Universidad de Guadalajara
Centro Universitario de Ciencias Exactas E Ingenierías
División de Electrónica y Computación
Ingeniería en computación

D02 SEMINARIO DE SOLUCION DE PROBLEMAS DE TRADUCTORES DE LENGUAJES II

Mini generador léxico

216788333 Alejandro de Jesús Romo Rosales

10/09/2020

Objetivo

Genera un pequeño analizador léxico, que identifique los siguientes tokens (identificadores y números reales) construidos de la siguiente manera.

identificadores = letra(letra|digito)*

Real = entero.entero+

Introducción

Hacer un analizador léxico significa leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis.

El analizador léxico es la primera fase de un compilador.

Su principal función consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis. Esta interacción, suele aplicarse convirtiendo al analizador léxico en una subrutina o corrutina del analizador sintáctico. Recibida la orden "obtén el siguiente componente léxico" del analizador sintáctico, el analizador léxico lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico.

Metodología

Posterior a una investigación sobre el tema en cuestión, implementar seleccionar un lenguaje que soporte el paradigma orientado a objetos para así poder modelar de una manera más efectiva y eficiente la abstracción de las clases, por ejemplo, la clase Token. Una vez realizado esto se procederá a programar el analizador léxico mediante el uso de funciones y finalizando esto predefinir una serie de pruebas las cuales nos ayudaran a corroborar el su debido comportamiento.

Materiales

1. Python 3.7
2. VS Code
3. Windows 10

Desarrollo

Lo primero que haremos será definir la clase Tokens.

```
""""""""""""""""""""""""""""""""
"" TOKENS
""""""""""""""""""""""""""""""""
# TipoToken
```

```
TT_INT        = "INT"
TT_FLOAT      = "FLOAT"
TT_EQ         = "IGUAL"
TT_IDENTIFIER = "IDENTIFICADOR"
TT_ASIGN      = "ASIGNACION"

DIGIT = '0123456789'
LETTERS = 'abcdefghijklmnopqrstwxyzABCDEFGHIJKLMNOPQRSTWXYZ'

class Token:
    def __init__(self, tpe, value=None):
        self._type = tpe
        self._value = value

    def __repr__(self):
        if self._value: return f'{self._type}:{self._value}'
        return f'{self._type}'

    # Regresa el valor del Token
    def Value(self):
        if self._value != None:
            return self._value
    # Regresa el tipo del Token
    def Type(self):
        return self._type
"""""""""""""""""""""""""""""""
```

Esto nos permitirá definir los componentes de nuestro texto de entrada.

Posteriormente definiremos los tipos de errores que podremos encontrar así como la posición ya que un buen compilador siempre le hace mención al programador donde esta el error.

```
"""""""""""""""""""""""""""""""
"" ERRORES
"""""""""""""""""""""""""""""""
class Error:
    def __init__(self, pos_start, pos_end, error_name, details):
        self._pos_start = pos_start
        self._pos_end = pos_end
        self._error_name = error_name
        self._details = details

    def as_string(self):
        result  = f'{self._error_name}: {self._details}\n'
        pos = self._pos_start._ln+1
        result += f'Archivo {self._pos_start._fn}, line {pos}'
```

```
        return result

class IllegalCharError(Error):
    def __init__(self, pos_start, pos_end, details):
        super().__init__(pos_start, pos_end, 'Caracter Invalido', details)

# Error en la definicion de la variable
class IllegalVariableError(Error):
    def __init__(self, pos_start, pos_end, details):
        super().__init__(pos_start, pos_end, 'Declaración de variable invali
da ', details)

class Position:
    def __init__(self, idx, ln, col, fn, fntxt):
        self._idx = idx
        self._ln = ln
        self._col = col
        self._fn = fn
        self._fntxt = fntxt

    def advance(self, current_char):
        self._idx += 1
        self._col += 1
        if current_char == '\n':
            self._ln += 1
            self._col += 0
        return self

    def copy(self):
        return Position(self._idx, self._ln, self._col, self._fn, self._fntx
t)
"""""""""""""""""""""""""""""""""
```

