCTICO Y TRADUCCIÓN
    console.log(" Iniciando análisis sintáctico...")


    const Parser = window.Parser
      const analizadorSintactico = new Parser(tokens) //instanciando el parser por medio de una objeto analizadorSintactico
    const resultado = analizadorSintactico.analizar()


    erroresSintacticos = resultado.errors || [] //retornando los errores o un arreglo vacío
    codigoPython = resultado.python || "" //retornando el código de salida o una expresión vacía


    console.log(" Errores sintácticos:", erroresSintacticos.length)
    console.log(" Código Python generado:", codigoPython.length, "caracteres")


    // Actualizar tabla de errores sintácticos
    actualizarTablaErroresSintacticos()


    const totalErrores = erroresLexicos.length + erroresSintacticos.length


    if (totalErrores > 0) {
      let mensajeError = ""
      let mensajeAlerta = "Se encontraron errores:\n"


      if (erroresLexicos.length > 0 && erroresSintacticos.length > 0) {
          mensajeError = `Hay ${erroresLexicos.length} errores léxicos y ${erroresSintacticos.length} errores sintácticos. Revisa las tablas de errores.`
          mensajeAlerta += `- ${erroresLexicos.length} errores léxicos\n- ${erroresSintacticos.length} errores sintácticos\n\nRevise las pestañas de errores.`
        statusMessage.textContent = "Error: Errores léxicos y sintácticos detectados" //mostrando en el txtArea
      } else if (erroresLexicos.length > 0) { //en caso de haber solamente errores léxicos
        mensajeError = `Hay ${erroresLexicos.length} errores léxicos. Revisa la tabla de errores léxicos.`
```

```
        mensajeAlerta += `- ${erroresLexicos.length} errores léxicos\n\nRevise la pestaña de errores léxicos.`
        statusMessage.textContent = "Error: Errores léxicos detectados"
      } else { //de lo contrario, hay errores sintácticos
        mensajeError = `Hay ${erroresSintacticos.length} errores sintácticos. Revisa la tabla de errores sintácticos.`
          mensajeAlerta += `- ${erroresSintacticos.length} errores sintácticos\n\nRevise la pestaña de errores sintácticos.`
        statusMessage.textContent = "Error: Errores sintácticos detectados"
      }

      pythonCodeArea.value = mensajeError //el código en python retornará el mensaje de error
      alert(mensajeAlerta) //mostrando un mensaje de alerta
      return //frenando la acción para no caer en un bucle infinito
    }
//_____________________________________________________________________________
_____________________________________________

//_____________________________________________________________________________
_____________________
//MOSTRAR CÓDIGO TRADUCIDO

    pythonCodeArea.value = codigoPython //retornando las traducciones
    statusMessage.textContent = "Traducción exitosa" //mensaje que demustra traduccióne exitosa

    alert(`¡Traducción exitosa!\nTokens analizados: ${tokens.length}`)
  } catch (error) {
    // Manejo de errores del sistema
    console.error(" Error en traducción:", error)
    pythonCodeArea.value = "# Error del sistema: " + error.message
    statusMessage.textContent = "Error del sistema"
    alert("Error del sistema: " + error.message)
```

```
  }
//_____________________________________________________________________________
_____________________

} //fin del método principal de traducción
//_____________________________________________________________________________
_____________________________________

//_____________________________________________________________________________
_____________________________________
// FUNCIONES DE ACTUALIZACIÓN DE TABLAS

//_____________________________________________________________________________
_____________________________________
//función para limpiar contenido
function limpiarTodo() {
  // Limpiar textareas
  javaCodeArea.value = ""
  pythonCodeArea.value = ""

  // Limpiar arrays de datos
  tokens = []
  erroresLexicos = []
  erroresSintacticos = []

  // Limpiar tablas
  actualizarTablaTokens()
```

```
  actualizarTablaErroresLexicos()
  actualizarTablaErroresSintacticos()

  // Resetear mensaje de estado
  statusMessage.textContent = "Listo para traducir"
  statusMessage.style.color = "#666"

