Noto Serif CJK JP
spanish CMU Serif

Implementacion del analizador lexico

1. Analisis del código
   1.1) Definición de los tokens

```
tokens = (
    'CUANDO', 'MIENTRAS', 'SI', 'SI_NO', 'IMPRIMIR', 'ASIGNAR', 'O_SI', 'TAMBIEN',
    'ENTERO', 'FLOTANTE', 'DOBLE', 'BOOLEANO', 'CADENA_CARACTER', 'CARACTER',
    'SUMA', 'MENOS', 'MULTIPLICACION', 'DIVISION',
    'MAYOR', 'MENOR', 'MENOR_O_IGUAL', 'MAYOR_O_IGUAL',
    'IGUAL_COMPARACION', 'DESIGUALDAD',
    'COMENTARIO_DE_BLOQUE', 'PARENTESIS_INICIO', 'PARENTESIS_FIN', 'SALTO_LINEA'
)
```

1.2) Las expresiones regulares para los tokens simples como operadores aritméticos (+ , -, , /) *y comparaciones* ($>, <, <=, >=, ==$ *! !=)*
t_SUMA $= r'\ + '$
t_MENOS $= r' - '$
t_MULTIPLICACION $= r'\ ^*$
t_DIVISION $= r'\ /'$
t_MAYOR $= r'\ >'$
t_MENOR $= r'\ <'$
t_MENOR_0_IGUAL $= r'\ <='$
t_MAYOR_0_IGUAL $= r'\ >='$
t_IGUAL_COMPARACION $= r'\ ==' $
t_DESIGUALDAD $= r'! ='$

1.3)$Sedefinelareglapararreconoceryomitircomentariosdebloquedelimitadospor/y*$
/.
$def t\_COMENTARIO\_DE\_BLOQUE(t):$
$r'/\ ^*.*?\ ?\ '/'$
pass # Los comentarios de bloque se ignoran
1.4) Se definen reglas para palabras clave específicas del lenguaje.

```
def t_CUANDO(t):
    r'cuando'
    return t

def t_MIENTRAS(t):
    r'mientras'
    return t

def t_SI(t):
    r'si'
    return t

def t_SI_NO(t):
    r'si_no'
    return t

def t_IMPRIMIR(t):
    r'imprimir'
    return t

def t_ASIGNAR(t):
    r'asignar'
    return t

def t_O_SI(t):
    r'o_si'
    return t

def t_TAMBIEN(t):
    r'tambien'
    return t

def t_RETORNAR(t):
    r'retorno'
    return t
```

1.5) Se definen reglas para los tipos de datos (ENTERO, FLOTANTE, DOBLE, BOOLEANO, CADENA_CARACTER, CARACTER).

```
def t_ENTERO(t):
    r'\b\d+\b'
    t.value = int(t.value)  # Convertir el valor a entero
    return t

def t_FLOTANTE(t):
    r'\b\d*\.\d+\b'
    t.value = float(t.value)  # Convertir el valor a flotante
    return t

def t_DOBLE(t):
    r'\b\d*\.\d+([eE][-+]?\d+)?\b'  # Número en notación científica
    t.value = float(t.value)  # Convertir el valor a doble precisión
    return t

def t_BOOLEANO(t):
    r'\bverdadero\b|\bfalso\b'
    t.value = (t.value == 'verdadero')  # Convertir a booleano
    return t

def t_CADENA_CARACTER(t):
    r'\".*?\"'
    t.value = t.value.strip('"')  # Eliminar comillas alrededor
    return t

def t_CARACTER(t):
    r'\'.\''
    t.value = t.value.strip('\'')  # Eliminar comillas alrededor
    return t
```

1.6) La regla para SALTO_LINEA es usada para incrementar el número de línea en el lexer cuando se encuentra la palabra finL.
def t_SALTO_LINEA (t) :
r' finL'
t.lexer.lineno t= 1 # Incrementa el número de línea en 1 return **t**
1.7) Construcción del Lexer: Se construye el lexer con lex.lex().
1.7.1)Prueba del Lexer: Se define un conjunto de datos de prueba, se alimenta al lexer y se imprimen los tokens reconocidos.

```
lexer = lex.lex()

if __name__ == "__main__":
    data = '''
    cuando a < 10
    mientras a > 5
    si a == 10
    imprimir "El valor de a es 10"
    si_no
    asignar a 20
    o_si a < 10
    tambien
    retorno 3.14
    finL
    /* Este es un comentario de bloque */
    (a + b) * c
    '''

    lexer.input(data)

    while True:
        tok = lexer.token()
        if not tok:
            break  # No más entrada
```

2) Tabal de tokens y expresiones regulares

Token	*Expresión Regular*	*Descripción*
CUANDO	*cuando*	*Palabra clave para iniciar un bloque similar a while*
MIENTRAS	*mientras*	*Palabra clave para un bucle similar a for*
SI	*si*	*Palabra clave para una condición (if)*
SI_NO	*si_no*	*Palabra clave para una alternativa (else)*
IMPRIMIR	*imprimir*	*Palabra clave para imprimir valores*
ASIGNAR	*asignar*	*Palabra clave para asignar valores a variables*
O_SI	*o_si*	*Palabra clave para una alternativa adicional (elif)*
TAMBIEN	*tambien*	*Palabra clave para operaciones adicionales*
RETORNAR	*retorno*	*Palabra clave para retornar valores*
FINL	*finL*	*Palabra clave para el fin de una línea o bloque*
ENTERO	$[0-9]^*$	*Tipo de dato entero*
FLOTANTE	$[0-9]^* + () + .[0-9]^*$	*Tipo de dato flotante*
DOBLE	$[0-9] * +() + .[0-9]*$	*Tipo de dato de doble precisión*
BOOLEANO	*[verdad — falso]*	*\bverdad \bfalso‘*
CADENA_CARACTER	$(") + [\bullet]^* + (')$	*Tipo de dato cadena de caracteres*
CARACTER	*(")+[]+(")*	*Tipo de dato carácter*
SUMA	+	*Operador de suma*
MENOS	-	*Operador de resta*
MULTIPLICACION	*	*Operador de multiplicación*
DIVISION	*1*	*Operador de división*
MAYOR	>	*Operador de mayor que*
MENOR	<	*Operador de menor que*
MENOR_O_IGUAL	<=	*Operador de menor o igual que*
MAYOR_O_IGUAL	=>	*Operador de mayor o igual que*
IGUAL_COMPARACION	==	*Operador de igualdad*

DESIGUALDAD	! =	*Operador de desigualdad*
COMENTARIO_DE_BLOQUE	*(/*) + [ ]* + (*/)*	*Comentario de bloque delimitado por /* y */*
PARENTESIS_INICIO	*(*	*Paréntesis de apertura*
PARENTESIS_FIN	*)*	*Paréntesis de cierre*
SALTO_LINEA	*finL*	*Marca el final de una línea o bloque*