INSTITUTO TECNOLOGICO DE PACHUCA

"El hombre alimenta el ingenio en contacto con la Ciencia"

Proyecto Analizador Léxico PyEspañol

INGENIERIA EN SISTEMAS COMPUTACIONALES

NOMBRE DE LA ASIGNATURA

SISTEMAS OPERATIVOS

INTEGRANTES DEL EQUIPO

Martin Feria Vázquez

21200594

Ramírez Hernández Josué

21200990

Valdez Zuñiga Leonardo Vicente

24200196

Zeron López Germán Eduardo

21200642

PROFESOR DE LA MATERIA

Ing. Rodolfo Baumé Lazcano

PACHUCA, HIDALGO, 14 DE MAYO DEL 2024

Índice

Propósito: 3
¿Qué es un analizador léxico? 3
1. Definición de Tokens 4
1.1 Definición del lenguaje (PyEspañol): 4
1.2 Especificación de tokens para PyEspañol: 4
1.3 Patrones regulares para cada tipo de token (en formato regex): 4
Tabla de Tokens: 5
2. Expresiones regulares 6
2.1 Tabla de Tokens con sus expresiones regulares y breve: 6
3. Manejo de Espacios en Blanco y Comentarios 10
4. Prioridad de Coincidencia 10
5. Acciones Asociadas a los Tokens 11
6. Manejo de errores: 11
Documentación del código: 12
Metadatos del Documento: 12
Estructura del Cuerpo 12
Código PyScript 13
..... 13
Código JavaScript 14
Conclusión: 15
Referencias: 15

Índice de Imagenes

Imagen 1 Imagen ejemplo sacado de la Referencia 1 3
Imagen 2 Ejemplo sacado de la Referencia 2 3
Imagen 3 Compilador con prueba de su funcionamiento con las expresiones regulares 9
Imagen 4 Muestra en codigo del manejo de espacios en blanco y comentarios 10
Imagen 5 Muestra en la interface del manejo de espacios en blanco y comentarios 10
Imagen 6 Muestra en el caso de uso del analizador léxico 11
Imagen 7 El analizador léxico no solo identifica los tokens sino que también realiza las tareas necesarias para mantener el estado del análisis. 11
Imagen 8 Manejo de errores en código 11
Imagen 9 Manejo de errores aplicado en el analizador léxico 12

Propósito:

El propósito de este trabajo es desarrollar un analizador léxico que identifique y categorice diferentes componentes léxicos (tokens) en un lenguaje de programación en español. Este enfoque puede ser beneficioso en entornos educativos y comunidades de programación hispanohablantes. Python es elegido como base debido a su simplicidad y legibilidad, lo que facilita la implementación y comprensión del analizador.

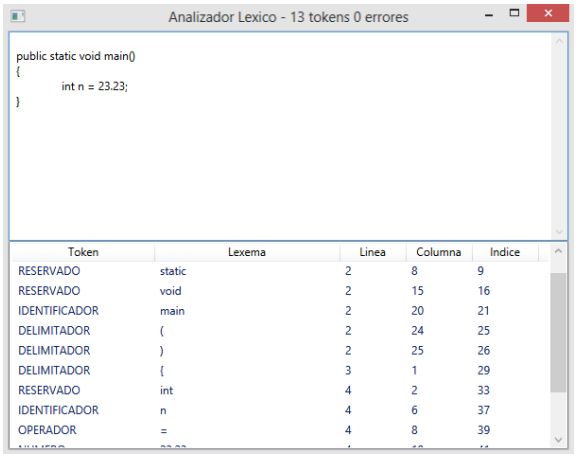

Imagen 1 Imagen ejemplo sacado de la Referencia 1

¿Qué es un analizador léxico?

El analizador léxico es un componente fundamental en el proceso de compilación o interpretación de un lenguaje de programación. Toma como entrada un texto y lo descompone en una secuencia de tokens basados en patrones definidos. Aquí se detallan todos los pasos y se documenta exhaustivamente cada componente.

