



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias. Compiladores

Práctica 1

Reporte de Práctica

Autor:

Rosas Marín Jesús Martín - **318291015**

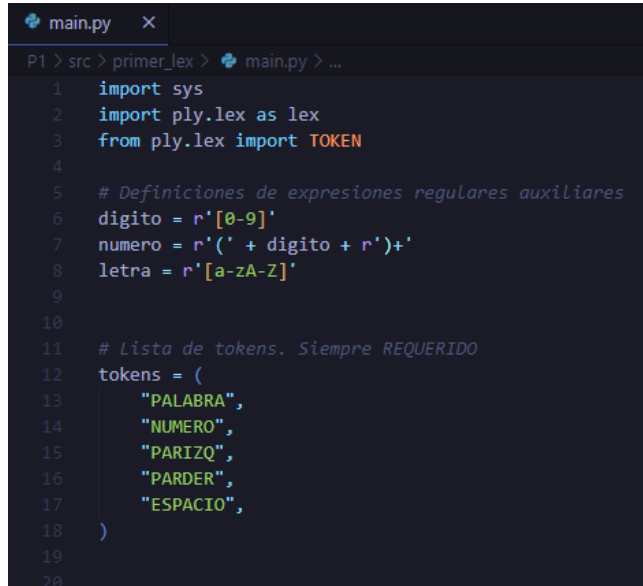
Profesores:

Ariel Adara Mercado Martínez
Carlos Gerardo Acosta Hernández
Erick Bernal Márquez
Yessica Janeth Pablo Martínez
Kevin Isaac Alcantara Estrada

25 de febrero de 2025

Pasos

- Transcribir el código anterior a un archivo con extensión .py

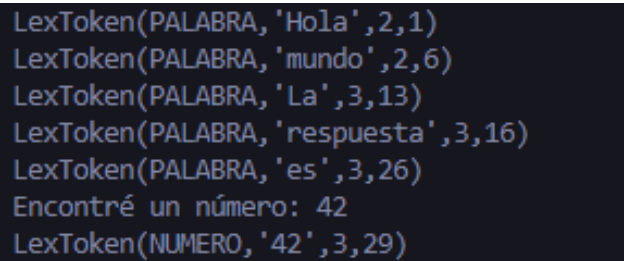


```
main.py x
P1 > src > primer_lex > main.py > ...
1 import sys
2 import ply.lex as lex
3 from ply.lex import TOKEN
4
5 # Definiciones de expresiones regulares auxiliares
6 digito = r'[0-9]'
7 numero = r'(' + digito + r')+'
8 letra = r'[a-zA-Z]'
9
10
11 # Lista de tokens. Siempre REQUERIDO
12 tokens = (
13     "PALABRA",
14     "NUMERO",
15     "PARIZQ",
16     "PARDER",
17     "ESPACIO",
18 )
19
20
```

El código se transcribió y se guardó en la ruta indicada como main.py.

- Ejecutar mediante la instrucción: `python archivo.py`

Veamos la ejecución sin parámetros:



```
LexToken(PALABRA, 'Hola', 2, 1)
LexToken(PALABRA, 'mundo', 2, 6)
LexToken(PALABRA, 'La', 3, 13)
LexToken(PALABRA, 'respuesta', 3, 16)
LexToken(PALABRA, 'es', 3, 26)
Encontré un número: 42
LexToken(NUMERO, '42', 3, 29)
```

- Crear un archivo de texto que será la entrada: `echo "Hay 100 nubes en el cielo hoy" > input.txt`

Se creó el archivo input.txt con el texto requerido y se guardó en la misma ubicación donde está el main.py, es decir en **src/primer_lex/**

- Ejecutar mediante: `python main.py input.txt`

```

LexToken(PALABRA, 'Hay', 1, 0)
Encontré un número: 100
LexToken(NUMERO, '100', 1, 4)
LexToken(PALABRA, 'nubes', 1, 8)
LexToken(PALABRA, 'en', 1, 14)
LexToken(PALABRA, 'el', 1, 17)
LexToken(PALABRA, 'cielo', 1, 20)
LexToken(PALABRA, 'hoy', 1, 26)

```

Ejercicios

- ¿Qué ocurre si agregamos una regla simple como `t_espacio = r' /+ '` y nada más?

