UNIVERSIDAD AUTÓNOMA DE TAMAULIPAS
FACULTAD DE INGENIERÍA TAMPICO

PROYECTO INTEGRADOR

NOMBRE DE LA UNIDAD
DESARROLLO DE UN ANALIZADOR LÉXICO
NOMBRE DE LA ASIGNATURA
PROGRAMACIÓN DE SISTEMAS DE BASE I

Docente: Muñoz Quintero Dante Adolfo
8vo. Semestre – Grupo **"I"**
2024-2

Fecha de Entrega: Lunes, 2 de Diciembre de 2024, 20:00

Integrante(s):
Cruz Bonifacio Luis Fernando - 2173390170

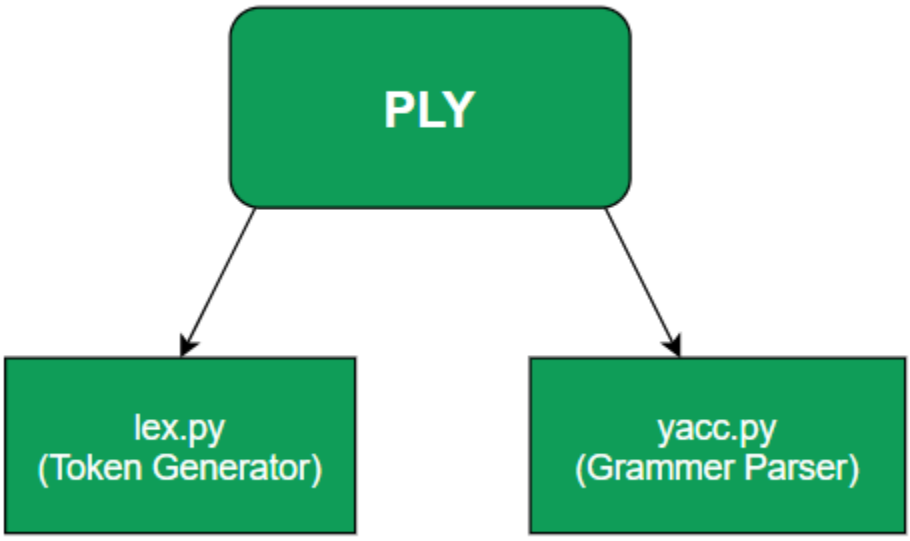
Contenido

1.-Descripcion General.

El objetivo de este proyecto es diseñar un analizador léxico (lexer) para un lenguaje personalizado, que será utilizado en una aplicación creada por el estudiante. Este leer transformará un archivo de texto que contenga un “programa” en una lista de tokens correspondientes a la gramática definida. La especificación del lenguaje será diseñada de manera informal utilizando ejemplos ilustrativos de código.

El proyecto incluye las siguientes fases:

- Diseño de la especificación del lenguaje: Definición de la sintaxis básica del lenguaje.
- Implementación del analizador léxico: Creación de un lexer funcional utilizando Python y su librería Ply (Python Lex-Yacc) que es una herramienta de generación de analizadores en **Python**, inspirada en **Lex** y **Yacc**, que son herramientas tradicionales para la generación de analizadores léxicos y sintácticos en C.
- Pruebas y validación: Verificación del correcto funcionamiento del lexer, incluyendo la identificación de errores léxicos.



2.-Breve explicación de un Analizador Léxico.

¿Qué es un analizador léxico (lexer)?

Desde la entrada del usuario hasta la ejecución del código, existen tres pasos principales, tanto para intérpretes como para compiladores:

1. **Análisis léxico** (1)
2. **Análisis sintáctico** (2)
3. **Generación de código** (por ejemplo, a código de máquina o bytecode)
4. **Ejecución** (3)

Un **analizador léxico** (o lexer) es una herramienta que realiza el análisis léxico, que es el primer paso en este proceso.

Diferencia entre análisis léxico y escaneo

Al principio, el "escaneo" y el "análisis léxico" eran dos pasos distintos, pero debido a la mayor velocidad de los procesadores, ahora ambos términos se utilizan de forma intercambiable y se refieren al mismo proceso.

¿Qué es el escaneo?

El **escaneo** significa pasar por una cadena de caracteres, uno por uno, analizando cada carácter (carácter por carácter).

¿Qué es el análisis léxico?

Es el proceso mediante el cual los caracteres escaneados se convierten en **lexemas**.

¿Qué es un lexema?

Un **lexema** es una secuencia de caracteres reconocida y procesada como una unidad significativa dentro de un lenguaje.