Por ejemplo, en el código x = 42, x es un identificador, = es un operador y 42 es un número.

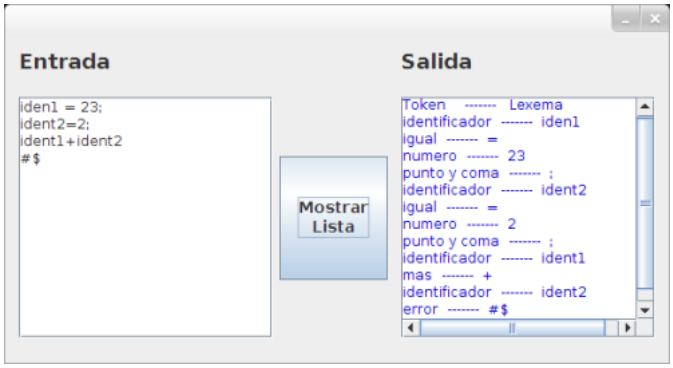

Imagen 2 Ejemplo sacado de la Referencia 2

1. Definición de Tokens

1.1 Definición del lenguaje (PyEspañol):

- Palabras clave: "si", "sino", "mientras", "para", "función", "retorno", "clase", "importar", "verdadero", "falso"

- Identificadores: Secuencias de letras y números que comienzan con una letra (Variables).

- Números: Secuencias de dígitos enteros (0-9).

- Operadores: "+", "-", "*", "/", "=", "==", "<", ">"

- Símbolos especiales: "(", ")", "{", "}", ";"

- Comentarios: Se pueden denotar con "#" para comentarios de una sola línea.

1.2 Especificación de tokens para PyEspañol:

- Palabras clave: si|sino|mientras|para|función|retorno|clase|importar|verdadero|falso -

Identificadores: [a-zA-Z][a-zA-Z0-9]*

- Números: \d+(\.\d+)?

- Operadores: \+ | - | * | / | = | == | < | >

- Símbolos especiales: \(| \) | { | } | ;

- Comentarios: #.*

1.3 Patrones regulares para cada tipo de token (en formato regex):

- Palabras clave: si|sino|mientras|para|función|retorno|clase|importar|verdadero|falso -

Identificadores: [a-zA-Z][a-zA-Z0-9]*

- Números: \d+(\.\d+)?

- Operadores: \+ | - | * | / | = | == | < | >

- Símbolos especiales: \(| \) | { | } | ;

- Comentarios: #.*

Tabla de Tokens:

TOKEN	Tipo
Si	Palabra Clave
Sino	Palabra Clave
Mientras	Palabra Clave
Para	Palabra Clave
Función	Palabra Clave
Retorno	Palabra Clave
Clase	Palabra Clave
Importar	Palabra Clave
Verdadero	Palabra Clave
Falso	Palabra Clave
Variable	Identificador
1-9	Numero
+	Operador
-	Operador
*****	Operador
/	Operador
=	Operador
==	Operador
<	Operador
>	Operador
(	Símbolo especial
)	Símbolo especial
{	Símbolo especial
}	Símbolo especial
;	Símbolo especial
#	Comentario

2. Expresiones regulares

2.1 Tabla de Tokens con sus expresiones regulares y breve:

Hemos definido una serie de tokens que nuestro analizador léxico debe reconocer. Estos tokens se clasifican en diferentes categorías: palabras clave, identificadores, números, operadores, símbolos especiales y comentarios. A continuación se presenta la tabla de tokens con su tipo y expresión regular correspondiente.