  // Volver a la pestaña de tokens
  document.querySelectorAll(".tab-btn").forEach(btn => btn.classList.remove("active"))
  document.querySelectorAll(".tab-content").forEach(content => content.classList.remove("active"))
  document.querySelector(".tab-btn").classList.add("active")
  document.getElementById("tokens-tab").classList.add("active")

  console.log("Aplicación limpiada completamente")
}
//_____________________________________________________________________________________
_____________________________________




//función que actualiza la tabla de tokens
//_____________________________________________________________________________________
_____________________________________
function actualizarTablaTokens() {
//sentencia cuando no hay tokens se muestra un mensaje alineado diciendo que no se han generado los tokens
  if (tokens.length === 0) {
      tokensBody.innerHTML = '<tr><td colspan="5" style="text-align: center;">No se han generado tokens</td></tr>'
    return
  }
```

```
  let html = ""
  tokens.forEach((token, index) => {
    //creando las filas de la tabla de tokens
    html += `
      <tr>
        <td>${index + 1}</td>
        <td><code>${token.value}</code></td>
        <td>${token.type}</td>
        <td>${token.line}</td>
        <td>${token.column}</td>
      </tr>
    `
  })
  tokensBody.innerHTML = html //insertando los tokens en la tabla generada
} //fin de actualizarTablaTokens
//_____________________________________________________________________________________
_____________________________________


//_____________________________________________________________________________________
_____________________________________
//funcion que actualiza la tabla de Errores Léxicos (mismo funcionamiento que el método para actualizar la tabla de tokens)
function actualizarTablaErroresLexicos() {
  if (erroresLexicos.length === 0) {
      lexicalErrorsBody.innerHTML = '<tr><td colspan="5" style="text-align: center;">No hay errores léxicos</td></tr>'
    return
  }

  let html = ""
  erroresLexicos.forEach((error, index) => {
```

```
  html += `
    <tr class="error-row">
      <td>${index + 1}</td>
      <td>${error.tipo}</td>
      <td>${error.descripcion}</td>
      <td>${error.linea}</td>
      <td>${error.columna}</td>
    </tr>
  `
 })
 lexicalErrorsBody.innerHTML = html
}
//_____________________________________________________________________________
_____________________________________


//_____________________________________________________________________________
_____________________________________
//funcion que actualiza la tabla de Errores Sintacticos (mismo funcionamiento que el método para actualizar la tabla de tokens)
function actualizarTablaErroresSintacticos() {
 if (erroresSintacticos.length === 0) {
      syntaxErrorsBody.innerHTML = '<tr><td colspan="5" style="text-align: center;">No hay errores sintácticos</td></tr>'
   return
 }

 let html = ""
 erroresSintacticos.forEach((error, index) => {
   html += `
```

```
      <tr class="error-row">
        <td>${index + 1}</td>
        <td>${error.tipo}</td>
        <td>${error.mensaje}</td>
        <td>${error.linea}</td>
        <td>${error.columna}</td>
      </tr>
    `
  })
  syntaxErrorsBody.innerHTML = html
}
//_____
_____


//_____
_____
// FUNCIONES DE NAVEGACIÓN DE PESTAÑAS
function cambiarPestana(tabName) {
  // Remover clase active de todas las pestañas
  tabs.forEach((tab) => tab.classList.remove("active"))
  tabContents.forEach((content) => content.classList.remove("active"))

  // Agregar clase active a la pestaña seleccionada
  const selectedTab = document.querySelector(`[data-tab="${tabName}"]`)
  const selectedContent = document.getElementById(`${tabName}-tab`)

  if (selectedTab && selectedContent) {
    selectedTab.classList.add("active")
```

```
    selectedContent.classList.add("active")
  }
}
//_____________________________________________________________________________
_____________________________________________


//_____________________________________________________________________________
_____________________________________________
function mostrarPestanaTokens() {
  // Verificar que haya datos para mostrar
  if (tokens.length === 0 && erroresLexicos.length === 0 && erroresSintacticos.length === 0) {
    alert("Primero debe generar la traducción")
    return
  }

