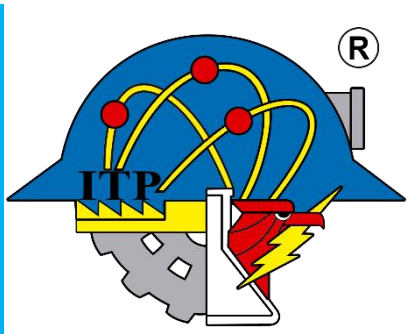


Tecnológico Nacional de México

INSTITUTO TECNOLÓGICO DE PACHUCA

Documentación del Analizador Léxico y Sintáctico

INGENIERIA EN SISTEMAS COMPUTACIONALES

NOMBRE DE LA ASIGNATURA

Lenguajes y Autómatas I

Docente:

Rodolfo Baumé Lazcano

Alumnos:

García Lugo Itzel Paola

Quintos Cabeza Diego

Vera Rodríguez Héctor Gabriel

Fecha

30/05/2024

Introducción

Este documento describe el diseño e implementación de un analizador léxico y sintáctico para expresiones aritméticas simples. El analizador es capaz de procesar expresiones que incluyen números enteros y de punto flotante, operadores aritméticos básicos (+, -, *, /) y paréntesis. Además, el analizador puede identificar y reportar errores léxicos y sintácticos.

Diseño del Analizador

Analizador Léxico (Lexer)

El analizador léxico se encarga de dividir la entrada en una secuencia de tokens. Los tokens son definidos mediante expresiones regulares y pueden representar números, operadores, paréntesis, espacios y caracteres inesperados. El Lexer genera una lista de tokens que se utilizarán posteriormente en el análisis sintáctico.

Especificación de Tokens

NUMBER: Números enteros o de punto flotante (\d+(\.\d*)?)

ADD: Operador de suma (\+)

SUB: Operador de resta (-)

MUL: Operador de multiplicación (*)

DIV: Operador de división (/)

LPAREN: Paréntesis izquierdo (\()

RPAREN: Paréntesis derecho (\))

SKIP: Espacios y tabuladores ([\t]+)

MISMATCH: Cualquier otro carácter (.)

Clase Lexer

La clase Lexer tiene las siguientes responsabilidades:

- tokenize: Convierte la cadena de entrada en una lista de tokens utilizando las expresiones regulares definidas.
- next_token: Devuelve el siguiente token de la lista de tokens.

Analizador Sintáctico (Parser)

El analizador sintáctico utiliza un enfoque de análisis descendente predictivo para evaluar las expresiones aritméticas. Las principales funciones del analizador sintáctico son:

- factor: Maneja números y expresiones entre paréntesis.
- term: Maneja multiplicación y división.
- expr: Maneja suma y resta.

Clase Parser:

La clase Parser tiene las siguientes responsabilidades:

- eat: Verifica y consume el token esperado, avanzando al siguiente token. Si el token actual no es el esperado, lanza un SyntaxError.
- factor: Procesa un número o una expresión entre paréntesis.
- term: Procesa términos que pueden incluir operadores de multiplicación y división.
- expr: Procesa expresiones que pueden incluir operadores de suma y resta.
- parse: Inicia el análisis sintáctico y devuelve el resultado de la evaluación de la expresión.

Casos de Prueba

Expresión válida con números enteros:

Entrada: "60 / 3 (60-40)"

Salida esperada: Resultado del análisis sintáctico: 400

Expresión válida con números de punto flotante:

Entrada: "89.72 + 67.56 * (4 /10)"

Salida esperada: Resultado del análisis sintáctico: 116.744

Error léxico (carácter inesperado):

Entrada: "389 + 504 a"

Salida esperada: Error de runtime: Error léxico: Carácter inesperado 'a' en posición 11

Error sintáctico (token inesperado):

Entrada: "3 + 5 * (10 -)"

Salida esperada: Error de sintaxis: Error sintáctico: Se esperaba NUMBER pero se encontró RPAREN en la posición [('RPAREN', ')')]

Expresión con espacios y tabuladores:

Entrada: " 3 + 5 * (10 - 4)"