Primero veamos que esta regla define un token llamado espacio que reconoce uno o más espacios, pero notemos que si solo agregamos la regla sin haber agregado el token correspondiente a ESPACIO, obtendremos un error.

Ahora, si también definimos un token llamado ESPACIO dentro de tokens:

```

# Lista de tokens. Siempre REQUERIDO
tokens = (
    "PALABRA",
    "NUMERO",
    "PARIZQ",
    "PARDER",
    "ESPACIO",
)

# Definición de reglas en una sola línea sin acción léxica
t_PALABRA = r'(' + letra + r'+)'
t_PARIZQ = r'\('
t_PARDER = r'\)'
t_ESPACIO = r'\ +'

```

En este caso obtendremos el mismo resultado que si no hubiéramos agregado nada al código original ya que tenemos la línea `t_ignore` que ignora los espacios y las tabulaciones.

Ahora, si comentamos esa línea para que pueda captar los espacios, veamos la salida:

```

LexToken(PALABRA, 'Hola', 2, 1)
LexToken(ESPACIO, ' ', 2, 5)
LexToken(PALABRA, 'mundo', 2, 6)
LexToken(ESPACIO, ' ', 2, 11)
LexToken(PALABRA, 'La', 3, 13)
LexToken(ESPACIO, ' ', 3, 15)
LexToken(PALABRA, 'respuesta', 3, 16)
LexToken(ESPACIO, ' ', 3, 25)
LexToken(PALABRA, 'es', 3, 26)
LexToken(ESPACIO, ' ', 3, 28)
Encontré un número: 42
LexToken(NUMERO, '42', 3, 29)

```

Veamos que ahora el lexer reconoce los espacios como tokens de tipo ESPACIO.

- ¿Qué ocurre si quitamos algún elemento de la lista de tokens?

Veamos primero que cada nombre en esta lista debe coincidir con una regla de token definida en el lexer, por ejemplo para el token PALABRA debe haber una regla t_PALABRA.

Ahora, si el token que eliminamos tiene una regla definida, PLY generará un error al construir el lexer. Esto se debe a que PLY espera que todos los tokens mencionados en la lista tengan una regla.

Vamos a eliminar el token PALABRA y dejaremos su regla t_PALABRA y veamos la salida:

```
ERROR: Rule 't_PALABRA' defined for an unspecified token PALABRA
Traceback (most recent call last):
  File "C:\Users\messi\OneDrive\Escritorio\compiladores\RosasMarinJesus_F
    lexer = lex.lex()
  File "C:\Users\messi\OneDrive\Escritorio\Cursos\Pruebas\tareasCrip\Prac
    raise SyntaxError("Can't build lexer")
SyntaxError: Can't build lexer
```

Ahora, vamos a eliminar también su regla t_PALABRA y veamos la salida:

```
Error léxico. Caracter no reconocido: 'H'
Error léxico. Caracter no reconocido: 'o'
Error léxico. Caracter no reconocido: 'l'
Error léxico. Caracter no reconocido: 'a'
Error léxico. Caracter no reconocido: 'm'
Error léxico. Caracter no reconocido: 'u'
Error léxico. Caracter no reconocido: 'n'
Error léxico. Caracter no reconocido: 'd'
Error léxico. Caracter no reconocido: 'o'
Error léxico. Caracter no reconocido: 'L'
Error léxico. Caracter no reconocido: 'a'
Error léxico. Caracter no reconocido: 'r'
Error léxico. Caracter no reconocido: 'e'
Error léxico. Caracter no reconocido: 's'
Error léxico. Caracter no reconocido: 'p'
Error léxico. Caracter no reconocido: 'u'
Error léxico. Caracter no reconocido: 'e'
Error léxico. Caracter no reconocido: 's'
Error léxico. Caracter no reconocido: 't'
Error léxico. Caracter no reconocido: 'a'
Error léxico. Caracter no reconocido: 'e'
Error léxico. Caracter no reconocido: 's'
Encontré un número: 42
LexToken(NUMERO, '42', 3, 29)
```

Notemos que como tenemos definida nuestra función t_error, podemos capturar el carácter no reconocido y de esta manera no podríamos reconocer el carácter, pero también notemos que los demás tokens y reglas siguen funcionando como NUMERO.

Por lo tanto, si solo eliminamos el token, PLY generará un error al construir el lexer.

- ¿Cómo podemos calcular la posición en columna en caso de un error léxico?