Con esto hecho, definiremos la lógica de nuestro analizador, lo cual será analizar el texto de entrada para obtener tokens, o bien, errores.

```
"""""""""""""""""""""""""""""""""
"" Mini analizador léxico
"""""""""""""""""""""""""""""""""
class Lexer:
    def __init__(self, fn, text):
        self._fn = fn
        self._text = text
        self._pos = Position(-1, 0, -1, fn, text)
        self._current_char = None
```

```
        self.advance()

    def advance(self):
        self._pos.advance(self._current_char)
        self._current_char = self._text[self._pos._idx] if self._pos._idx <
len(self._text) else None

    def make_number(self):
        num_str = ''
        dot_count = 0

        while self._current_char != None and self._current_char in DIGIT + '
.':
            if self._current_char == '.':
                if dot_count == 1: break
                dot_count += 1
                num_str += '.'
            else:
                num_str += self._current_char
            self.advance()
        if dot_count == 0:
            return Token(TT_INT, int(num_str))
        else:
            return Token(TT_FLOAT, float(num_str))

    def make_identifier(self):
        id_str = ''
        pos = self._pos.copy()
        while self._current_char != None and self._current_char in LETTERS:
            id_str += self._current_char
            self.advance()
        return Token(TT_IDENTIFIER, id_str)

    def analyze(self):
        tokens = []
        count = self._text.count('=')

        if count > 1:
            pos = self._pos.copy()
            char = self._current_char
            self.advance()
            return [], IllegalVariableError(pos, self._pos, '-
> ', self._text)
        elif count == 1:
            name = self._text.split('=')[0]
```

```
            tokens.append(Token(TT_IDENTIFIER, name))
            for n in range(len(name)):
               self.advance()
            tokens.append(self.make_tokens(tokens)[0])
            return tokens, None
        else:
            tokens = self.make_tokens(tokens)[0]
            return tokens, None

    def make_tokens(self, tokens):
        # tokens = []

        while self._current_char != None:
            if self._current_char in '\t' or self._current_char == ' ':
                self.advance()
            elif self._current_char in DIGIT:
                tokens.append(self.make_number())
            elif self._current_char == '=':
                tokens.append(Token(TT_ASIGN))
                self.advance()
            else:
                pos = self._pos.copy()
                char = self._current_char
                self.advance()
                return [], IllegalCharError(pos, self._pos, "'" + char + "'"
)
        return tokens, None
"""""""""""""""""""""""""""
def make_identifier(text):
    num = float(' '.join(map(str, text)).split('=')[-1])
    return num

def run(fn, text):
    lexer = Lexer(fn, text)
    tokens, error = lexer.analyze()
    return tokens, error
```

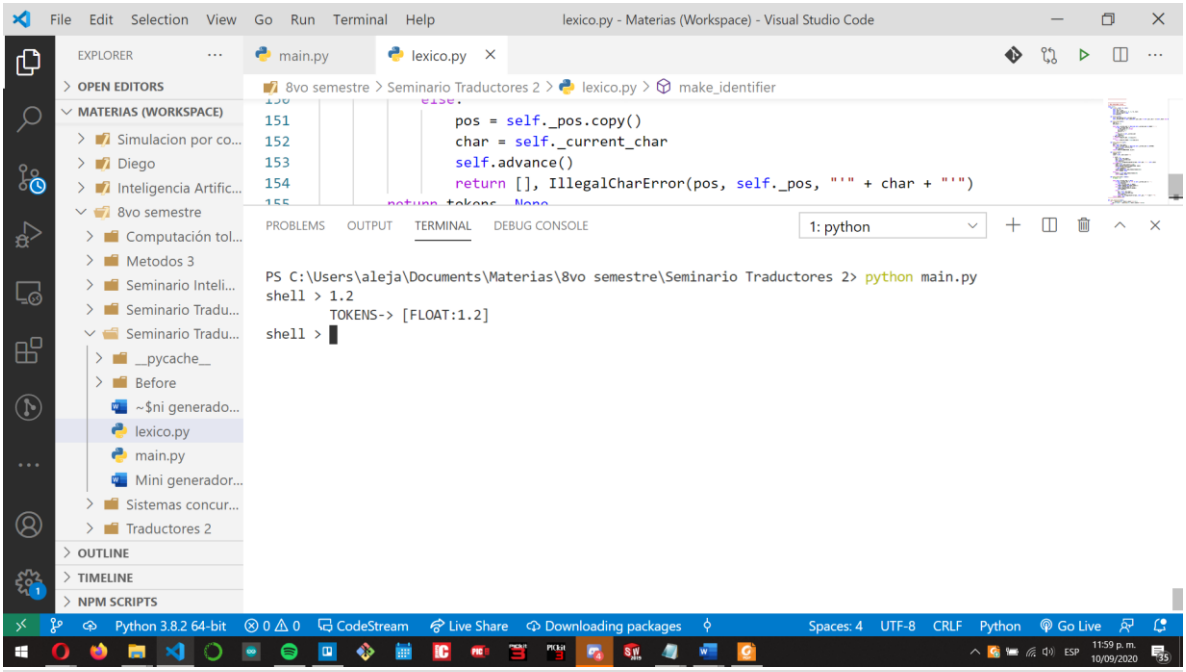
Al finalizar esto, tendremos completo nuestro mini analizador léxico, por lo cual sigue hacer la entrada del texto así como lo que esto conlleva.