Salida esperada: Resultado del análisis sintáctico: 33

GNU Octave (GUI).lnk | lenguaje.py | sintactico.py

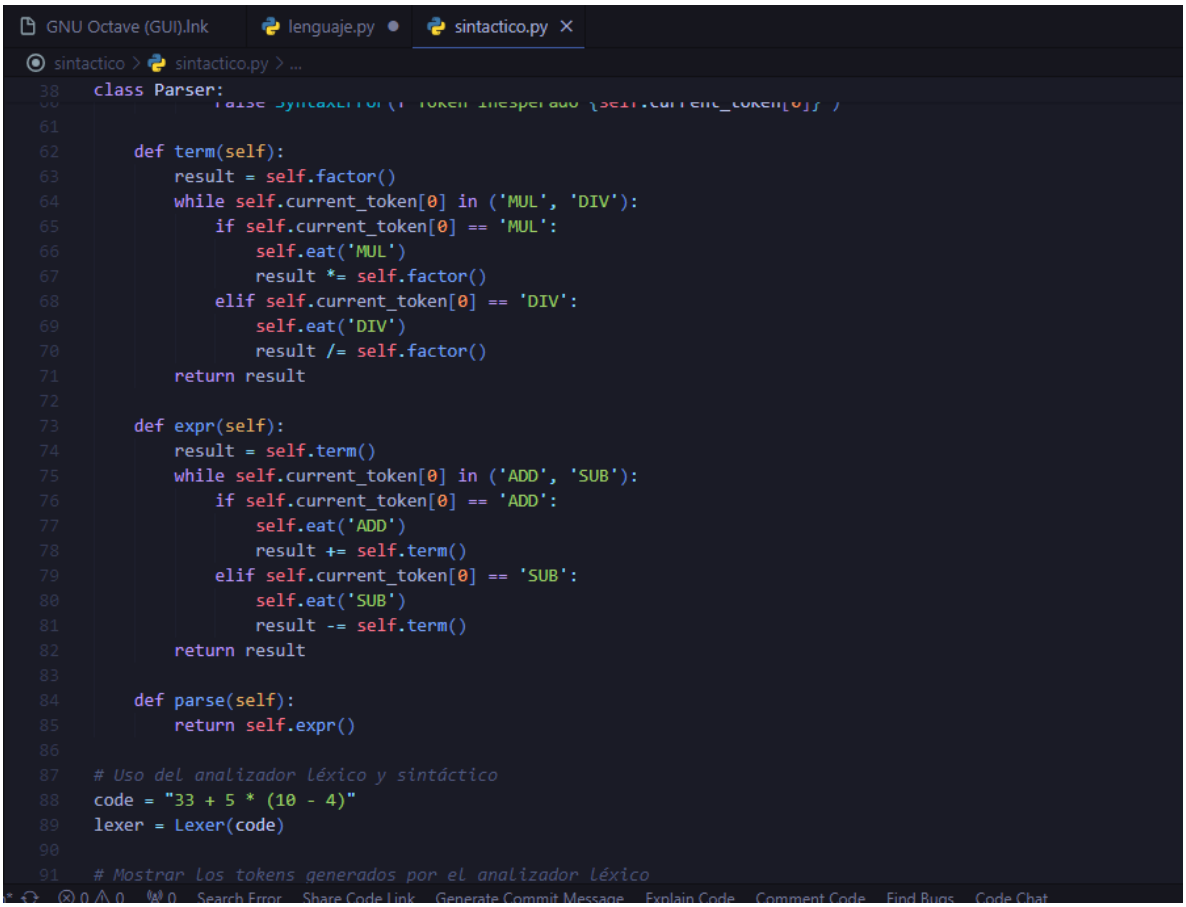
sintactico > sintactico.py > ...

```python
import re

# Definición de tokens
token_specification = [
    ('NUMBER',   r'\d+(\.\d*)?'),  # Números enteros o de punto flotante
    ('ADD',      r'\+'),           # Suma
    ('SUB',      r'-'),            # Resta
    ('MUL',      r'\*'),           # Multiplicación
    ('DIV',      r'/'),            # División
    ('LPAREN',   r'\('),           # Paréntesis izquierdo
    ('RPAREN',   r'\)'),           # Paréntesis derecho
    ('SKIP',     r'[ \t]+'),       # Espacios y tabuladores
    ('MISMATCH', r'.'),            # Cualquier otro carácter
]

tok_regex = '|'.join('(?P<%s>%s)' % pair for pair in token_specification)

class Lexer:
    def _init_(self, code):
        self.tokens = []
        self.tokenize(code)

    def tokenize(self, code):
        for mo in re.finditer(tok_regex, code):
            kind = mo.lastgroup
            value = mo.group()
            if kind == 'NUMBER':
                value = float(value) if '.' in value else int(value)
            elif kind == 'SKIP':
                continue
            elif kind == 'MISMATCH':
                raise RuntimeError(f'{value!r} inesperado en {mo.start()}')
```