Veamos un ejemplo con una frase:

The quick brown fox

Los lexemas serían:

- Lexema 1: the
- Lexema 2: quick
- Lexema 3: brown
- Lexema 4: fox

Ahora, con un fragmento de código similar:

for(i; i<arr.length; i++){ }

Los lexemas serían:

- Lexema 1: for
- Lexema 2: (
- Lexema 3: i
- Lexema 4: ;
- Lexema 5: i
- Lexema 6: <
- Lexema 7: arr
- Lexema 8: .
- Lexema 9: length
- Lexema 10: ;
- Lexema 11: i
- Lexema 12: ++
- Lexema 13:)
- Lexema 14: {
- Lexema 15: }

¿Qué necesita un lexer como entrada?

Un lexer necesita dos cosas:

1. El código fuente (source code).
2. Las palabras clave (keywords).

¿Qué es una palabra clave (keyword)?

Una **palabra clave** es un lexema que tiene un significado especial para el lexer. Normalmente, palabras como *print*, *from*, *to* son conocidas como palabras clave, pero también símbolos como *(* o *{* pueden ser considerados como palabras clave.

Tipos de palabras clave

- Existen palabras clave de un solo carácter, como los espacios en blanco.
- Existen palabras clave de múltiples caracteres, como *print*.

Ignorando palabras clave

Hay casos en los que queremos ignorar algunas palabras clave, como cuando aparecen entre comillas ("") o dentro de comentarios.

¿Qué son los identificadores (IDs)?

Los **identificadores** son nombres definidos por el usuario, como los nombres de espacios de nombres (namespaces), variables, clases y funciones. Las funciones de la biblioteca estándar son identificadores predefinidos. Normalmente, los nombres de palabras clave no pueden ser utilizados como identificadores.

3.- Tareas del Proyecto

1. Especificación del Lenguaje Personalizado(Simplex):

Descripción del Lenguaje *Simplex*

Simplex es un lenguaje de programación de propósito general diseñado para ser fácil de aprender y utilizar. Su sintaxis es simple y está inspirada en lenguajes como Python y JavaScript, lo que lo hace accesible para programadores novatos, pero con suficiente capacidad para desarrollar proyectos pequeños y medianos.

Características Principales:

1. **Sintaxis Simple y Clara**: *Simplex* utiliza una sintaxis sencilla y legible, adecuada para programadores principiantes, eliminando palabras clave complicadas o redundantes.
2. **Control de Flujo**: El lenguaje soporta estructuras básicas de control de flujo, como condicionales (if, else), bucles (while, for) y sentencias de control de ejecución (break, continue).
3. **Tipos de Datos**: *Simplex* maneja tipos de datos primitivos como enteros (int), números decimales (float), cadenas de texto (string) y booleanos (bool). Además, permite declarar variables fácilmente.
4. **Operadores**: Incluye operadores aritméticos (+, -, *, /), lógicos (and, or), de comparación (==, !=, >, <) y de asignación (=, +=, -=).
5. **Funciones**: *Simplex* permite la definición de funciones con la palabra clave function, sin necesidad de especificar tipos de retorno o parámetros, lo que permite una programación ágil y flexible.
6. **Estructuras de Datos**: Soporta la creación de arrays y listas para almacenar colecciones de datos, con índices numéricos o asociativos.
7. **Manejo de Errores**: El lenguaje proporciona un mecanismo sencillo para el manejo de errores mediante la detección de identificadores no válidos y símbolos desconocidos, generando mensajes claros que incluyen la línea y columna donde ocurrió el error.
8. **Comentarios**: Los comentarios de una sola línea se indican con // y los comentarios multi-línea con /* */, permitiendo una documentación clara dentro del código.

Ejemplo de Código en *Simplex*:

simplex

```
function main() {
    // Declaración de variables
    int x = 5;
    float y = 10.5;

    // Estructura condicional
    if (x < y) {
        print("x es menor que y");
    } else {
        print("x no es menor que y");
    }

    // Bucle while
    while (x < 10) {
        x = x + 1;
    }

    // Uso de una función
    int z = suma(x, y);
    print(z);
}

// Función para sumar dos números
function suma(int a, float b) {
    return a + b;
}
```

Propósito del Lenguaje:

Simplex está diseñado para proyectos pequeños donde se busca rapidez en la escritura y legibilidad del código, pero también con la capacidad de extenderse para tareas de complejidad media. Ideal para programadores novatos, pero también útil en prototipos rápidos y proyectos de scripting.

Casos de Uso:

- **Educación**: Como herramienta de aprendizaje de programación.
- **Prototipos**: Para crear prototipos rápidamente sin complicaciones.
- **Automatización**: En scripts simples para automatización de tareas.