Token	Tipo	Expresión Regular	Explicación
`si`	PALABRA_CLAVE	`\bsi\b`	Coincide con "si", utilizado para condicionales (equivalente a "if").
`sino`	PALABRA_CLAVE	`\bsino\b`	Coincide con "sino", utilizado para condicionales alternativos (equivalente a "else").
`mientras`	PALABRA_CLAVE	`\bmientras\b`	Coincide con "mientras", utilizado para bucles (equivalente a "while").
`para`	PALABRA_CLAVE	`\bpara\b`	Coincide con "para", utilizado para bucles (equivalente a "for").
`función`	PALABRA_CLAVE	`\bfunción\b`	Coincide con "función", utilizado para definir funciones (equivalente a "def").
`retorno`	PALABRA_CLAVE	`\bretorno\b`	Coincide con "retorno", utilizado para devolver valores de funciones (equivalente a "return").
`clase`	PALABRA_CLAVE	`\bclase\b`	Coincide con "clase", utilizado para definir clases (equivalente a "class").
`importar`	PALABRA_CLAVE	`\bimportar\b`	Coincide con "importar", utilizado para importar módulos (equivalente a "import").
`verdadero`	PALABRA_CLAVE	`\bverdadero\b`	Coincide con "verdadero", valor booleano verdadero (equivalente a "true").
`falso`	PALABRA_CLAVE	`\bfalso\b`	Coincide con "falso", valor booleano falso (equivalente a "false").
Identificador	IDENTIFICADOR	`[a-zA-Z_][a-zA-Z0-9_]*`	Coincide con nombres de variables y funciones.
Número	NUMERO	`\d+(\.\d+)?`	Coincide con números enteros y decimales.
Operadores	OPERADOR	`==`	!=
Símbolos especiales	SIMBOLO_ESPECIAL	`[{}();,]`	Coincide con símbolos de puntuación.
Comentario	COMENTARIO	`#.*`	Coincide con comentarios.

Espacios en blanco	None	`\s+`	Coincide con espacios en blanco.

2.11 Palabras Clave

Las palabras clave son términos reservados en el lenguaje que no pueden usarse como identificadores.

Token	Tipo	Expresión Regular
`si`	PALABRA_CLAVE	`\bsi\b`
`sino`	PALABRA_CLAVE	`\bsino\b`
`mientras`	PALABRA_CLAVE	`\bmientras\b`
`para`	PALABRA_CLAVE	`\bpara\b`
`función`	PALABRA_CLAVE	`\bfunción\b`
`retorno`	PALABRA_CLAVE	`\bretorno\b`
`clase`	PALABRA_CLAVE	`\bclase\b`
`importar`	PALABRA_CLAVE	`\bimportar\b`
`verdadero`	PALABRA_CLAVE	`\bverdadero\b`
`falso`	PALABRA_CLAVE	`\bfalso\b`

La expresión regular `\bpalabra\b` asegura que la palabra clave no sea parte de otra palabra, delimitada por límites de palabra (`\b`).

2.2 Identificadores

Los identificadores son nombres para variables, funciones y otros elementos definidos por el usuario. Deben comenzar con una letra o un guion bajo y pueden contener letras, números y guiones bajos.

Token	Tipo	Expresión Regular
Identificador	IDENTIFICADOR	`[a-zA-Z_][a-zA-Z0-9_]*`

2.3 Números

Los números pueden ser enteros o decimales. La expresión regular captura ambos formatos.

Token	Tipo	Expresión Regular
Número	NUMERO	`\d+(\.\d+)?`

La expresión regular `\d+(\.\d+)?` permite coincidir con números enteros (`\d+`) y decimales (`\d+\.\d+`).

2.4 Operadores

Los operadores realizan operaciones sobre los operandos. Incluimos operadores aritméticos y de comparación.

Token	Tipo	Expresión Regular
Operadores	OPERADOR	`==

2.5 Símbolos Especiales

Los símbolos especiales incluyen caracteres como paréntesis, llaves, punto y coma, y comas.

Token	Tipo	Expresión Regular
Símbolos especiales	SIMBOLO_ESPECIAL	`[{}();,]`

2.6 Comentarios

Los comentarios son ignorados por el compilador o intérprete y se utilizan para anotaciones.

