Grammar Parsing in Python

Neil Muller

9 Feb 2008 (CTPUG 8)

Why?

(if you have to ask, you haven't been coding long enough)

- Often required to be able to deal with mini-languages
  - Config files
  - structured input data
  - and so forth
- Need to enforce syntax contraints
- Need to obtain a useful representation of data

Parser and Lexer Modules

- PLY: http://www.dabeaz.com/ply/
- pyparsing: http://pyparsing.wikispaces.com/
- antlr: http://www.anltr.org/
- SimpleParse: http://simpleparse.sourceforge.net/
- Martel: http://www.dalkescientific.com/Martel/
- ZestyParser: http://zestyparser.admatlas.org/
- Rparse: http://della1rv.googlepages.com/theparserparsergenerator
- Parsing: http://www.canonware.cm/Parsing
- Numerous others
- Not to mention special purpose parsers - ConfigParser, parser, shlex and so on.

Parser and Lexer Modules

- *PLY: http://www.dabeaz.com/ply/*
- *pyparsing: http://pyparsing.wikispaces.com/*

Hello World! Example

Problem:

Parse 'Hello World!'

Hello World! Example

Problem:

Parse 'Hello World!'

Grammar specification:

Parser:

Greeting: Word Word SentenceEnd

Hello World! Example

Problem:

Parse 'Hello World!'

Grammar specification:

Parser:

Greeting: Word Word SentenceEnd

Lexer:

Word = [A-Za-z]*

SentenceEnd = !

Hello World! Example

Problem:

Parse 'Hello World!'

Grammar specification:

Parser:

Greeting: [Hello|Hey|Howdy] Word SentenceEnd

Lexer:

Word = [A-Za-z]*

SentenceEnd = !

Further Examples

- Commas
  - Is the behaviour of ' “A, B”, C, “D, E” '.split(',') desired?
- Simple Calculator
  - Handle expressions: 1+2 *3
  - Usual precedence
  - Handle ()'s

Comments

- PyParsing:
  - Pythonic syntax: relies heavily on operator overloading

Comments

- PyParsing:
  - Pythonic syntax: relies heavily on operator overloading
  - Lexer and Parser defined simulatenously

Comments

- PyParsing:
  - Pythonic syntax: relies heavily on operator overloading
  - Lexer and Parser defined simulatenously
  - Useful high-level constructs (Keyword, etc)

Comments

- PyParsing:
  - Pythonic syntax: relies heavily on operator overloading
  - Lexer and Parser defined simulatenously
  - Useful high-level constructs (Keyword, etc)
  - tends to be slow

Comments

- PyParsing:
  - Pythonic syntax: relies heavily on operator overloading
  - Lexer and Parser defined simulatenously
  - Useful high-level constructs (Keyword, etc)
  - tends to be slow
- PLY:
  - magic function/method names

Comments

- PyParsing:
  - Pythonic syntax: relies heavily on operator overloading
  - Lexer and Parser defined simulatenously
  - Useful high-level constructs (Keyword, etc)
  - tends to be slow
- PLY:
  - magic function/method names
  - File-orientated: class-based approaches can be fudged, but it is a fudge

Comments

- PyParsing:
  - Pythonic syntax: relies heavily on operator overloading
  - Lexer and Parser defined simulatenously
  - Useful high-level constructs (Keyword, etc)
  - tends to be slow
- PLY:
  - magic function/method names
  - File-orientated: class-based approaches can be fudged, but it is a fudge
  - docstrings are significant

Comments

- PyParsing:
  - Pythonic syntax: relies heavily on operator overloading
  - Lexer and Parser defined simulatenously
  - Useful high-level constructs (Keyword, etc)
  - tends to be slow
- PLY:
  - magic function/method names
  - File-orientated: class-based approaches can be fudged, but it is a fudge
  - docstrings are significant *(but at least that means docstrings are there)*
  - BNF syntax

Comments

- PyParsing:
  - Pythonic syntax: relies heavily on operator overloading
  - Lexer and Parser defined simulatenously
  - Useful high-level constructs (Keyword, etc)
  - tends to be slow
- PLY:
  - magic function/method names
  - File-orientated: class-based approaches can be fudged, but it is a fudge
  - docstrings are significant *(but at least that means docstrings are there)*
  - BNF syntax
  - Seperate lexer and parser