2. Especificación del Analizador Léxico:

Objetivo del Analizador Léxico

El analizador léxico (lexer) de *Simplex* se encargará de leer el código fuente del lenguaje *Simplex* y dividirlo en tokens. Un token es una secuencia de caracteres que tiene un significado en el lenguaje. El lexer identificará y clasificará estos tokens según su tipo, como identificadores, palabras reservadas, operadores, delimitadores y números.

Elementos del Lexer:

1. **Identificadores:**
   - **Reglas**: Los identificadores de *Simplex* son secuencias de caracteres alfanuméricos que comienzan con una letra (a-z, A-Z) o un guion bajo (_), seguidos de letras, números o guiones bajos.
   - **Ejemplo**:
     - Válido: variable, _variable123
     - Inválido: 123variable, @variable
   - **Expresión Regular**:[a-zA-Z_][a-zA-Z0-9_]*
   - **Token**: IDENTIFIER.

2. **Palabras Reservadas:**
   - **Reglas**: Las palabras reservadas son términos predefinidos que tienen un significado especial en *Simplex*. Estas palabras no pueden usarse como identificadores.
   - **Palabras reservadas**: function, if, else, while, return, print, int, float, string, bool, true, false, etc.
   - **Expresión Regular**: El lexer puede tener un conjunto fijo de palabras clave, que se compararán al leer el texto.
   - **Token**: KEYWORD.

3. **Operadores:**
    - **Reglas**: *Simplex* soporta operadores aritméticos, lógicos y de asignación, entre otros.
    - **Operadores aritméticos**: +, -, *, /, %
    - **Operadores lógicos**: and, or
    - **Operadores de asignación**: =, +=, -=, *=, /=, %=
    - **Operadores de comparación**: ==, !=, <, >, <=, >=
    - **Expresión Regular**: [\+\-*/%]=?|==|!=|<=|>=|and|or
    - **Token**: OPERATOR.

4. **Delimitadores:**
    - **Reglas**: Los delimitadores son símbolos que se usan para separar sentencias, bloques de código o elementos dentro de las estructuras de datos.
    - **Delimitadores comunes**: ;, ,, {, }, (,), [,]
    - **Expresión Regular**: ;|,|\{|\}|\(|\)|\[|\]
    - **Token**: DELIMITER.

5. **Números:**
    - **Reglas**: *Simplex* soporta números enteros y flotantes. Los enteros pueden ser decimales, octales o hexadecimales. Los números flotantes pueden tener una parte decimal opcional.
    - **Números enteros**:
        - Decimales: 123, 456
        - Hexadecimales: 0x1A, 0xFF
        - Octales: 075
    - **Números flotantes**: 3.14, 0.99, 1.0e10
    - **Expresión Regular**:
      0x[0-9A-Fa-f]+|0[0-7]+|\d+\.\d+|\d+(\.\d*)?([eE][+]?\d+)?
    - **Token**: NUMBER.

Manejo de Errores

El lexer debe manejar los errores léxicos de manera adecuada. Si encuentra un símbolo que no puede clasificar según las reglas anteriores, generará un token especial de error. Esto es especialmente importante para caracteres no válidos en *Simplex*, como símbolos no permitidos o identificadores mal formateados.

- **Símbolos no válidos**: Cuando se encuentran símbolos no reconocidos (como caracteres fuera del rango alfabético o números mal formateados), el lexer generará un error.
- **Expresión Regular para caracteres no válidos**:
  [^a-zA-Z0-9_ \t\n\r;,.{}()\[\]=+\-*\/<>!&|]

El token correspondiente será INVALID_SYMBOL o similar.

Proceso del Lexer

1. **Lectura del Código Fuente**: El lexer toma el código fuente como entrada y lo lee carácter por carácter.
2. **Clasificación de Tokens**: Según las expresiones regulares y las reglas definidas para cada tipo de token (identificadores, palabras reservadas, operadores, etc.), el lexer identifica y clasifica los tokens.
3. **Manejo de Errores**: Si se encuentra un símbolo no reconocido, se reporta un error léxico con la ubicación (número de línea y columna).
4. **Generación de Tokens**: Cada token identificado se genera y se almacena en una lista o estructura adecuada para su posterior procesamiento en el analizador sintáctico (parser).

