textparser Documentation

Release 0.21.1

Erik Moqvist

Feb 28, 2019

Contents

CHAPTER 1

About

A text parser written in the Python language.

The project has one goal, speed! See the benchmark below more details.

Project homepage: https://github.com/eerimoq/textparser

Documentation: http://textparser.readthedocs.org/en/latest

CHAPTER 2

Credits

- Thanks PyParsing for a user friendly interface. Many of `textparser`'s class names are taken from this project.

CHAPTER 3

Installation

```
pip install textparser
```

CHAPTER 4

Example usage

The Hello World example parses the string `Hello, World!` and outputs its parse tree `['Hello', ',', 'World', '!']`.

The script:

```
import textparser
from textparser import Sequence


class Parser(textparser.Parser):

    def token_specs(self):
        return [
            ('SKIP',           r'[ \r\n\t]+'),
            ('WORD',           r'\w+'),
            ('EMARK',     '!', r'!'),
            ('COMMA',     ',', r','),
            ('MISMATCH',       r'.')
        ]

    def grammar(self):
        return Sequence('WORD', ',', 'WORD', '!')


tree = Parser().parse('Hello, World!')

print('Tree:', tree)
```

Script execution:

```
$ env PYTHONPATH=. python3 examples/hello_world.py
Tree: ['Hello', ',', 'World', '!']
```

CHAPTER 5

Benchmark

A benchmark comparing the speed of 10 JSON parsers, parsing a 276 kb file.

```
$ env PYTHONPATH=. python3 examples/benchmarks/json/speed.py

Parsed 'examples/benchmarks/json/data.json' 1 time(s) in:

PACKAGE         SECONDS   RATIO  VERSION
textparser         0.09    100%  0.19.0
parsimonious       0.17    183%  unknown
lark (LALR)        0.29    306%  0.6.6
funcparserlib      0.33    346%  unknown
textx              0.53    557%  1.8.0
pyparsing          0.67    710%  2.3.1
pyleri             0.78    825%  1.2.2
parsy              0.91    969%  1.2.0
lark (Earley)      2.11   2240%  0.6.6
parsita            2.26   2393%  unknown
```

NOTE 1: The parsers are not necessarily optimized for speed. Optimizing them will likely affect the measurements.

NOTE 2: The structure of the resulting parse trees varies and additional processing may be required to make them fit the user application.

NOTE 3: Only JSON parsers are compared. Parsing other languages may give vastly different results.

CHAPTER 6

Contributing

1. Fork the repository.
2. Install prerequisites.

```
pip install -r requirements.txt
```

3. Implement the new feature or bug fix.
4. Implement test case(s) to ensure that future changes do not break legacy.
5. Run the tests.

```
make test
```

6. Create a pull request.

CHAPTER 7

The parser class

class textparser.**Parser**

The abstract base class of all text parsers.

```
>>> from textparser import Parser, Sequence
>>> class MyParser(Parser):
...     def token_specs(self):
...         return [
...             ('SKIP',           r'[ \r\n\t]+'),
...             ('WORD',           r'\w+'),
...             ('EMARK',     '!', r'!'),
...             ('COMMA',     ',', r','),
...             ('MISMATCH',       r'.')
...         ]
...     def grammar(self):
...         return Sequence('WORD', ',', 'WORD', '!')
```

keywords()

A set of keywords in the text.

```
def keywords(self):
    return set(['if', 'else'])
```

token_specs()

The token specifications with token name, regular expression, and optionally a user friendly name.

Two token specification forms are available; `(kind, re)` or `(kind, name, re)`. If the second form is used, the grammar should use *name* instead of *kind*.

See *Parser* for an example usage.

tokenize(*text*)

Tokenize given string *text*, and return a list of tokens. Raises *TokenizeError* on failure.

This method should only be called by *parse()*, but may very well be overridden if the default implementation does not match the parser needs.

grammar()
The text grammar is used to create a parse tree out of a list of tokens.

See *Parser* for an example usage.

parse(*text*, *token_tree=False*, *match_sof=False*)
Parse given string *text* and return the parse tree. Raises *ParseError* on failure.

Returns a parse tree of tokens if *token_tree* is `True`.

```
>>> MyParser().parse('Hello, World!')
['Hello', ',', 'World', '!']
>>> tree = MyParser().parse('Hello, World!', token_tree=True)
>>> from pprint import pprint
>>> pprint(tree)
[Token(kind='WORD', value='Hello', offset=0),
 Token(kind=',', value=',', offset=5),
 Token(kind='WORD', value='World', offset=7),
 Token(kind='!', value='!', offset=12)]
```

CHAPTER 8

Building the grammar

The grammar built by combining the classes below and strings.