  // Determinar qué pestaña mostrar según los errores encontrados
  if (erroresLexicos.length > 0) {
    cambiarPestana("lexical-errors")
  } else if (erroresSintacticos.length > 0) {
    cambiarPestana("syntax-errors")
  } else {
    cambiarPestana("tokens")
  }

  // Hacer scroll hacia la sección de reportes
  document.querySelector(".report-container").scrollIntoView({ behavior: "smooth" })
}
//_____________________________________________________________________________
_____________________________________________
```

```
//_____
_____
//funciones para el manejo de archivos de las traducciones y los reportes
function abrirArchivoJava() {
  console.log(" Abriendo archivo Java...")

  // Crear input file temporal
  const input = document.createElement("input")
  input.type = "file"
  input.accept = ".java"

  input.onchange = (e) => {
    const file = e.target.files[0] //obteniendo el primer archivo seleccionado
    if (!file) return //si no hay archivo, se retorna nulo

    const reader = new FileReader() //api para leer archivos del sistema
    reader.onload = (event) => { //evento de lectura del archivo por medio de la variable reader
      javaCodeArea.value = event.target.result //mostrando el código en el área de código JAVA
      statusMessage.textContent = `Archivo "${file.name}" cargado`
      console.log(" Archivo Java cargado:", file.name) //mensaje que se cargó correctamente
    }
    reader.onerror = () => {
      alert("Error al leer el archivo")
      console.error(" Error al leer archivo") //mensaje de error si no se lee el archivo
    }
    reader.readAsText(file)
  }
```

```
  input.click()
}
//_____-
```

```
//_____
```

```
//acción para guardar el código de JAVA
function guardarCodigoJava() {
  console.log(" Guardando código Java...")

  if (!javaCodeArea.value.trim()) {
    alert("No hay código Java para guardar")
    return
  }

  const nombreArchivo = prompt("Ingrese el nombre del archivo (sin extensión):", "codigo")
  if (!nombreArchivo) return

  const blob = new Blob([javaCodeArea.value], { type: "text/plain" })
  const url = URL.createObjectURL(blob)
  const a = document.createElement("a")
  a.href = url
  a.download = `${nombreArchivo}.java`
  a.click()
  URL.revokeObjectURL(url)

  statusMessage.textContent = `Archivo "${nombreArchivo}.java" guardado`
```

```
  console.log(" Archivo Java guardado:", nombreArchivo)
}
//_____
_____

//_____
_____
//Guardar el código de Python traducido
function guardarCodigoPython() {
  console.log(" Guardando código Python...")

  if (!pythonCodeArea.value.trim()) {
    alert("No hay código Python para guardar. Primero genere la traducción.") //validando si no hay código para traducir
    return
  }

  const nombreArchivo = prompt("Ingrese el nombre del archivo (sin extensión):", "traducido")
  if (!nombreArchivo) return

  const blob = new Blob([pythonCodeArea.value], { type: "text/plain" })
  const url = URL.createObjectURL(blob)
  const a = document.createElement("a")
  a.href = url
  a.download = `${nombreArchivo}.py`
  a.click()
  URL.revokeObjectURL(url)

  statusMessage.textContent = `Archivo "${nombreArchivo}.py" guardado`
```

```
  console.log(" Archivo Python guardado:", nombreArchivo)
}
//_____________________________________________________________________________
_____________________


//_____________________________________________________________________________
_____________________
//funciones para generar los reportes html

//reporte de Tokens
function generarReporteTokens() {
  console.log(" Generando reporte de tokens...")

  if (tokens.length === 0) {
    alert("No hay tokens para generar el reporte. Primero genere la traducción.")
    return
  }

  let html = `
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Reporte de Tokens</title>
</head>
<body>
    <h1>Reporte de Tokens</h1>
    <p>Total de tokens: ${tokens.length}</p>
    <table border="1" cellpadding="5" cellspacing="0">
```

