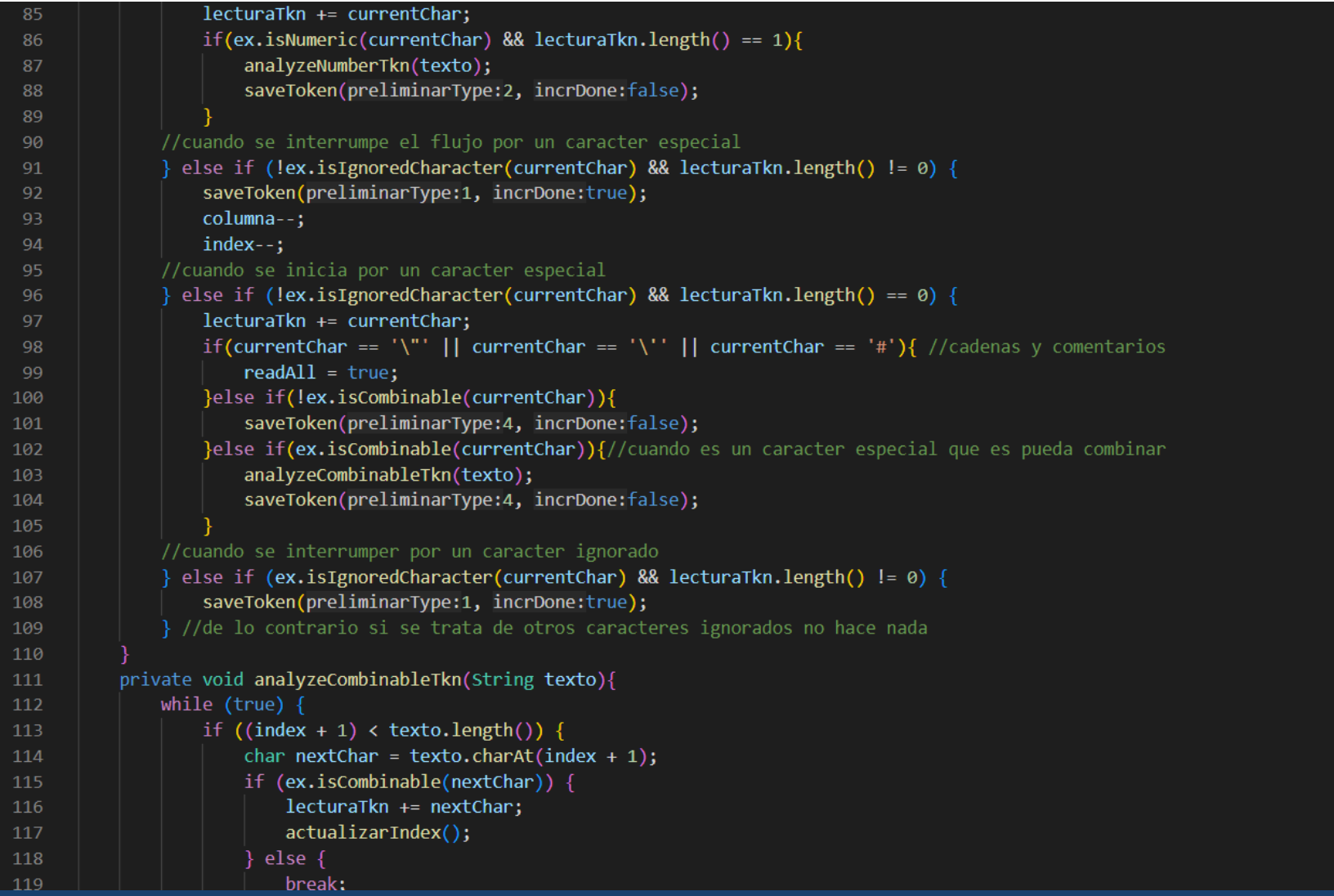


```
85              lecturaTkn += currentChar;
86              if(ex.isNumeric(currentChar) && lecturaTkn.length() == 1){
87                  analyzeNumberTkn(texto);
88                  saveToken(preliminarType:2, incrDone:false);
89              }
90          //cuando se interrumpe el flujo por un caracter especial
91          } else if (!ex.isIgnoredCharacter(currentChar) && lecturaTkn.length() != 0) {
92              saveToken(preliminarType:1, incrDone:true);
93              columna--;
94              index--;
95          //cuando se inicia por un caracter especial
96          } else if (!ex.isIgnoredCharacter(currentChar) && lecturaTkn.length() == 0) {
97              lecturaTkn += currentChar;
98              if(currentChar == '\"' || currentChar == '\'' || currentChar == '#'){ //cadenas y comentarios
99                  readAll = true;
100             }else if(!ex.isCombinable(currentChar)){
101                 saveToken(preliminarType:4, incrDone:false);
102             }else if(ex.isCombinable(currentChar)){//cuando es un caracter especial que es pueda combinar
103                 analyzeCombinableTkn(texto);
104                 saveToken(preliminarType:4, incrDone:false);
105             }
106         //cuando se interrumper por un caracter ignorado
107         } else if (ex.isIgnoredCharacter(currentChar) && lecturaTkn.length() != 0) {
108             saveToken(preliminarType:1, incrDone:true);
109         } //de lo contrario si se trata de otros caracteres ignorados no hace nada
110     }
111     private void analyzeCombinableTkn(String texto){
112         while (true) {
113             if ((index + 1) < texto.length()) {
114                 char nextChar = texto.charAt(index + 1);
115                 if (ex.isCombinable(nextChar)) {
116                     lecturaTkn += nextChar;
117                     actualizarIndex();
118                 } else {
119                     break;
```