Here is a fictitious example grammar:

```
grammar = Sequence(
    'BEGIN',
    Optional(choice('IF', Sequence(ZeroOrMore('NUMBER')))),
    OneOrMore(Sequence('WORD', Not('NUMBER'))),
    Any(),
    DelimitedList('WORD', delim=':'),
    'END')
```

class textparser.**Sequence**(**patterns*)
Matches a sequence of patterns. Becomes a list in the parse tree.

class textparser.**Choice**(**patterns*)
Matches any of given ordered patterns *patterns*. The first pattern in the list has highest priority, and the last lowest.

class textparser.**ChoiceDict**(**patterns*)
Matches any of given patterns. The first token kind of all patterns must be unique, otherwise and *Error* exception is raised.

This class is faster than *Choice*, and should be used if the grammar allows it.

textparser.**choice**(**patterns*)
Returns an instance of the fastest choice class for given patterns *patterns*. It is recommended to use this function instead of instantiate *Choice* or *ChoiceDict* directly.

class textparser.**ZeroOrMore**(*pattern*)
Matches *pattern* zero or more times.

See *Repeated* for more details.

class textparser.**ZeroOrMoreDict**(*pattern*, *key=None*)
Matches *pattern* zero or more times.

See *RepeatedDict* for more details.

class textparser.**OneOrMore**(*pattern*)
Matches *pattern* one or more times.

See *Repeated* for more details.

class textparser.**OneOrMoreDict**(*pattern*, *key=None*)
Matches *pattern* one or more times.

See *RepeatedDict* for more details.

class textparser.**DelimitedList**(*pattern*, *delim=', '*)
Matches a delimented list of *pattern* separated by *delim*. *pattern* must be matched at least once. Any match becomes a list in the parse tree, excluding the delimitors.

class textparser.**Optional**(*pattern*)
Matches *pattern* zero or one times. Becomes a list in the parse tree, empty on mismatch.

class textparser.**Any**
Matches any token.

class textparser.**AnyUntil**(*pattern*)
Matches any token until given pattern is found. Becomes a list in the parse tree, not including the given pattern match.

class textparser.**And**(*pattern*)
Matches *pattern*, without consuming any tokens. Any match becomes an empty list in the parse tree.

class textparser.**Not**(*pattern*)
Matches if *pattern* does not match. Any match becomes an empty list in the parse tree.

Just like *And*, no tokens are consumed.

class textparser.**NoMatch**
Never matches anything.

class textparser.**Tag**(*name*, *pattern*)
Tags any matched *pattern* with name *name*. Becomes a two-tuple of *name* and match in the parse tree.

class textparser.**Forward**
Forward declaration of a pattern.

```
>>> foo = Forward()
>>> foo <<= Sequence('NUMBER')
```

class textparser.**Repeated**(*pattern*, *minimum=0*)
Matches *pattern* at least *minimum* times. Any match becomes a list in the parse tree.

class textparser.**RepeatedDict**(*pattern*, *minimum=0*, *key=None*)
Same as *Repeated*, but becomes a dictionary instead of a list in the parse tree.

key is a function taking the match as input and returning the dictionary key. By default the first element in the match is used as key.

class textparser.**Pattern**
Base class of all patterns.

match(*tokens*)
Returns *MISMATCH* on mismatch, and anything else on match.

textparser.**MISMATCH = <textparser._Mismatch object>**
Returned by *match()* on mismatch.

CHAPTER 9

Exceptions

class textparser.**Error**
General textparser exception.

class textparser.**ParseError**(*text*, *offset*)
This exception is raised when the parser fails to parse the text.

text
The input text to the parser.

offset
Offset into the text where the parser failed.

line
Line where the parser failed.

column
Column where the parser failed.

class textparser.**TokenizeError**(*text*, *offset*)
This exception is raised when the text cannot be converted into tokens.

text
The input text to the tokenizer.

offset
Offset into the text where the tokenizer failed.

class textparser.**GrammarError**(*offset*)
This exception is raised when the tokens cannot be converted into a parse tree.

offset
Offset into the text where the parser failed.

CHAPTER 10

Utility functions

textparser.**markup_line**(*text*, *offset*, *marker*='>>!<<')
Insert *marker* at *offset* into *text*, and return the marked line.

```
>>> markup_line('0\n1234\n56', 3)
1>>!<<234
```

textparser.**tokenize_init**(*spec*)
Initialize a tokenizer. Should only be called by the *tokenize()* method in the parser.

Index