Ejemplo de Entrada y Salida

Entrada:

Simplex

```
function main() {
   int x = 10;
   if (x > 5) {
      print("x is greater than 5");
   }
}
```

Salida del Lexer:

Token	**Valor**
KEYWORD	function
IDENTIFIER	main
DELIMITER	(
DELIMITER	)
KEYWORD	int
IDENTIFIER	x
OPERATOR	=
NUMBER	10
DELIMITER	;
KEYWORD	if
DELIMITER	(
IDENTIFIER	x
OPERATOR	>
NUMBER	5

Token	Valor
DELIMITER	)
DELIMITER	{
KEYWORD	print
DELIMITER	(
STRING	"x is greater than 5"
DELIMITER	)
DELIMITER	;
DELIMITER	}

Resumen del Flujo del Lexer:

- El lexer recibe un archivo de texto con código fuente en *Simplex*.
- Clasifica las secuencias de caracteres en tokens (identificadores, palabras clave, operadores, delimitadores, números, etc.).
- Si encuentra un símbolo no reconocido o una sintaxis incorrecta, lo marca como error.
- Los tokens generados se pasan al siguiente paso del compilador o intérprete para el análisis sintáctico.

Este enfoque proporcionará una base sólida para el analizador léxico de *Simplex*, asegurando que se puedan procesar correctamente los archivos fuente y que cualquier error léxico sea reportado adecuadamente.

3. Implementación del Analizador Léxico

El analizador léxico se implementó utilizando la biblioteca **PLY (Python Lex-Yacc)**, siguiendo las funcionalidades especificadas en el proyecto. Este lexer procesa archivos de texto que contienen código escrito en el lenguaje **Simplex** y realiza las siguientes tareas:

Funcionalidades del Lexer:

1. **Identificación de Tokens**.

   El lexer identifica y clasifica los elementos válidos en el programa de entrada, generando una lista de tokens. Los tokens incluyen:

   - **T_KEYWORD**: Palabras clave reservadas como if, else, while.
   - **T_IDENTIFIER**: Identificadores válidos como mi_variable o funcion.
   - **T_INVALID_IDENTIFIER**: Identificadores inválidos, como aquellos que no cumplen con las reglas del lenguaje (por ejemplo, 123variable).
   - **T_OPERATOR**: Operadores aritméticos, lógicos y de comparación (+, -, *, /, >, <, etc.).
   - **T_ASSIGNMENT_OPERATOR**: Operadores de asignación (=, +=, -=, etc.).
   - **T_DELIMITER**: Delimitadores como ;, ,, {, }, (y).
   - **T_NUMBER**: Números enteros, flotantes o hexadecimales (como 10, 3.14, 0x1F).
   - **T_STRING**: Literales de cadenas de texto, delimitadas por comillas simples o dobles.
   - **T_COMMENT**: Comentarios de una línea (//) y de múltiples líneas (/* ... */).
   - **T_UNRECOGNIZED_SYMBOL**: Símbolos no reconocidos, como $ y #.

2. **Manejo de Errores**.

   El lexer detecta e informa errores relacionados con:

   - **Identificadores no válidos**: Identificadores que no cumplen con las reglas del lenguaje, como aquellos que comienzan con un número o contienen caracteres no permitidos.
   - **Símbolos desconocidos**: Caracteres no válidos en el lenguaje, como $ y #.

3. **Mensajes de Error Informativos.**

   Cada error léxico incluye:

   - **Número de línea**: Línea del archivo donde ocurrió el error.
   - **Columna**: Posición exacta en la línea donde se detectó el error.
   - **Descripción clara del problema**: Mensaje detallado explicando la naturaleza del error.

4. Pruebas del Lexer

4.1 Programas Válidos

Objetivo: Verificar que el lexer pueda identificar correctamente los tokens en programas que siguen la especificación del lenguaje **Simplex**.

Ejemplo 1: Condicional básico

Código:

```
if x > 10 then
   y = x * 2;
else
   z = y / 3;
```

Tokens esperados:

(KEYWORD, "if")
(IDENTIFIER, "x")
(OPERATOR, ">")
(NUMBER, "10")
(KEYWORD, "then")
(IDENTIFIER, "y")
(OPERATOR, "=")
(IDENTIFIER, "x")
(OPERATOR, "*")
(NUMBER, "2")
(DELIMITER, ";")
(KEYWORD, "else")
(IDENTIFIER, "z")
(OPERATOR, "=")
(IDENTIFIER, "y")
(OPERATOR, "/")
(NUMBER, "3")
(DELIMITER, ";")

Descripción: Este es un ejemplo de un programa con una estructura básica de un condicional if-else. El lexer debería identificar correctamente cada token según las reglas definidas en las expresiones regulares.

Resultado esperado: Los tokens generados deben coincidir exactamente con los que se listan arriba. Si el lexer está funcionando correctamente, no debería haber errores y el programa debería generar una lista de tokens.