# Manual de Usuario

# Parser-pY


Yennifer María de León Samuc

Registro Académico No. 202231084


Lenguajes Formales y de programación, segundo semestre 2023

# INDICE

## Contenido

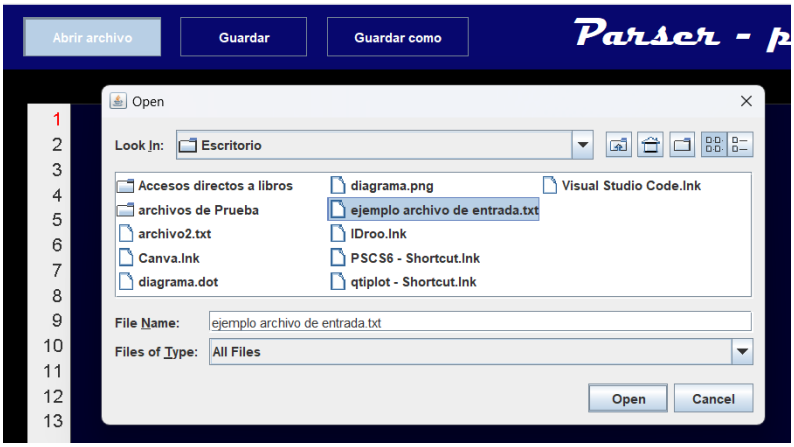
# PARTES DE LA APLICACIÓN


1. **Barra de opciones:** contiene opciones útiles.
2. **Editor:** área en donde se escribe el código, este cuenta con una visualización preliminar de los tokens, coloreándolos para una más cómoda visualización del código.
3. **Botón limpiar:** limpia el editor, el panel de errores y la tabla de reporte (véase el área de reporte).
4. **Botón Análisis:** ejecuta el análisis léxico y posteriormente el análisis sintáctico.
5. **Panel de errores:** al realizar el análisis, en este panel se muestran los errores léxicos así como sintácticos si se encontraran.
6. **Área de reportes de tokens:** contiene opciones útiles para visualizar y guardar reportes del análisis léxicos.