Token	Tipo	Expresión Regular
Comentario	COMENTARIO	`#.*`

La expresión regular `#.*` captura cualquier cosa desde el símbolo `#` hasta el final de la línea.

2.7 Espacios en Blanco

Los espacios en blanco se ignoran durante el análisis léxico, pero son necesarios para separar tokens.

Token	Tipo	Expresión Regular
Espacios en blanco	None	`\s+`

Imagen 3 Compilador con prueba de su funcionamiento con las expresiones regulares

3. Manejo de Espacios en Blanco y Comentarios

Los espacios en blanco son importantes para la legibilidad del código pero no tienen significado semántico en muchos lenguajes de programación, por lo que se ignoran en el proceso de tokenización. Los comentarios se reconocen pero se pueden manejar de manera diferente según las necesidades del analizador (por ejemplo, se pueden ignorar, conservar para documentación, etc.).

En el código:

```
(r'#.*', 'COMENTARIO'),
(r'\s+', None), # Ignorar espacios en blanco
```

Imagen 4 Muestra en codigo del manejo de espacios en blanco y comentarios

En la interface:

Imagen 5 Muestra en la interface del manejo de espacios en blanco y comentarios

4. Prioridad de Coincidencia

Para resolver conflictos cuando una secuencia de caracteres puede corresponder a múltiples tokens, el orden de los patrones en token_patterns determina la prioridad. Los patrones definidos primero tienen prioridad sobre los posteriores.

Imagen 6 Muestra en el caso de uso del analizador léxico

5. Acciones Asociadas a los Tokens

Cada vez que se reconoce un token, se realiza la siguiente acción:

- Si el tipo de token no es None (por ejemplo, espacios en blanco son ignorados), se añade una tupla a la lista de tokens, que contiene el texto coincidente y su tipo.
- Se actualiza el texto eliminando la parte coincidente.

Imagen 7 El analizador léxico no solo identifica los tokens sino que también realiza las tareas necesarias para mantener el estado del análisis.

6. Manejo de errores:

Si se encuentra un carácter o secuencia que no coincide con ningún patrón de token, se genera un mensaje de error específico. Esto permite a los desarrolladores identificar y corregir errores en el código fuente de manera eficiente.

En código:

```
        if not match:
            raise ValueError(f'Error léxico: {texto[0]} en "{texto}"')
    return tokens
```

Imagen 8 Manejo de errores en código

En el analizador léxico:

Imagen 9 Manejo de errores aplicado en el analizador léxico

Documentación del código:

Este código define y ejecuta un analizador léxico utilizando PyScript, HTML, JavaScript y CSS. A continuación, se explica detalladamente el propósito de cada parte del código y cómo funciona en conjunto, exceptuando el código CSS el cual es solo para diseño y no tiene una función practica en el código para el analizador léxico.

Metadatos del Documento:

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Analizador Léxico</title>
  <script defer src="https://pyscript.net/alpha/pyscript.js"></script>
  <link rel="stylesheet" href="style.css">
</head>
```

- Define la codificación de caracteres, la configuración de visualización y el título de la página.
- Importa PyScript y un archivo de estilos CSS.

Estructura del Cuerpo

```
<body>
  <h1>Analizador Léxico</h1>
  <div class="container">
    <div class="input-container">
      <h2>Entrada</h2>
      <textarea id="input-text" rows="10" style="width: 100%;">si x == 10 { imprimir(x); }</textarea>
      <div class="button-container">
        <button id="analyze-button" onclick="ejecutarScript()" class="ui-btn"><span>Analizar</span></button>
      </div>
    </div>
    <div class="output-container">
```

- Contiene la interfaz de usuario con áreas de texto para la entrada y la salida y un botón para iniciar el análisis.

Código PyScript