Ejemplo 2: Declaración de variable

Código:

```
int x = 5;
float y = 3.14;
```

Tokens esperados:

(KEYWORD, "int")
(IDENTIFIER, "x")
(ASSIGNMENT_OPERATOR, "=")
(NUMBER, "5")
(DELIMITER, ";")
(KEYWORD, "float")
(IDENTIFIER, "y")
(ASSIGNMENT_OPERATOR, "=")
(NUMBER, "3.14")
(DELIMITER, ";")

Descripción: Este código contiene dos asignaciones de variables. Se debe verificar que los tipos int y float sean reconocidos como palabras clave, y que las asignaciones se manejen correctamente. También se deben identificar los identificadores y los números flotantes correctamente.

Resultado esperado: Los tokens deben coincidir con los valores esperados, con un manejo adecuado de palabras clave, identificadores, operadores y delimitadores.

Ejemplo 3: Operaciones aritméticas

Código:

a = 10 + 20 * (x - y);

Tokens esperados:

(IDENTIFIER, "a")

(ASSIGNMENT_OPERATOR, "=")

(NUMBER, "10")

(OPERATOR, "+")

(NUMBER, "20")

(OPERATOR, "*")

(DELIMITER, "(")

(IDENTIFIER, "x")

(OPERATOR, "-")

(IDENTIFIER, "y")

(DELIMITER, ")")

(DELIMITER, ";")

Descripción: Este ejemplo incluye operadores matemáticos y paréntesis. El lexer debe reconocer los operadores aritméticos (+, *, -), los números y los delimitadores, como el paréntesis y el punto y coma.

Resultado esperado: Cada token debe ser correctamente identificado y clasificado, según el tipo (identificador, número, operador, delimitador).

4.2 Casos con Errores

Objetivo: Verificar que el lexer pueda detectar y reportar correctamente los errores léxicos, como identificadores inválidos o caracteres no reconocidos.

Ejemplo 1: Identificador inválido

Código:

123abc = 5;

Errores esperados:

Error: Línea 1, columna 1: Identificador inválido '123abc'.

Descripción: En este caso, el identificador 123abc no es válido, ya que los identificadores deben comenzar con una letra o un guion bajo. El lexer debería identificar esto como un error y generar un mensaje de error indicando que el identificador es inválido.

Resultado esperado: El lexer debe generar un mensaje de error que indique la ubicación del identificador inválido.

Ejemplo 2: Símbolo no reconocido

Código:

x $ y;

Errores esperados:

Error: Línea 1, columna 4: Símbolo no reconocido '$'.

Descripción: El símbolo $ no está definido como un token válido en el lenguaje Simplex. El lexer debería identificar este símbolo no reconocido y generar un mensaje de error adecuado.

Resultado esperado: El lexer debe generar un mensaje de error que indique que el símbolo $ no es reconocido.

Ejemplo 3: Comillas desbalanceadas

Código:

string texto = "Esto es un ejemplo;

Errores esperados:

Error: Línea 1, columna 29: Error léxico debido a comillas desbalanceadas.

Descripción: En este ejemplo, la cadena de texto comienza con comillas dobles (") pero no se cierra adecuadamente. El lexer debe detectar esta desbalance de comillas y generar un error.

Resultado esperado: El lexer debe generar un mensaje de error indicando que hay comillas desbalanceadas.

4.3 Conclusiones

Pruebas realizadas: Se realizaron pruebas con programas válidos y casos que contienen errores léxicos para asegurar que el lexer funcione correctamente y reporte adecuadamente los errores.

Resultado esperado: El lexer generó tokens correctamente para los programas válidos, y los errores fueron reportados con precisión, incluyendo la línea y columna del error. Esto demuestra que el lexer está manejando correctamente tanto los casos válidos como los errores de acuerdo con las especificaciones del lenguaje Simplex.

3. Funcionalidades del Lexer

3.1 Reglas Léxicas

El lexer implementa las siguientes reglas léxicas para reconocer y clasificar los distintos componentes de un programa en el lenguaje **Simplex**:

1. **Identificadores Válidos**: El lexer es capaz de identificar identificadores válidos según las siguientes reglas:
    - Deben comenzar con una letra (a-z, A-Z) o un guion bajo (_).
    - Pueden contener letras, dígitos (0-9), y acentos (á, é, í, ó, ú).

Ejemplos de identificadores válidos:

- variable1, my_function, x, edad_estudiante.

2. **Identificadores Inválidos**: El lexer también detecta identificadores inválidos. Un identificador inválido:
    - No puede comenzar con un número ni contener caracteres no permitidos, como símbolos o signos.

Ejemplos de identificadores inválidos:

- 123abc, @var, $variable, var#1.

3. **Diferenciación de Palabras Reservadas**: El lexer diferencia correctamente las palabras reservadas (o palabras clave) de los identificadores. Las palabras clave son términos reservados del lenguaje **Simplex** y no pueden usarse como identificadores.

Ejemplos de palabras reservadas:

- if, else, while, for, return, int, float, string, true, false.