# Barra de Opciones

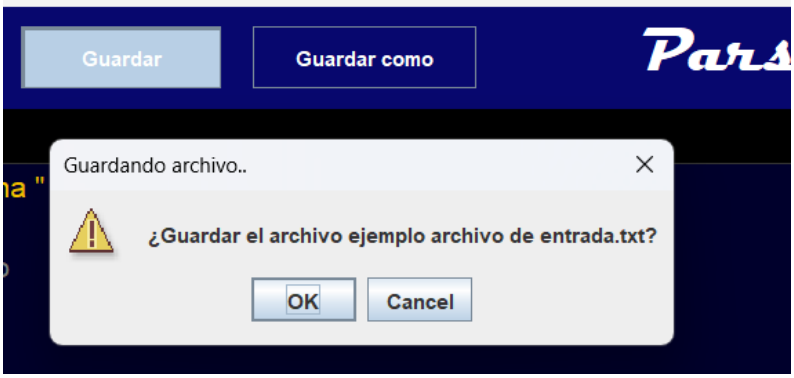

## 1. Botón Abrir Archivo

Permite abrir un archivo que puede contener código, este se mostrará en el editor para poder simplemente visualizarlo o editarlo.




## 2. Botón Guardar

Permite guardar el código del editor en un archivo previamente abierto, de lo contrario, permite crear uno nuevo.



## 3. Botón Guardar Como

Permite crear un nuevo documento que contendrá el código del editor.


## 4. Botón de reportes

Muestra los reportes correspondientes al análisis sintáctico y errores léxicos.

*Se muestra un resumen de los reportes:*


➔ **A: Tabla de símbolos global**

Tabla que muestra las variables y funciones definidas en el código del editor, con su tipo, valor, línea y columna.


| Simbolo | Tipo | Valor | Linea | Columna |
|---|---|---|---|---|
| miVariable | Cadena | "cadena de texto" | 2 | 1 |
| miVariable2 | int | 5 | 3 | 1 |
| miId | int | 10 | 4 | 1 |
| miDe2 | int | 20 | 5 | 1 |
| arreglo1 | desconocido | desconocido | 6 | 1 |
| arreglo2 | desconocido | desconocido | 7 | 1 |
| dicc | desconocido | desconocido | 8 | 1 |

### ➔ B: Tabla de símbolos por bloque

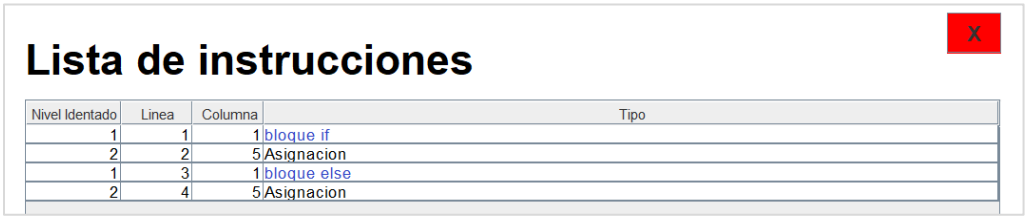
Tabla extendida de la tabla de símbolos global, que incluye un nivel de indentado; las líneas marcadas con azul representan un bloque de código global.

**Tabla de simbolos por Bloque de Codigo**

| Indentacion | Simbolo | Tipo | Valor | Linea | Columna |
|---|---|---|---|---|---|
| 1 | miVariable | Cadena | "cadena de texto" | 2 | 1 |
| 1 | miVariable2 | int | 5 | 3 | 1 |
| 1 | miId | int | 10 | 4 | 1 |
| 1 | miDe2 | int | 20 | 5 | 1 |
| 1 | arreglo1 | desconocido | desconocido | 6 | 1 |
| 1 | arreglo2 | desconocido | desconocido | 7 | 1 |
| 1 | dicc | desconocido | desconocido | 8 | 1 |

### ➔ C: Lista de instrucciones

Tabla que contiene un resumen de las instrucciones del código, incluye nivel de indentado, que especifica el bloque de código, la línea, la columna y el tipo.