GNU Octave (GUI).lnk | lenguaje.py | sintactico.py

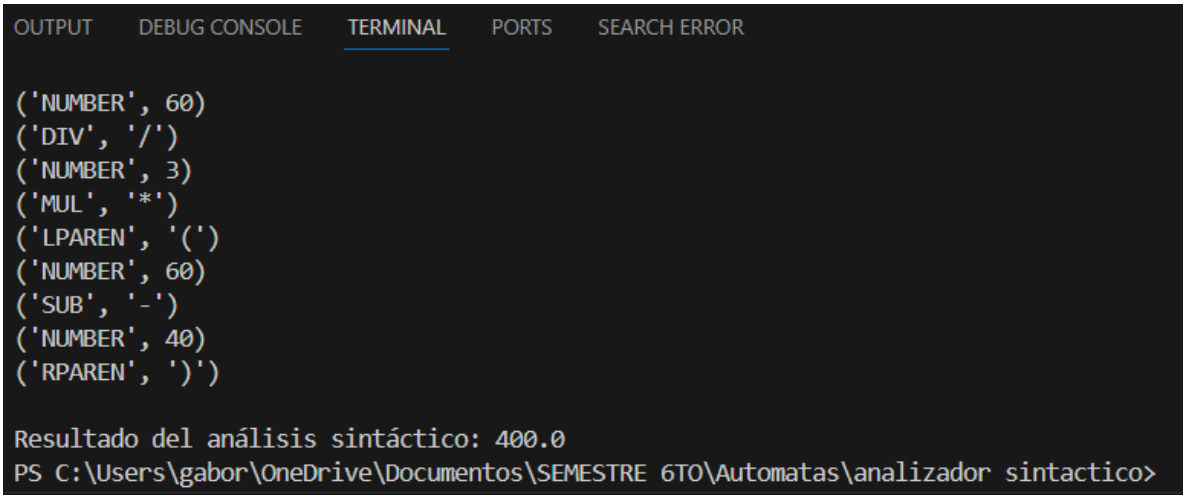
sintactico > sintactico.py > ...

```python
class Lexer:
    def tokenize(self, code):
                raise RuntimeError(f'{value!r} inesperado en {mo.start()}')
            self.tokens.append((kind, value))

    def next_token(self):
        return self.tokens.pop(0) if self.tokens else ('EOF', None)

class Parser:
    def _init_(self, lexer):
        self.lexer = lexer
        self.current_token = lexer.next_token()

    def eat(self, token_type):
        if self.current_token[0] == token_type:
            self.current_token = self.lexer.next_token()
        else:
            raise SyntaxError(f"Se esperaba {token_type} pero se encontró {self.current_token[0]}")

    def factor(self):
        if self.current_token[0] == 'NUMBER':
            value = self.current_token[1]
            self.eat('NUMBER')
            return value
        elif self.current_token[0] == 'LPAREN':
            self.eat('LPAREN')
            result = self.expr()
            self.eat('RPAREN')
            return result
        else:
            raise SyntaxError(f"Token inesperado {self.current_token[0]}")
```

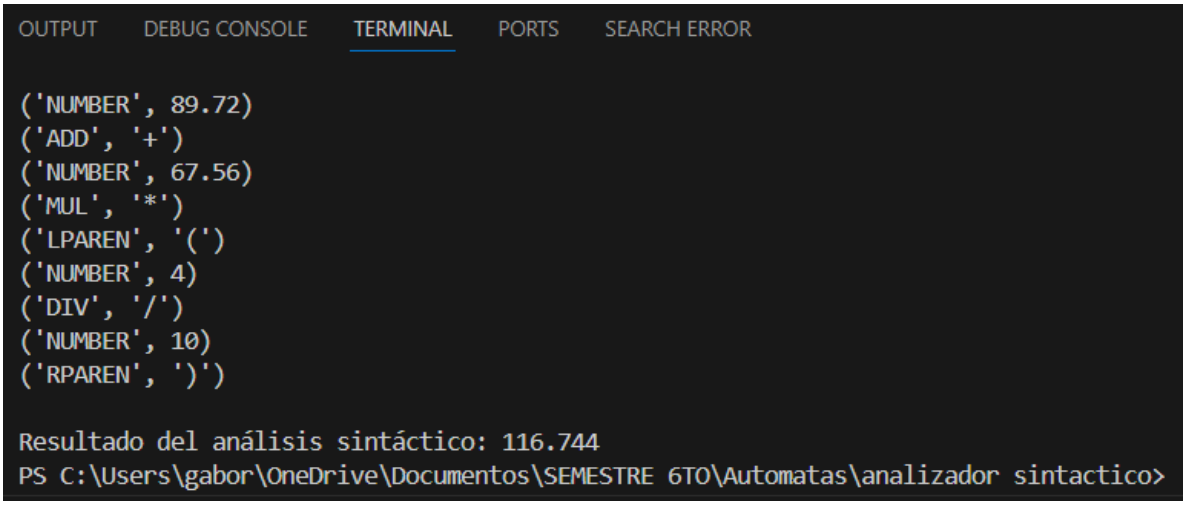



```python
# Uso del analizador léxico y sintáctico
code = "33 + 5 * (10 - 4)"
lexer = Lexer(code)

# Mostrar los tokens generados por el analizador léxico
print("Tokens generados por el analizador léxico:")
for token in lexer.tokens:
    print(token)

parser = Parser(lexer)

try:
    result = parser.parse()
    print(f"\nResultado del análisis sintáctico: {result}")
except SyntaxError as e:
    print(f"Error de sintaxis: {e}")
except RuntimeError as e:
    print(f"Error de runtime: {e}")
```