El lexer reconocerá las palabras clave y las clasificará como KEYWORD, mientras que cualquier otro término válido que no sea palabra clave se clasifica como IDENTIFIER.

4. **Reconocimiento de Símbolos Válidos**: El lexer reconoce varios símbolos válidos del lenguaje, que incluyen:
    - **Operadores**: +, -, *, /, =, ==, >, <.
    - **Delimitadores**: ;, {, }, (,).
    - **Comillas** para cadenas: ".

Ejemplos de operadores y delimitadores:

- Operadores: =, +, *, ==, >, <.
- Delimitadores: ;, {, }, (,).

5. **Manejo de Números**: El lexer maneja números en varios formatos permitidos:
    - **Números enteros**: 10, 123.
    - **Números decimales**: 3.14, 0.75.
    - Los números son clasificados como NUMBER.

Ejemplos de números:

- Números enteros: 5, 100, 2000.
- Números decimales: 3.14, 0.1, 100.56.

3.2 Manejo de Errores

El lexer es capaz de manejar los siguientes tipos de errores:

1. **Símbolos No Reconocidos**: El lexer detecta símbolos no reconocidos y los reporta como errores. Símbolos como $ o # no están definidos en el lenguaje **Simplex**, por lo que deben ser marcados como errores léxicos.

Ejemplos de símbolos no reconocidos:

- $, #, @, ^.

Acción: El lexer genera un mensaje de error indicando que el símbolo no es reconocido, junto con la ubicación (línea y columna) donde se encuentra.

2. **Identificadores Inválidos**: El lexer también detecta identificadores que no cumplen con las reglas de nomenclatura del lenguaje **Simplex**. Un identificador no puede comenzar con un número ni contener caracteres no permitidos (como símbolos).

Ejemplos de identificadores inválidos:

- 123abc, @var, variable#.

Acción: El lexer genera un mensaje de error detallado indicando que el identificador es inválido, y muestra la línea y columna donde ocurre el error.

3. **Comillas Desbalanceadas**: En el caso de cadenas de texto (strings), el lexer detecta comillas desbalanceadas (cuando una cadena comienza con una comilla pero no termina correctamente).

Ejemplo de error:

- "Texto de cadena sin cierre.

Acción: El lexer informa que las comillas están desbalanceadas y proporciona la ubicación exacta del error.

3.3 Salida del Lexer

El lexer genera dos tipos de salida dependiendo de si el análisis fue exitoso o si se detectaron errores:

1. **Entrada Válida**: Cuando el lexer procesa un programa sin errores, genera una lista ordenada de tokens, en la que se detalla el tipo de cada token, su valor y su ubicación en el archivo de entrada (número de línea y columna).

Ejemplo de salida para una entrada válida:

KEYWORD: if en línea 1, columna 1

IDENTIFIER: x en línea 1, columna 4

OPERATOR: > en línea 1, columna 6

NUMBER: 10 en línea 1, columna 8

KEYWORD: then en línea 1, columna 11

2. **Entrada con Errores**: Cuando el lexer detecta errores, los muestra en un formato detallado, especificando la naturaleza del error (símbolo no reconocido, identificador inválido, etc.), y proporcionando la ubicación precisa del error en el código fuente (línea y columna).

Ejemplo de salida para un error:

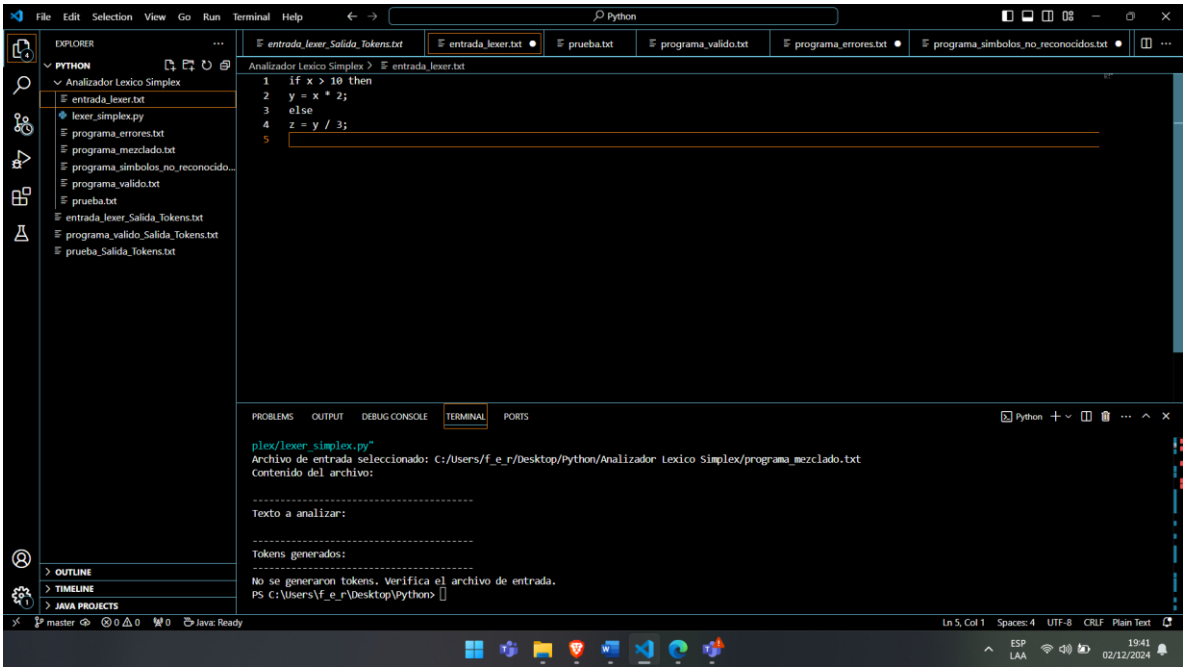
Error: Carácter no reconocido '$' en línea 1, columna 4