Notemos que cada LexToken viene con la posición en columna y en fila del inicio de cada Token, es decir, veamos:

```
LexToken(PALABRA, 'Hay', 1, 0)
Encontré un número: 100
LexToken(NUMERO, '100', 1, 4)
LexToken(PALABRA, 'nubes', 1, 8)
LexToken(PALABRA, 'en', 1, 14)
LexToken(PALABRA, 'el', 1, 17)
LexToken(PALABRA, 'cielo', 1, 20)
LexToken(PALABRA, 'hoy', 1, 26)
```

La palabra Nubes esta en la línea 1 y en la columna 8 comenzando en 0.

Usaremos la propiedad `t.lexpos` que nos indica la posición absoluta del carácter en la entrada, pero para calcular la posición en columna, necesitamos tener en cuenta los saltos de línea por lo que haremos lo siguiente:

Crearemos una función auxiliar para encontrar la columna y la agregaremos a nuestra función `t_error`:

```
# Calcula la posición de la columna en la línea actual
def find_column(input, token):
    line_start = input.rfind('\n', 0, token.lexpos) + 1
    return (token.lexpos - line_start) + 1

# Esta función nos permite manejar el estado de error a nuestra conveniencia
def t_error(t):
    col = find_column(t.lexer.lexdata, t)
    print("Error léxico. Caracter no reconocido: '%s'" % t.value[0] +
          " en la línea " + str(t.lineno) + ", columna " + str(col))
    t.lexer.skip(1)
```

Veamos que de esta manera podemos contar la línea en donde esta el error y ademas encontrar la columna.

Ahora para probar este código, crearemos un nuevo archivo llamado `error.txt` que contendrá la cadena:

`"Error*"`

Veamos que el error esta en la línea 1 ya que solo hay una línea y ademas en la columna 6 empezando a contar en 1, notemos que el carácter con error es el asterisco, veamos la salida de nuestro problema:

```
LexToken(PALABRA, 'Error', 1, 0)
Error léxico. Caracter no reconocido: '*' en la línea 1, columna 6
```

- ¿Qué significa el valor que se aloja en `t.value`?

El valor de `t.value` es el lexema o la cadena de caracteres que coinciden con la expresión regular definida para el token. Significa que `t.value` contiene el texto específico de la entrada que ha sido identificado y capturado por la regla.

Por ejemplo, en la regla `t_NUMERO` coincide con un lexema que representa un número. Si la entrada es 42, entonces `t.value` será "42", y se imprimirá "Encontré un número: 42".

- ¿Qué pasa al ejecutar el programa e introducir cadenas de caracteres y de dígitos sin espacios en el archivo de entrada?

Se creo un archivo llamado `sin_espacios.txt` donde se encuentra la cadena:
"H0la123adios"

Ahora veamos la salida:

```
LexToken(PALABRA, 'H', 1, 0)
Encontré un número: 0
LexToken(NUMERO, '0', 1, 1)
LexToken(PALABRA, 'la', 1, 2)
Encontré un número: 123
LexToken(NUMERO, '123', 1, 4)
LexToken(PALABRA, 'adios', 1, 7)
```

Veamos que tanto en la búsqueda de una palabra y un número, buscara el mayor número de elementos consecutivos, de esta manera 0 y 123 los ve como un número y H, La y adios los ve como una palabra.

- ¿Qué ocurre si introducimos caracteres como "*" en el archivo de entrada?

Se creo un archivo llamado `asterisco.txt` donde se encuentra la cadena:
"Hola*"

Ahora veamos la salida:

```
LexToken(PALABRA, 'Hola', 1, 0)
Error léxico. Caracter no reconocido: '*'
```

Veamos que debido a que no tenemos ninguna regla sobre este tipo de caracteres, entonces no lo va a reconocer y entrara en nuestra función `t_error`.

- Modificar al código anterior en un archivo nuevo, de tal manera que ejecute una acción léxica al detectar lo siguiente:
 - La expresión regular para los hexadecimales en lenguaje C.
 - 5 palabras reservadas del lenguaje Python.
 - Los identificadores válidos del lenguaje Java, con longitud máxima de 32 caracteres (Sugerencia: use el operador `m,n`).
 - Los espacios en blanco.

Se creo un script nuevo llamado **ejercicio.7** dentro de una carpeta llamada `Ejercicio7`.