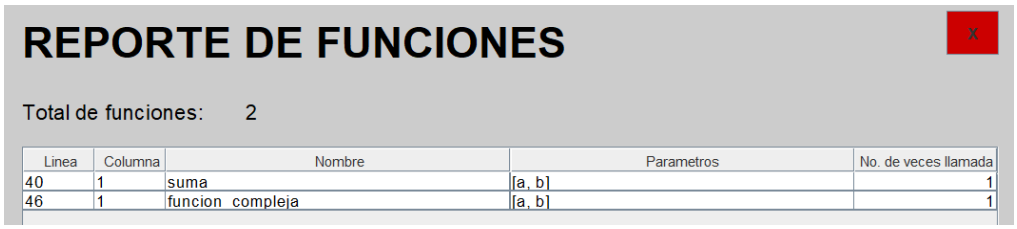


**Lista de instrucciones**

| Nivel Identado | Linea | Columna | Tipo |
|---|---|---|---|
| 1 | 1 | 1 | bloque if |
| 2 | 2 | 5 | Asignacion |
| 1 | 3 | 1 | bloque else |
| 2 | 4 | 5 | Asignacion |

### ➔ D: Reporte de métodos

Reporte de los métodos existentes en el código ingresado, muestra un contador del total, la línea y columna donde inicia su definición, el nombre de la función, los parámetros y el número de veces que ha sido llamado a lo largo del código.



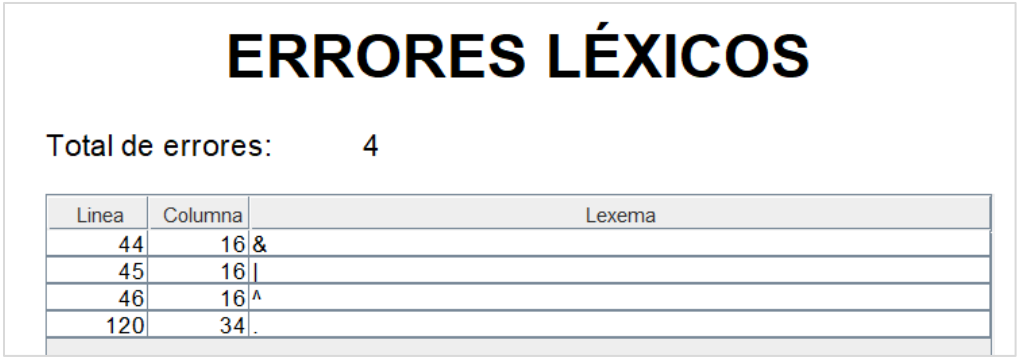
**REPORTE DE FUNCIONES**

Total de funciones: 2

| Linea | Columna | Nombre | Parametros | No. de veces llamada |
|---|---|---|---|---|
| 40 | 1 | suma | [a, b] | 1 |
| 46 | 1 | funcion_compleja | [a, b] | 1 |

### ➔ E: Errores léxicos

Muestra los errores léxicos encontrados en el código, se muestra vacía, si no se encuentra ninguno.



