Ayudantía Lenguajes de Programación

PLY: Python Lex-Yacc

Rodolfo Castillo Mateluna

U. Técnica Federico Santa María
Departamento de Informática

Segundo semestre académico, 2017

Disclaimer

Gran parte del contenido de esta presentación está basado en la documentación oficial de PLY.

Esta puede ser encontrada en www.dabeaz.com/ply/ply.html

Requerimientos

- Python (> 2.6)
- Conocimiento general de **gramáticas formales**
- Expresiones regulares

Lex

Definición

Un *lexer* o "analizador léxico" (crudamente traducido al español) es un programa que ejecuta el proceso de convertir una secuencia de caracteres en una secuencia de *strings* con un significado asociado.

- `Lex` es un programa que genera *lexers*. Es originalmente una librería con *bindings* para el lenguaje de programación `C`.
- Comunmente es acompañado de `Yacc`

Yacc (*Yet Another Compiler-Compiler*)

Definición

Parsing o "análisis sintáctico" es el proceso de analizar un *string* que sigue las reglas de una **gramática formal**

Yacc (*Yet Another Compiler-Compiler*)

Definición

La **derivación** (*derivation*) de un *string* es la secuencia de reglas de una gramática formal aplicadas para transformar el símbolo inicial en el *string*

Es muy importante tener en consideración si el proceso de análisis sintáctico se realiza como una *leftmost* o *rightmost derivation*

Yacc (*Yet Another Compiler-Compiler*)

Definición

Un *LALR parser* (*Look ahead LR parser*) es un tipo de analizador sintáctico optimizado en base a un *LR parser* (*Left to right, rightmost derivation*)

Yacc (*Yet Another Compiler-Compiler*)

Yacc es un *LALR parser generator*. En otras palabras, es un generador de *parsers*. Un *parser* es un programa que realiza análisis sintáctico. Entonces, es correcto decir que Yacc, es un programa que genera un programa para analizar *strings* potencialmente pertenecientes a un lenguaje generado por una gramática formal.

Yacc (*Yet Another Compiler-Compiler*)

Yacc es un *LALR parser generator*. En otras palabras, es un generador de *parsers*. Un *parser* es un programa que realiza análisis sintáctico. Entonces, es correcto decir que Yacc, es un programa que genera un programa para analizar *strings* potencialmente pertenecientes a un lenguaje generado por una gramática formal.

¡Ahora *Compiler Compiler* tiene sentido!

A lo que nos convoca

- PLY es una librería que implementa Lex+Yacc en Python
- Inicialmente fue desarrollada para un curso de compiladores
- PLY utiliza **reflexión** para generar *lexers* y *parsers*

Instalación

Existen varias maneras de instalar paquetes en `Python`. La más utilizada (a veces repudiada por *sysadmins* puristas) es mediante el gestor de paquetes `pip`. Para usarlo de manera correcta se recomienda el uso de `virtualenv`. Para más información sobre este tópico, investigue en internet o siga los pasos en el vídeo de esta ayudantía.

lex.py

Conociendo la definición de un *lexer* y la funcionalidad del programa *Lex*, a continuación se mostrarán ejemplos de uso de su implementación en *Python*

Achtung!

Se utilizarán los mismos ejemplos que se pueden encontrar en la documentación

lex.py

Suponga que está haciendo un lenguaje de programación y un usuario ingresa el siguiente string:

```
x = 3 + 42 * (s - t)
```

Un **tokenizer** o *lexer* separará el string en *tokens* uno por uno:

```
'x','=','3','+','42','*','(','s','-','t',')'
```

Más específicamente, el *string* será separado en pares de la forma *(tipo, valor)*:

```
('ID','x'), ('EQUALS','='), ('NUMBER','3'),
('PLUS','+'), ('NUMBER','42'),('TIMES','*'),
('LPAREN','('), ('ID','s'), ('MINUS','-'),
('ID','t'), ('RPAREN',')'
```

lex.py

La identificación del tipo de un *token* es típicamente definida por una serie de expresiones regulares. A continuación se mostrará como puede ser logrado con `lex.py` en `calclex.py`

Entendiendo `calclex.py`

Un *lexer* debe proveer una lista de *tokens* que definen todos los posibles tipos que deben ser reconocidos. El resultado del proceso de *tokenization* es utilizado por `yacc.py`

Ejemplo

```
tokens = (
    'NUMBER',
    'PLUS',
    'MINUS',
    'TIMES',
    'DIVIDE',
    'LPAREN',
    'RPAREN',
)
```

Entendiendo `calclex.py`

Cada *token* es definido mediante una expresión regular compatible con el módulo `re` de *Python*. Cada una de estas reglas están definidas en declaraciones con el prefijo especial `t_`

Entendiendo `calclex.py`

Si la especificación de un *token* es simple, es tan sencillo como declarar una variable de la forma `t_<TOKEN> = <RE>`

Ejemplo

```
t_PLUS = r'\+'
```

En el ejemplo estamos diciendo que el *token* 'PLUS' está definido por la expresión regular '\+'

Entendiendo `calclex.py`

Si la especificación de un *token* es compleja, se puede declarar una función que trabaje con el *token* extraído desde el *input*. La función deberá llamarse `t_<TOKEN>`, recibirá un argumento (el *token* capturado) y retornará el *token* con las modificaciones pertinentes. Es importante destacar que la expresión regular que define al *token* es definida en el *docstring* de la función

Ejemplo

```
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)
    return t
```

Importante!

El *token* debe ser retornado o será descartado

Lea la documentación!

- Internamente `lex.py` usa la *flag* `re.VERBOSE` en el uso el uso del módulo `re`. Esto significa que se aceptan comentarios con el símbolo # y los espacios se descartan, por lo que en caso que su expresión regular necesite indicar explícitamente uno de estos caracteres, debe indicarlos como `[#]` y `\s` respectivamente
- Revise en la documentación la manera apropiada de definir palabras reservadas

Esto no es todo!

Hasta ahora hemos visto como hacer el proceso de análisis léxico con `PLY`, pero eso es sólo la mitad del trabajo. Ahora hace falta pasar por el análisis sintáctico. De esto se encargará `yacc.py`