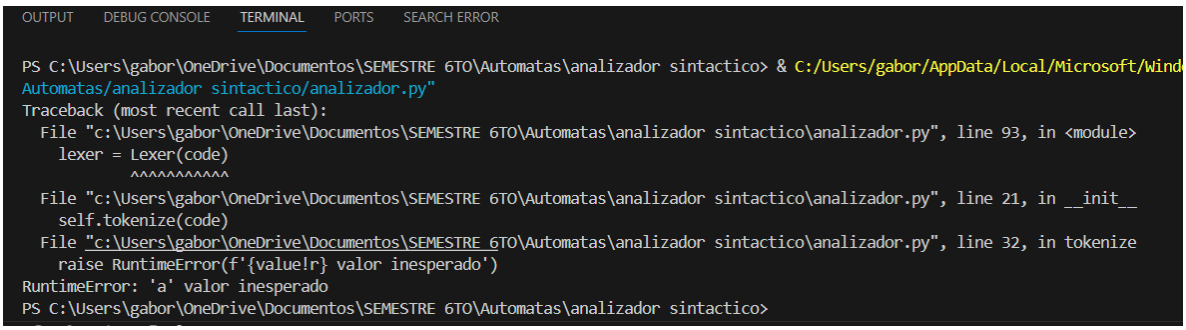
Caso de prueba 1.



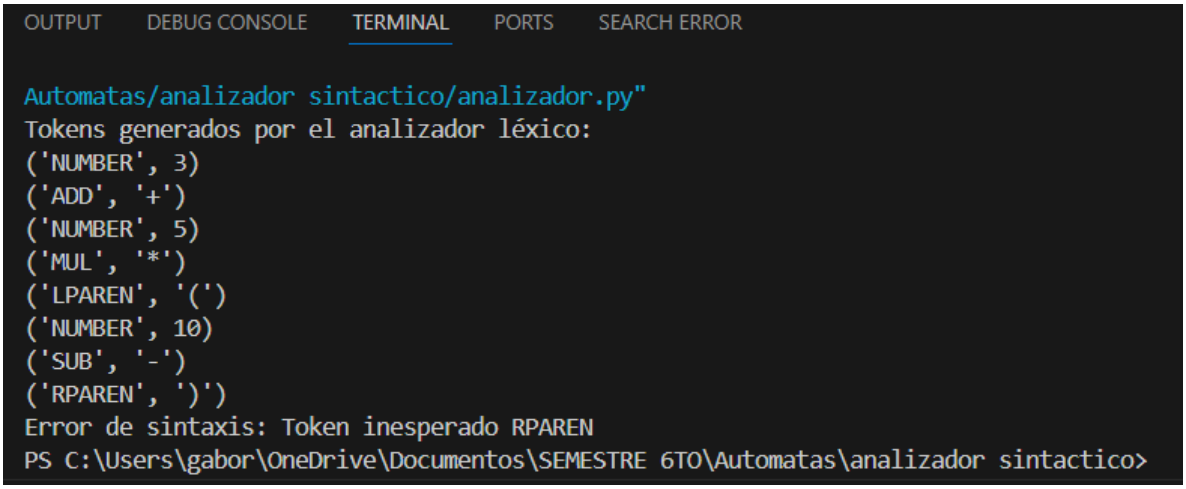
Caso de prueba 2.



Caso de prueba 3.



Caso de prueba 4.



Caso de prueba 5.