Ahora veamos nuestros nuevos TOKENS:

```
# Lista de tokens. Siempre REQUERIDO
tokens = (
    "NUMERO",
    "HEXADECIMAL",
    "PALABRA_PYTHON",
    "IDENTIFICADOR_JAVA",
    "ESPACIO",
)
```

Donde:

- **HEXADECIMAL**: Admite los hexadecimales en lenguaje C.
- **PALABRA_PYTHON**: Admite 5 palabras reservadas del lenguaje Python.
- **IDENTIFICADOR_JAVA**: Admite los identificadores válidos del lenguaje Java, con longitud máxima de 32 caracteres.
- **ESPACIO**: Admite los espacios en blanco.

Ahora veamos nuestras reglas:

```
# Definición de reglas en una sola línea sin acción léxica

# captar palabras reservadas de Python
t_PALABRA_PYTHON = r'(import|return|if|else|for)'
t_IDENTIFICADOR_JAVA = r'(!return|if|for|else|import)[a-zA-Z_$][a-zA-Z0-9_$]{0,31}'
t_ESPACIO = r'[\s \t]+'

# Definición de reglas con acción léxica
@TOKEN(r'0[xX][0-9a-fA-F]+')
def t_HEXADECIMAL(t):
    print(f"Encontré un hexadecimal: {t.value} y su valor es {int(t.value, 16)}")
    return t
```

Veamos que tenemos tres reglas de una sola línea:

- **t_PALABRA_PYTHON**: Que va a poder detectar las palabras **import**, **return**, **if**, **else** y **for**
- **t_IDENTIFICADOR_JAVA**: Que podrá detectar todos los identificadores de tamaño máximo 32 en el lenguaje Java a excepción de las 5 palabras reservadas en Python anteriores ya que si observamos detenidamente, en Java también son reservadas.
- **t_ESPACIO**: Que podrá detectar los espacios.

Por último tenemos una regla con acción léxica que es la función **t_HEXADECIMAL** esta función detectará todos los números en hexadecimal en el lenguaje C, además nos imprimirá un mensaje cuando lo llegue a encontrar junto con su

valor entero.

También se agregó un texto predeterminado a la variable **data**, esta se ejecutará cuando el programa se ejecute sin parámetros:

```
# Código fuente
data = """
import tarea1
x
if x esIgual a 5
    return 5
else
    return 6
"""
```

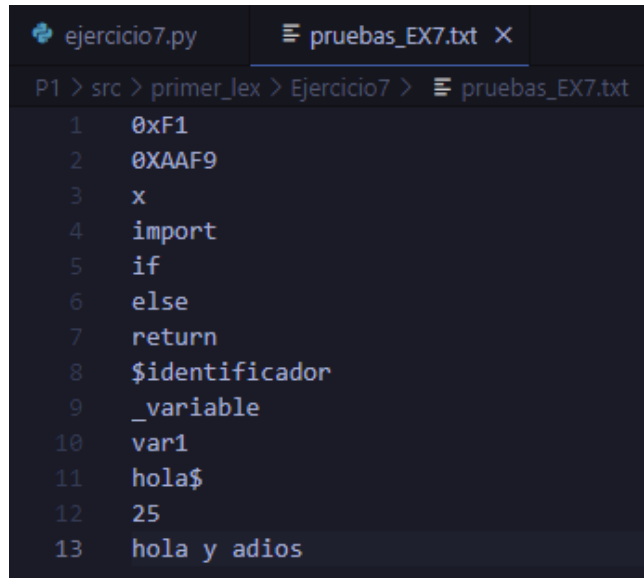
Veamos la salida que recibimos si corremos el programa sin argumentos:

```
LexToken(PALABRA_PYTHON,'import',2,1)
LexToken(ESPACIO,' ',2,7)
LexToken(IDENTIFICADOR_JAVA,'tarea1',2,8)
LexToken(IDENTIFICADOR_JAVA,'x',3,15)
LexToken(PALABRA_PYTHON,'if',4,17)
LexToken(ESPACIO,' ',4,19)
LexToken(IDENTIFICADOR_JAVA,'x',4,20)
LexToken(ESPACIO,' ',4,21)
LexToken(IDENTIFICADOR_JAVA,'esIgual',4,22)
LexToken(ESPACIO,' ',4,29)
LexToken(IDENTIFICADOR_JAVA,'a',4,30)
LexToken(ESPACIO,' ',4,31)
Encontré un número: 5
Encontré un número: 5
LexToken(NUMERO,'5',4,32)
LexToken(ESPACIO,' ',5,34)
LexToken(PALABRA_PYTHON,'return',5,38)
LexToken(ESPACIO,' ',5,44)
Encontré un número: 5
LexToken(NUMERO,'5',5,45)
LexToken(PALABRA_PYTHON,'else',6,47)
LexToken(ESPACIO,' ',7,52)
LexToken(PALABRA_PYTHON,'return',7,56)
LexToken(ESPACIO,' ',7,62)
Encontré un número: 6
LexToken(NUMERO,'6',7,63)
```