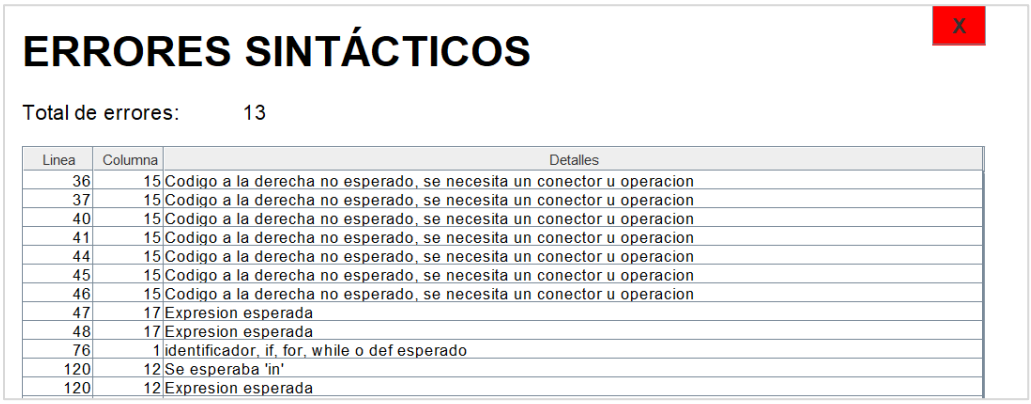
ERRORES LÉXICOS

Total de errores: 4

| Linea | Columna | Lexema |
|---|---|---|
| 44 | 16 | & |
| 45 | 16 | \| |
| 46 | 16 | ^ |
| 120 | 34 | . |

➔ **F: Errores sintácticos**

Muestra los errores léxicos encontrados en el código, se muestra vacía, si no se encuentra ninguno.



ERRORES SINTÁCTICOS

X

Total de errores: 13

| Linea | Columna | Detalles |
|---|---|---|
| 36 | 15 | Codigo a la derecha no esperado, se necesita un conector u operacion |
| 37 | 15 | Codigo a la derecha no esperado, se necesita un conector u operacion |
| 40 | 15 | Codigo a la derecha no esperado, se necesita un conector u operacion |
| 41 | 15 | Codigo a la derecha no esperado, se necesita un conector u operacion |
| 44 | 15 | Codigo a la derecha no esperado, se necesita un conector u operacion |
| 45 | 15 | Codigo a la derecha no esperado, se necesita un conector u operacion |
| 46 | 15 | Codigo a la derecha no esperado, se necesita un conector u operacion |
| 47 | 17 | Expresion esperada |
| 48 | 17 | Expresion esperada |
| 76 | 1 | identificador, if, for, while o def esperado |
| 120 | 12 | Se esperaba 'in' |
| 120 | 12 | Expresion esperada |

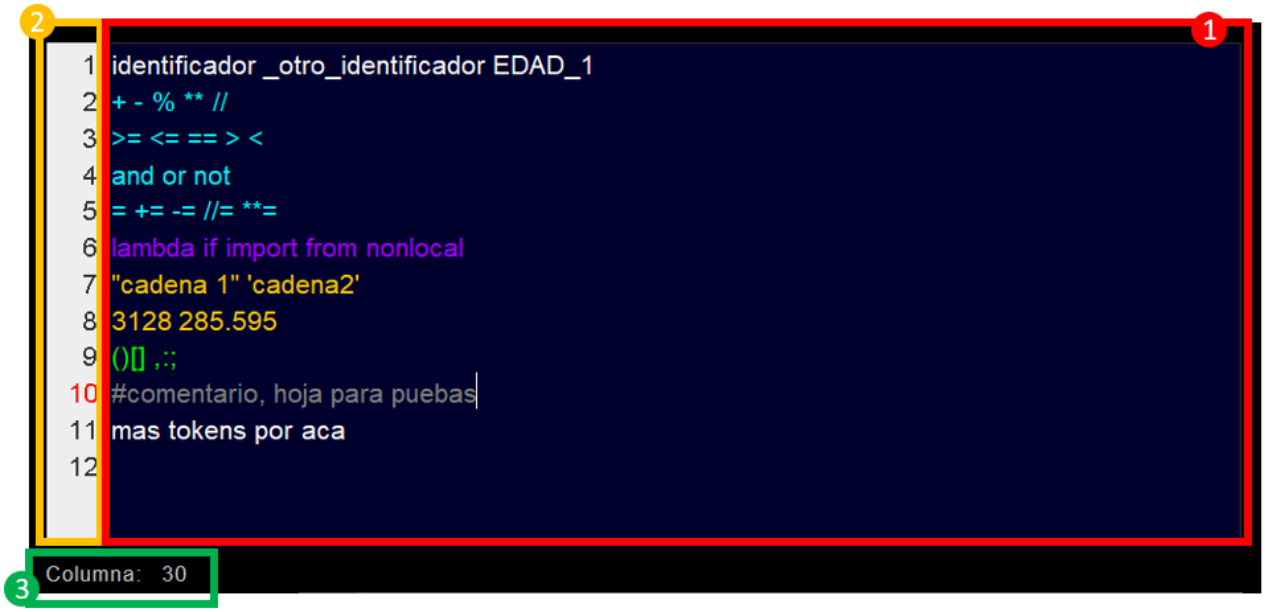
## 5. Botón Ayuda

Muestra un enlace a esta guía.