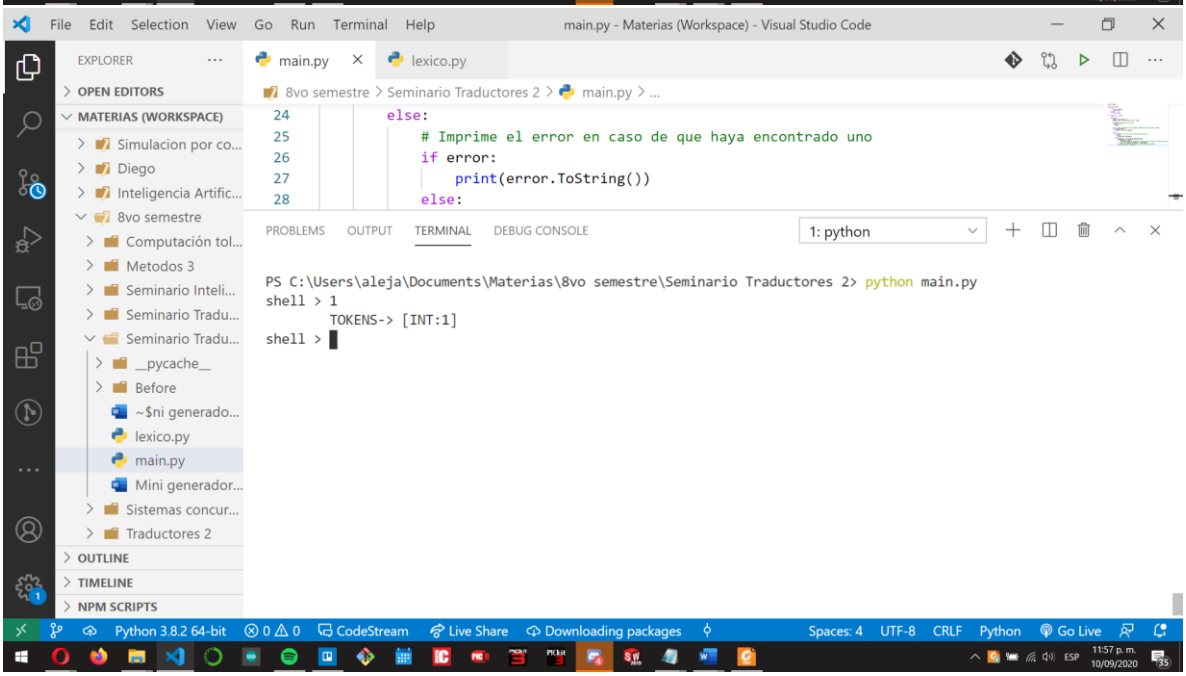
```
import lexico

def isExist(text):
    try:
        globals()[text]
```