Veamos que identifica cada token pedido en el ejercicio, ahora si se quiere probar este código se puede ejecutar con:

python ejercicio7.py

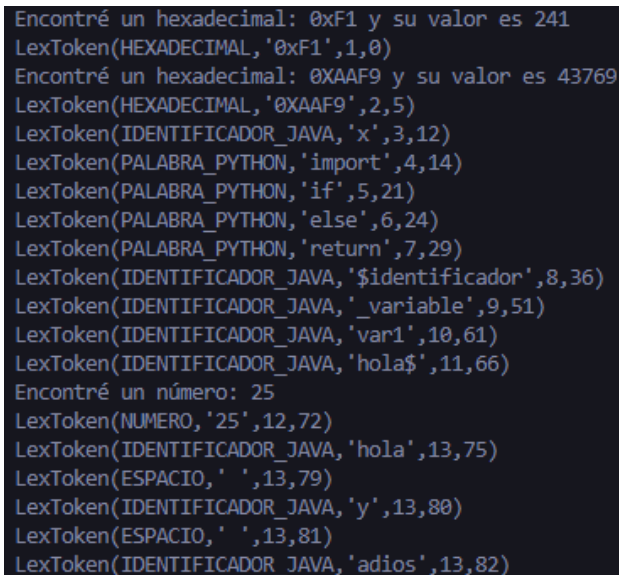
Por ultimo, se creo un archivo llamado **pruebas_EX7.txt** ubicado en la misma carpeta Ejercicio7, donde se incluyen algunas pruebas para poder probar el codigo de manera que se le pasen argumentos, veamoslo:



```
ejercicio7.py pruebas_EX7.txt X
P1 > src > primer_lex > Ejercicio7 > pruebas_EX7.txt
1 0xF1
2 0XAAF9
3 x
4 import
5 if
6 else
7 return
8 $identificador
9 _variable
10 var1
11 hola$
12 25
13 hola y adios
```

Ahora ejecutemoslo con:

python ejercicio7.py pruebas_EX7.txt



```
Encontré un hexadecimal: 0xF1 y su valor es 241
LexToken(HEXADECIMAL,'0xF1',1,0)
Encontré un hexadecimal: 0XAAF9 y su valor es 43769
LexToken(HEXADECIMAL,'0XAAF9',2,5)
LexToken(IDENTIFICADOR_JAVA,'x',3,12)
LexToken(PALABRA_PYTHON,'import',4,14)
LexToken(PALABRA_PYTHON,'if',5,21)
LexToken(PALABRA_PYTHON,'else',6,24)
LexToken(PALABRA_PYTHON,'return',7,29)
LexToken(IDENTIFICADOR_JAVA,'$identificador',8,36)
LexToken(IDENTIFICADOR_JAVA,'_variable',9,51)
LexToken(IDENTIFICADOR_JAVA,'var1',10,61)
LexToken(IDENTIFICADOR_JAVA,'hola$',11,66)
Encontré un número: 25
LexToken(NUMERO,'25',12,72)
LexToken(IDENTIFICADOR_JAVA,'hola',13,75)
LexToken(ESPACIO,' ',13,79)
LexToken(IDENTIFICADOR_JAVA,'y',13,80)
LexToken(ESPACIO,' ',13,81)
LexToken(IDENTIFICADOR_JAVA,'adios',13,82)
```

Veamos que se identifican cada token solicitado. En este archivo de prueba se prueban dos nuemros hexadecimales dife-
rentes, distintos tipos de identificador en java, las palabras reservadas en python, algunos numeros y algunos espacios.