Error: Identificador inválido '123abc' en línea 2, columna 1

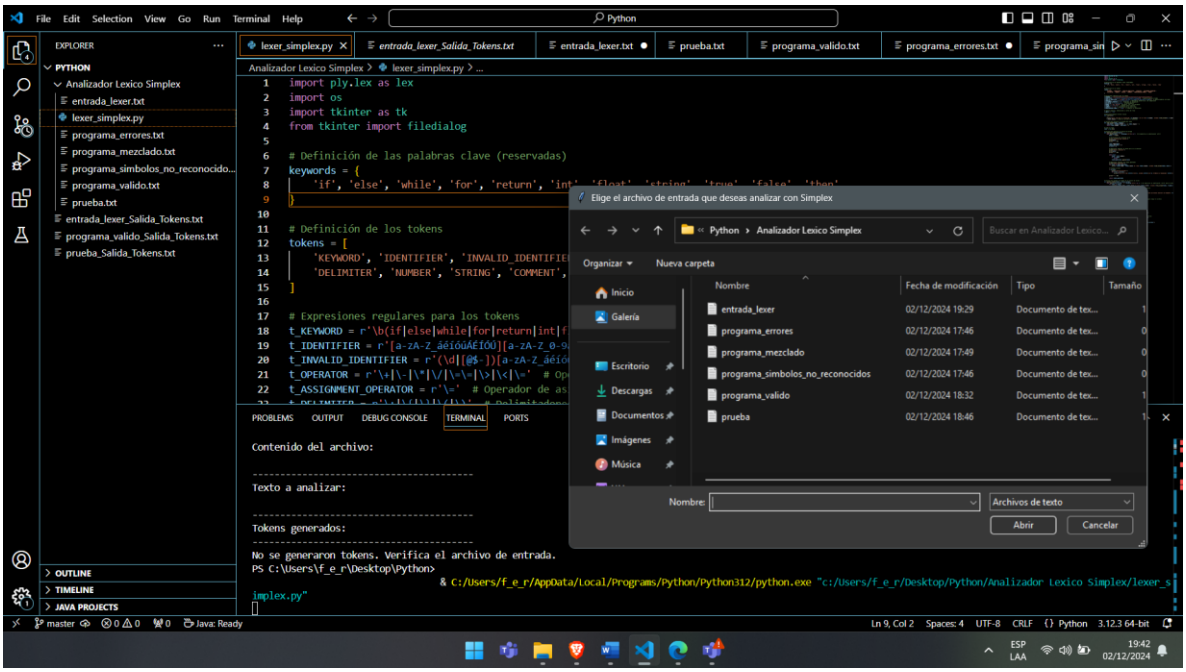
Resumen: El lexer implementa reglas léxicas robustas para manejar la sintaxis básica del lenguaje **Simplex**, permite la identificación de errores y proporciona salidas detalladas de los tokens generados o los errores detectados, facilitando la depuración y el análisis de programas escritos en **Simplex**.

4.-Ejemplo de Salida del Lexer.