## 6. Botón Créditos

Muestra los créditos de la realización de este proyecto.

# Editor




Área en donde se puede escribir el código **(1)**, dependiendo del tipo de token se coloreará para una mejor visualización, esto es según la siguiente tabla (para detalles más técnicos, véase el manual técnico):

| *TOKEN* | *COLOR* |
|---|---|
| *Identificadores* | Blanco |
| *Operadores Aritmeticos, comparativos, lógicos, asignación* | Celeste |
| *Palabras clave o reservadas* | Morado |
| *Constantes (numeros, cadenas, booleanos)* | Anaranjado |
| *Comentarios* | Gris |
| *Otros (corchetes, paréntesis, etc.)* | Verde |
| *Errores (carácter irreconocible, cadena que no cierra)* | Rojo |

Se muestra además una sencilla visualización de la línea **(2)** y la columna **(3)** en donde se encuentra el puntero.

# Botón Limpiar

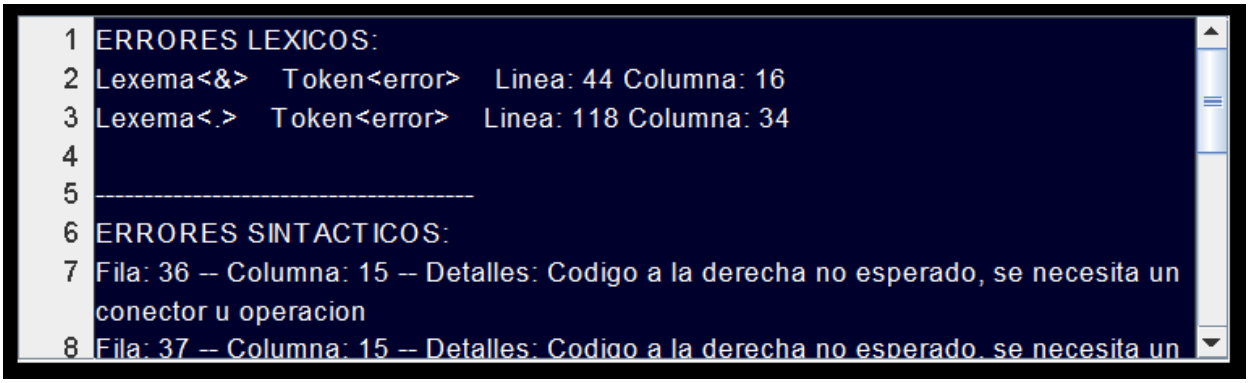
Limpia el editor, el panel de errores y la tabla de reporte.

# Botón Análisis

Ejecuta el análisis léxico y sintáctico del código contenido en el editor; el análisis incluye mostrar los errores en el panel de errores y mostrar un reporte de tokens en la tabla correspondiente.