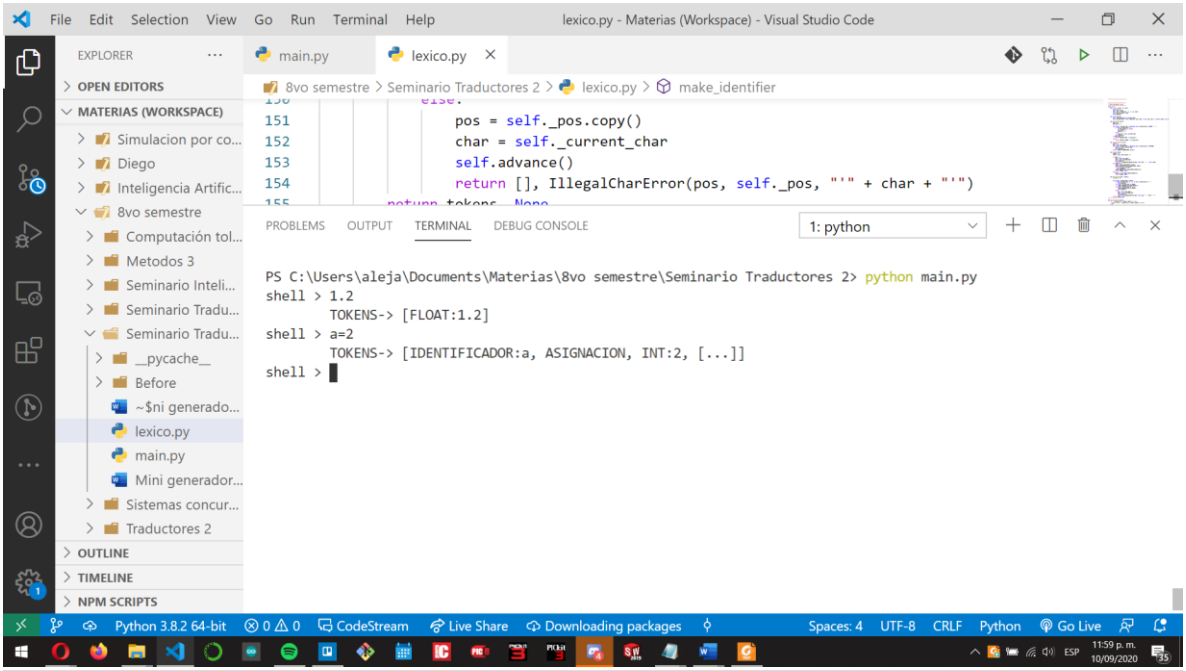
```
        return True
    except:
        return False

if __name__ == '__main__':

    while True:
        text = input('shell > ')
        tokens, error = lexico.run('<stdin>', text)

        # Comando para salir de la consola
        if text == '\c':
            break

        # Si el texto ingresado es una variable previamente guarda se manda
a llamar
        elif isExist(text):
            print("\t", globals()[text])

        else:
            # Imprime el error en caso de que haya encontrado uno
            if error:
                print(error.ToString())
            else:
                expression = lexico.make_identifier(text)
                if expression != None:
                    # De haber hecho la declaracion de una variable se añade
 al entorno junto con su valor
                    globals()[(tokens[0].Value())] = expression
                # Imprime los tokens encontrados con sus respectivos valores
                print("\tTOKENS->", tokens)
```

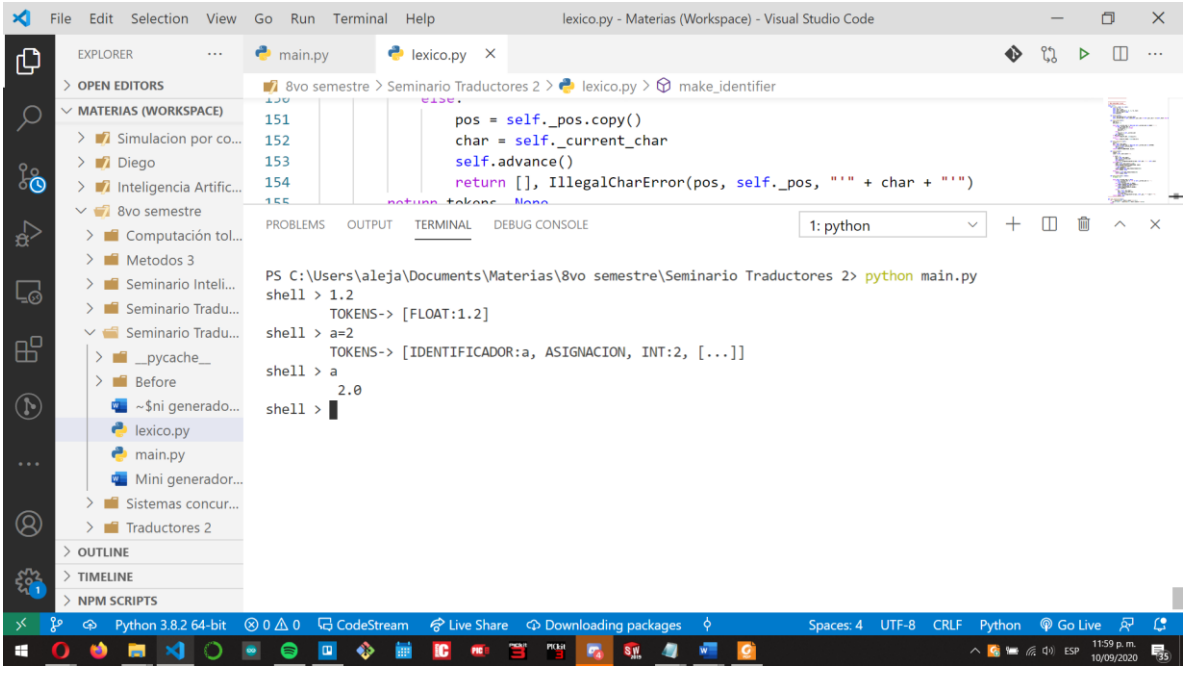
Resultados

Con el analizador léxico programado se procedieron a realizar diversas pruebas para la verificación del funcionamiento del mismo.