```
<py-script>
import re
# Definición de los patrones de tokens con sus respectivos tipos
token_patterns = [
    (r'\b(si|sino|mientras|para|función|retorno|clase|importar|verdadero|falso)\b', 'PALABRA_CLAVE'),  # Palabras clave del lenguaje
    (r'[a-zA-Z_][a-zA-Z0-9_]*', 'IDENTIFICADOR'),  # Identificadores válidos
    (r'\d+(\.\d+)?', 'NUMERO'),  # Números enteros y de punto flotante
    (r'==|!=|<=|>=|<|>|=|\+|\-|\*|/', 'OPERADOR'),  # Operadores aritméticos y de comparación
    (r'[{}();,]', 'SIMBOLO_ESPECIAL'),  # Símbolos especiales
    (r'#.*', 'COMENTARIO'),  # Comentarios de una sola línea
    (r'\s+', None),  # Espacios en blanco (ignorados)
]
# Función principal del analizador léxico
def analizador_lexico(texto):
    """
    Analiza el texto de entrada y lo descompone en una lista de tokens.
    Args:
        texto (str): El código fuente a analizar.
    Returns:
        list: Una lista de tuplas, donde cada tupla contiene un token y su tipo.
    Raises:
        ValueError: Si se encuentra un carácter inesperado en el texto.
    """
    tokens = []  # Lista para almacenar los tokens reconocidos
    while texto:  # Mientras haya texto por analizar
        match = None
        for token_regex, token_tipo in token_patterns:
            regex = re.compile(token_regex)  # Compilar la expresión regular
            match = regex.match(texto)  # Intentar hacer coincidir el patrón con el inicio del texto
            if match:
                if token_tipo:  # Si el tipo de token no es None, añadir a la lista de tokens
                    tokens.append((match.group(0), token_tipo))
                texto = texto[match.end():]  # Eliminar la parte coincidente del texto
                break
        if not match:  # Si no se encuentra coincidencia, lanzar un error léxico
            raise ValueError(f'Error léxico: carácter inesperado "{texto[0]}" en "{texto}"')
    return tokens
```

```
# Registrar la función de procesamiento en el objeto `window` de JavaScript
from js import window
window.procesarEntrada = procesar_entrada
</py-script>
```

- token_patterns: Define las expresiones regulares y sus tipos de tokens.
- analizador_lexico: Analiza el texto de entrada, reconoce tokens y maneja errores léxicos.
- procesar_entrada: Convierte el texto de entrada en una lista de tokens o en un mensaje de error si ocurre un error léxico.
- Registro de la función: Permite que la función procesar_entrada se llame desde JavaScript.

Código JavaScript

```
<script>
  function ejecutarScript() {
    // Obtener el texto de entrada del área de texto
    const inputText = document.getElementById('input-text').value;
    // Llamar a la función Python para procesar el texto y obtener los tokens
    const outputText = window.procesarEntrada(inputText);
    // Actualizar el área de texto de salida con los tokens generados
    document.getElementById('output-text').value = outputText;
  }
</script>
```

ejecutarScript:

- Obtiene el texto de entrada.
- Llama a la función Python registrada procesarEntrada.
- Muestra el resultado en el área de texto de salida.

Conclusión:

La implementación del analizador léxico en Python proporciona una sólida base para el reconocimiento de tokens en un lenguaje de programación en español. Utilizando expresiones regulares, se pueden identificar palabras clave, identificadores, números, operadores, símbolos especiales y comentarios, lo que permite analizar un código fuente y convertirlo en una secuencia de tokens para su posterior procesamiento por un analizador sintáctico y semántico.

Referencias:

- Tutor de programación. (2014, February 20). Analizador Léxico. Blogspot.com; Blogger. https://acodigo.blogspot.com/2014/02/analizador-lexico.html
- Analizador Léxico: Implementación en Java. (2013, July 23). Learnercys; learnercys. https://learnercys.wordpress.com/2013/07/23/analizador-lexico-implementacion-en-java/
- *ChatGPT*. (2024). Chatgpt.com. https://chatgpt.com/c/57a69674-e7b0-4289-9dbb-ecff472ed63c