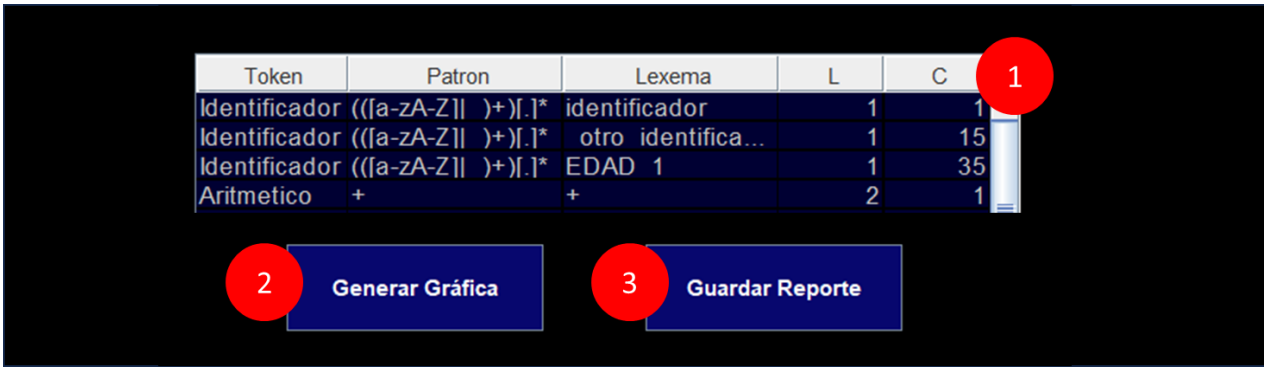
# Panel de errores

Al realizar el análisis, en este panel se muestran los errores si se encontraran.




***Nota:*** *si no hubiera ningún error léxico y/o sintáctico, en este apartado se especifica que no se encontraron errores*

# Área de Reportes




1. **Tabla de reporte:** al realizar el análisis léxico, en este apartado se muestran los tokens encontrados, así como información útil sobre los mismos.
2. **Botón Generar Gráfica:** genera la gráfica de un toquen previamente seleccionado.
3. **Botón Guardar Reporte:** realiza un análisis léxico y guarda el reporte del mismo en un documento de texto.

## Tabla de reporte

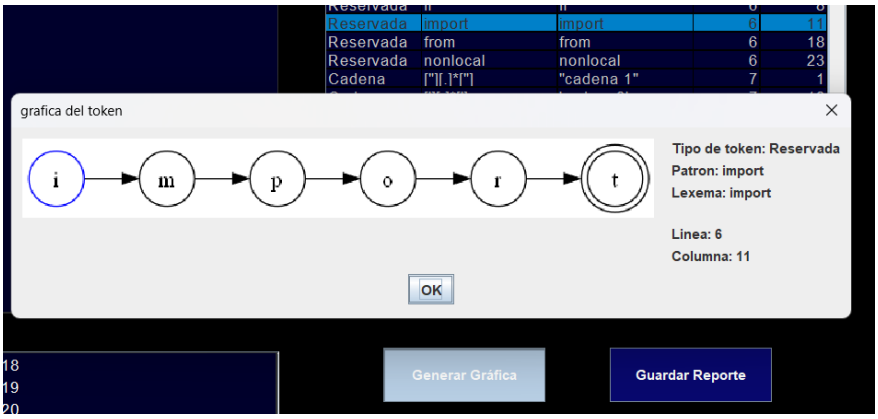
Al realizar el análisis léxico, en este apartado se muestra la lista de tokens analizados, mostrando la siguiente información:

1. **Type:** qué tipo de token es, a que categoría pertenece.
2. **Token:** el tipo de token especifico.
3. **Patrón:** representado con expresiones regulares, muestra el patrón que cumplió el lexema para ser considerado de algún tipo de token en específico
4. **Lexema:** el contenido del token.
5. **L:** abreviatura para Linea, muestra el numero de linea en donde se encuentra el token analizado.
6. **C:** abreviatura para Columna, muestra el numero de la columna en donde inicial el token analizado

## Botón “Generar gráfica”

Genera la gráfica de un toquen previamente seleccionado.

Para seleccionar un token se debe de hacer clic en la tabla de reporte, se iluminará la celda seleccionada, posteriormente se debe de dar clic a este botón, y nos aparecerá la gráfica del token seleccionado.




El circulo pintado de azul (también puede ser más de uno), representa el inicio del token

El circulo con un margen doble (puede ser más de uno ) representa el fin de un token

**Notas:**

- ✓ *Si se selecciona más de una fila, al presionar este botón se graficará unicamente el primer token seleccionado.*
- ✓ *No se generarán gráficas de los errores léxicos.*

## Botón “Guardar Reporte”

Realiza un análisis léxico y guarda la información presentada en la Tabla de reporte en un documento de texto que podemos nombrar.