```
        <thead>
            <tr>
                <th>No.</th>
                <th>Lexema</th>
                <th>Tipo</th>
                <th>Línea</th>
                <th>Columna</th>
            </tr>
        </thead>
        <tbody>
`

  tokens.forEach((token, index) => {
    html += `
            <tr>
                <td>${index + 1}</td>
                <td>${token.value}</td>
                <td>${token.type}</td>
                <td>${token.line}</td>
                <td>${token.column}</td>
            </tr>
`
  })

  html += `
        </tbody>
    </table>
</body>
</html>
`
```

```
  descargarReporteHTML(html, "reporte_tokens.html")
  console.log(" Reporte de tokens generado")
}
//_____________________________________________________________________________
_______



//_____________________________________________________________________________
_______
//generación de reporte de errores léxicos
function generarReporteErroresLexicos() {
  console.log(" Generando reporte de errores léxicos...")

  if (erroresLexicos.length === 0) {
    alert("No hay errores léxicos para generar el reporte.")
    return
  }

  let html = `
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Reporte de Errores Léxicos</title>
</head>
<body>
    <h1>Reporte de Errores Léxicos</h1>
    <p>Total de errores: ${erroresLexicos.length}</p>
```

```
    <table border="1" cellpadding="5" cellspacing="0">
        <thead>
            <tr>
                <th>No.</th>
                <th>Tipo</th>
                <th>Descripción</th>
                <th>Línea</th>
                <th>Columna</th>
            </tr>
        </thead>
        <tbody>
`


  erroresLexicos.forEach((error, index) => {
    html += `
            <tr>
                <td>${index + 1}</td>
                <td>${error.tipo}</td>
                <td>${error.descripcion}</td>
                <td>${error.linea}</td>
                <td>${error.columna}</td>
            </tr>
`
  })

  html += `
        </tbody>
    </table>
</body>
</html>
```

```
`

  descargarReporteHTML(html, "reporte_errores_lexicos.html")
  console.log(" Reporte de errores léxicos generado")
}
//_____
_______

//_____
_______
//generación de reportes de errores sintácticos
function generarReporteErroresSintacticos() {
  console.log(" Generando reporte de errores sintácticos...")

  if (erroresSintacticos.length === 0) {
    alert("No hay errores sintácticos para generar el reporte.")
    return
  }

  let html = `
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Reporte de Errores Sintácticos</title>
</head>
<body>
```

```
    <h1>Reporte de Errores Sintácticos</h1>
    <p>Total de errores: ${erroresSintacticos.length}</p>
    <table border="1" cellpadding="5" cellspacing="0">
        <thead>
            <tr>
                <th>No.</th>
                <th>Tipo</th>
                <th>Mensaje</th>
                <th>Línea</th>
                <th>Columna</th>
            </tr>
        </thead>
        <tbody>
`


    erroresSintacticos.forEach((error, index) => {
        html += `
            <tr>
                <td>${index + 1}</td>
                <td>${error.tipo}</td>
                <td>${error.mensaje}</td>
                <td>${error.linea}</td>
                <td>${error.columna}</td>
            </tr>
`
    })

    html += `
        </tbody>
    </table>
```

```
</body>
</html>
`

  descargarReporteHTML(html, "reporte_errores_sintacticos.html")
  console.log(" Reporte de errores sintácticos generado")
}
//_____
_____



//_____
_____
//funcion para descargar los reportes
function descargarReporteHTML(html, nombreArchivo) {
  console.log(" Descargando reporte:", nombreArchivo)

  const blob = new Blob([html], { type: "text/html" })
  const url = URL.createObjectURL(blob)
  const a = document.createElement("a")
  a.href = url
  a.download = nombreArchivo
  a.click()
  URL.revokeObjectURL(url)

  statusMessage.textContent = `Reporte "${nombreArchivo}" generado`
}

//_____
_____
```

console.log(" Todas las funciones cargadas correctamente")

Recomendaciones

- Usar un manejo de errores adecuado como se implementó en este código
- Trabajar modularmente las responsabilidades en caso de querer hacer una remodelación furura
-