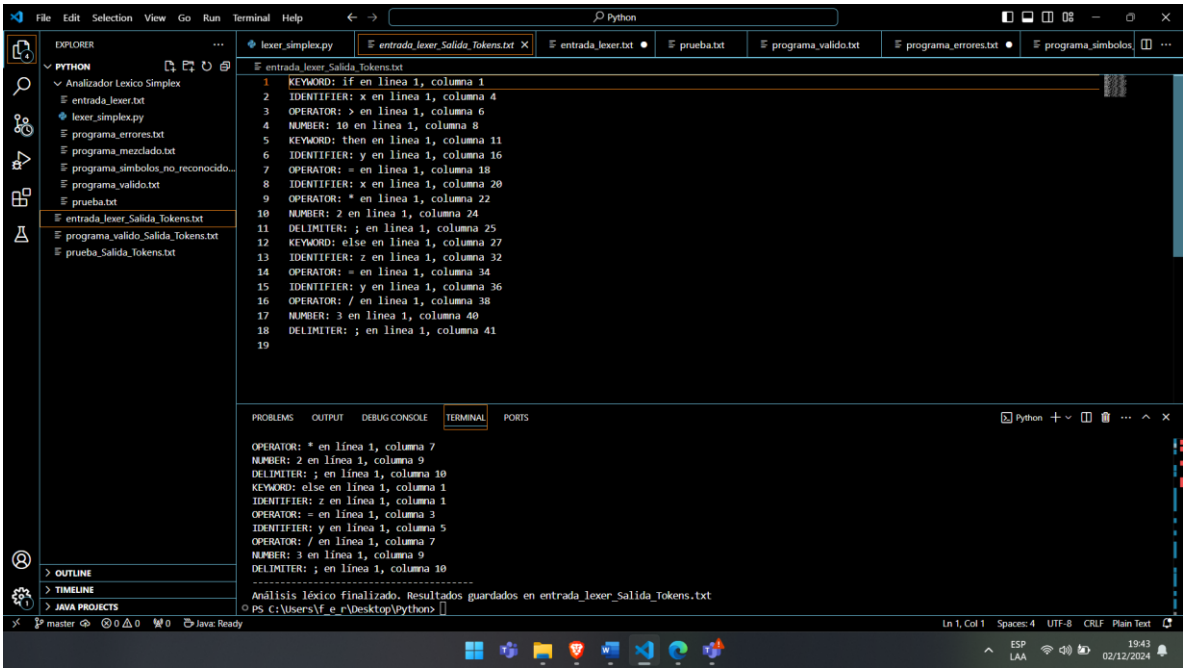
Dada la entrada:



Ejecucion:



Salida:


File Edit Selection View Go Run Terminal Help
Python
EXPLORER
PYTHON
Analizador Lexico Simplex
entrada_lexer.txt
lexer_simplex.py
programa_errores.txt
programa_mezclado.txt
programa_simbolos_no_reconocido...
programa_valido.txt
prueba.txt
entrada_lexer_Salida_Tokens.txt
programa_valido_Salida_Tokens.txt
prueba_Salida_Tokens.txt
lexer_simplex.py
entrada_lexer_Salida_Tokens.txt
entrada_lexer.txt
prueba.txt
programa_valido.txt
programa_errores.txt
programa_simbolos
entrada_lexer_Salida_Tokens.txt
1 KEYWORD: if en linea 1, columna 1
2 IDENTIFIER: x en linea 1, columna 4
3 OPERATOR: > en linea 1, columna 6
4 NUMBER: 10 en linea 1, columna 8
5 KEYWORD: then en linea 1, columna 11
6 IDENTIFIER: y en linea 1, columna 16
7 OPERATOR: = en linea 1, columna 18
8 IDENTIFIER: x en linea 1, columna 20
9 OPERATOR: * en linea 1, columna 22
10 NUMBER: 2 en linea 1, columna 24
11 DELIMITER: ; en linea 1, columna 25
12 KEYWORD: else en linea 1, columna 27
13 IDENTIFIER: z en linea 1, columna 32
14 OPERATOR: = en linea 1, columna 34
15 IDENTIFIER: y en linea 1, columna 36
16 OPERATOR: / en linea 1, columna 38
17 NUMBER: 3 en linea 1, columna 40
18 DELIMITER: ; en linea 1, columna 41
19
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
OPERATOR: * en linea 1, columna 7
NUMBER: 2 en linea 1, columna 9
DELIMITER: ; en linea 1, columna 10
KEYWORD: else en linea 1, columna 1
IDENTIFIER: z en linea 1, columna 1
OPERATOR: = en linea 1, columna 3
IDENTIFIER: y en linea 1, columna 5
OPERATOR: / en linea 1, columna 7
NUMBER: 3 en linea 1, columna 9
DELIMITER: ; en linea 1, columna 10
Análisis léxico finalizado. Resultados guardados en entrada_lexer_Salida_Tokens.txt
PS C:\Users\f_e_r\Desktop\Python>
OUTLINE
TIMELINE
JAVA PROJECTS
master
Java: Ready
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Plain Text
ESP LAA
19:43 02/12/2024