# LENGUAJE RECONOCIDO POR PARSER-PY

*El lenguaje que reconoce Parser-pY, se inspiró en Python, a continuación, se especifica la sintaxis para programar:*

## Bloques de código

Todas las sentencias que pertenecen a un bloque de código deben de tener la misma indentación.

## Funciones

### Definición de funciones

```
def nombre_funcion(parametro1, parametro2, ...):
    #bloque de codigo
    return valor_de_retorno  # opcional
```

### Uso de funciones

```
funcion(parametro1, parametro2, ...)
funcion()
```

## Declaración y asignación de variables

```
variable += "Expresion aqui"
variable_1, variable_2, variable_3 = expresion_1, expresion_2, expresion_3
```

## Sentencia condicional

```
if condicion :
    print("se cumplio la condicion 1")
elif otra_condicion : #opcional
    print("la primera condición no se cumplió, pero la segunda si")
else: #opcional
    print("no se cumplieron las condiciones anteriores")
```

# Ciclos:

## For

```
for item in container:
    if busca_algo(item):
        procesar(item)
        break;
else: #opcional, se ejecuta si el ciclo no fue interrumpido por el break
    print("no se encontro")
```

## While

```
while contador < 5:
    print(contador)
    contador += 1
```

# Expresiones

Se muestran expresiones asignadas a variables específicas, pero las expresiones se aceptan en otras partes del código, (para más información, consultar el manual técnico).

```
#constantes
var_int = 1 #enteros
var_float = 1.5 #decimales
var_String = "Yennifer" #cadenas de texto
var_boolean = True

#diccionarios y listas
empty_dictionary = {}
dictionary = { "id": 1, "name": "Yennifer"}
empty_list = []
list = [1,2,3]

#operador ternario
var = "verdadero" if condición else "falso"
```

## Operaciones entre expresiones:

```
#Operaciones matemáticas
#suma, resta, multiplicación, división, potenciación
var = ( (1 + a) / (b - 8) )*8

#operaciones comparativas
var = a > b #devulve True/False

#operaciones logicas
var = ((a > b) and not(condition)) or second_condition #devulve True/False
```

# TERMINOLOGÍA

- **Analisis Léxico:** Es la primera fase de un compilador, toma el programa fuente de los pre-procesadores que está escrito en forma de declaraciones. Este proceso desglosa el código en una serie de tokens, deshaciéndose primero de todos los comentarios en el código y los espacios en blanco.

- **Análisis Sintáctico:** Es la fase del analizador que se encarga de chequear el texto de entrada en base a una gramática dada.

- **Analizador Léxico:** programa que realiza un análisis léxico.

- **Analizador Sintáctico:** es una de las partes de un compilador que transforma su entrada en un árbol de derivación.

- **Errores léxicos:** Son aquellos que se detectan cuando el analizador léxico intenta reconocer componentes léxicos y la cadena de caracteres de la entrada no encaja con ningún patrón. *Algunos ejemplos son: Se escribe un carácter inválido, No se cierran comillas.*

- **Errores Sintácticos:** Son los que se producen cuando se viola la sintaxis, del lenguaje. *Ejemplo: cuando se omiten los dos puntos de un bucle while.*

- **Expresión Regular**: son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto. Las expresiones regulares proporcionan una manera muy flexible de buscar o reconocer cadenas de texto.

- **Gramática**: conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

- **Lexema:** secuencia de caracteres en el programa fuente, que coinciden con el patrón para un token y que el analizador léxico identifica como una instancia de ese token.

- **Patrón:** sucesión de elementos que se construyen siguiendo una regla.

- **Sintaxis:** conjunto de reglas que definen la manera de escribir instrucciones de código.

- **Token:** El token es una referencia (un identificador) que regresa a los datos sensibles a través de un sistema de tokenización.