File Edit Selection View Go Run Terminal Help
lexico.py - Materias (Workspace) - Visual Studio Code
EXPLORER
main.py
lexico.py
8vo semestre > Seminario Traductores 2 > lexico.py > make_identifier
151 pos = self._pos.copy()
152 char = self._current_char
153 self.advance()
154 return [], IllegalCharError(pos, self._pos, "'" + char + "'")
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
1: python
PS C:\Users\aleja\Documents\Materias\8vo semestre\Seminario Traductores 2> python main.py
shell > 1.2
TOKENS-> [FLOAT:1.2]
shell >
OPEN EDITORS
MATERIAS (WORKSPACE)
Simulacion por co...
Diego
Inteligencia Artific...
8vo semestre
Computación tol...
Metodos 3
Seminario Inteli...
Seminario Tradu...
Seminario Tradu...
__pycache__
Before
~$ni generado...
lexico.py
main.py
Mini generador...
Sistemas concur...
Traductores 2
OUTLINE
TIMELINE
NPM SCRIPTS
Python 3.8.2 64-bit
CodeStream
Live Share
Downloading packages
Spaces: 4 UTF-8 CRLF Python Go Live
11:59 p. m.
10/09/2020


File Edit Selection View Go Run Terminal Help
main.py - Materias (Workspace) - Visual Studio Code
EXPLORER
main.py
lexico.py
8vo semestre > Seminario Traductores 2 > main.py > ...
24 else:
25 # Imprime el error en caso de que haya encontrado uno
26 if error:
27 print(error.ToString())
28 else:
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
1: python
PS C:\Users\aleja\Documents\Materias\8vo semestre\Seminario Traductores 2> python main.py
shell > 1
TOKENS-> [INT:1]
shell >
OPEN EDITORS
MATERIAS (WORKSPACE)
Simulacion por co...
Diego
Inteligencia Artific...
8vo semestre
Computación tol...
Metodos 3
Seminario Inteli...
Seminario Tradu...
Seminario Tradu...
__pycache__
Before
~$ni generado...
lexico.py
main.py
Mini generador...
Sistemas concur...
Traductores 2
OUTLINE
TIMELINE
NPM SCRIPTS
Python 3.8.2 64-bit
CodeStream
Live Share
Downloading packages
Spaces: 4 UTF-8 CRLF Python Go Live
11:57 p. m.
10/09/2020


File Edit Selection View Go Run Terminal Help
lexico.py - Materias (Workspace) - Visual Studio Code
EXPLORER
main.py
lexico.py
8vo semestre > Seminario Traductores 2 > lexico.py > make_identifier
pos = self._pos.copy()
char = self._current_char
self.advance()
return [], IllegalCharError(pos, self._pos, "'" + char + "'")
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
1: python
PS C:\Users\aleja\Documents\Materias\8vo semestre\Seminario Traductores 2> python main.py
shell > 1.2
TOKENS-> [FLOAT:1.2]
shell > a=2
TOKENS-> [IDENTIFICADOR:a, ASIGNACION, INT:2, [...]]
shell >
Python 3.8.2 64-bit
CodeStream
Live Share
Downloading packages
Spaces: 4 UTF-8 CRLF Python Go Live


File Edit Selection View Go Run Terminal Help
lexico.py - Materias (Workspace) - Visual Studio Code
EXPLORER
main.py
lexico.py
8vo semestre > Seminario Traductores 2 > lexico.py > make_identifier
pos = self._pos.copy()
char = self._current_char
self.advance()
return [], IllegalCharError(pos, self._pos, "'" + char + "'")
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
1: python
PS C:\Users\aleja\Documents\Materias\8vo semestre\Seminario Traductores 2> python main.py
shell > 1.2
TOKENS-> [FLOAT:1.2]
shell > a=2
TOKENS-> [IDENTIFICADOR:a, ASIGNACION, INT:2, [...]]
shell > a
2.0
shell >
Python 3.8.2 64-bit
CodeStream
Live Share
Downloading packages
Spaces: 4 UTF-8 CRLF Python Go Live