parsec documentation

Release 3.15

parsec

Nov 03, 2022

Contents

CHAPTER 1

Introduction

What's the parsec.py ?

A universal Python parser combinator library inspired by Parsec library of Haskell.

Parser combinator is a technique to implement a parser. A parser combinator is a function (higher-order function) that accepts several parsers as arguments and return a new parser as result. Parser combinators enable a recursive descent parsing strategy, this parsing technique facilitates modular piecewise construction and testing. Parser combinators can be used to combine basic parsers to construct parsers for more complex rules, and parser built using combinators are straightforward to construct, readable, modular, well-structured and easily maintainable.

The parsec package is a famous monadic parser combinator library in Haskell. Now, the parsec is a parser combinator library implemented in Python, providing Python developers an easy and elegant approach to build a complex but efficient parser.

Let's go ahead to see the MAGIC of parser combinators!

CHAPTER 2

Examples

The design of parsec's API was inspired by Parsec library of Haskell.

2.1 Simple parser primitives

For more details about this library, see it's documentation at

2.1.1 Documentation for the Code

A universal Python parser combinator library inspired by Parsec library of Haskell.

exception parsec.**ParseError**(*expected*, *text*, *index*)
Type for parse error.

loc()
Locate the error position in the source code text.

static loc_info(*text*, *index*)
Location of *index* in source code *text*.

class parsec.**Parser**(*fn*)
A Parser is an object that wraps a function to do the parsing work. Arguments of the function should be a string to be parsed and the index on which to begin parsing.

The function should return either Value.success(next_index, value) if parsing successfully, or Value.failure(index, expected) on the failure.

bind(*fn*)
This is the monadic binding operation. Returns a parser which, if parser is successful, passes the result to fn, and continues with the parser returned from fn.

choice(*other*)
(|) This combinator implements choice. The parser p | q first applies p.

- If it succeeds, the value of p is returned.

- If p fails **without consuming any input**, parser q is tried.

NOTICE: without backtrack.

compose(*other*)
(>>) Sequentially compose two actions, discarding any value produced by the first.

desc(*description*)
Describe a parser, when it failed, print out the description text.

ends_with(*other*)
(<) Ends with a specified parser, and at the end parser hasn't consumed any input.

excepts(*other*)
Fail though matched when the consecutive parser *other* success for the rest text.

joint(**parsers*)
(+) Joint two or more parsers into one. Return the aggregate of two results from this two parser.

mark()
Mark the line and column information of the result of this parser.

parse(*text*)
Parses a given string *text*.

parse_partial(*text*)
Parse the longest possible prefix of a given string.

Return a tuple of the result value and the rest of the string.

If failed, raise a ParseError.

parse_strict(*text*)
Parse the longest possible prefix of the entire given string.

If the parser worked successfully and NONE text was rested, return the result value, else raise a ParseError.

The difference between *parse* and *parse_strict* is that whether entire given text must be used.

parsecapp(*other*)
Returns a parser that applies the produced value of this parser to the produced value of *other*.

parsecmap(*fn*)
Returns a parser that transforms the produced value of parser with *fn*.

result(*res*)
Return a value according to the parameter *res* when parse successfully.

skip(*other*)
(<<) Ends with a specified parser, and at the end parser consumed the end flag.

try_choice(*other*)
(^) Choice with backtrack. This combinator is used whenever arbitrary look ahead is needed. The parser p || q first applies p, if it success, the value of p is returned. If p fails, it pretends that it hasn't consumed any input, and then parser q is tried.

class parsec.**Value**
Represent the result of the Parser.

aggregate(*other=None*)
collect the furthest failure from self and other.

static combinate(*values*)
Aggregate multiple values into tuple

static failure(*index*, *expected*)
Create failure value.

static success(*index*, *actual*)
Create success value.

parsec.**any**()
Parses a arbitrary character.

parsec.**bind**(*p*, *fn*)
Bind two parsers, implements the operator of *(>>=)*.

parsec.**choice**(*pa*, *pb*)
Choice one from two parsers, implements the operator of *(|)*.

parsec.**compose**(*pa*, *pb*)
Compose two parsers, implements the operator of *(>>)*, or *(>)*.

parsec.**count**(*p*, *n*)
count p n parses n occurrences of p. If n is smaller or equal to zero, the parser equals to return []. Returns a list of n values returned by p.

parsec.**desc**(*p*, *description*)
Describe a parser, when it failed, print out the description text.

parsec.**digit**()
Parse a digit.

parsec.**endBy**(*p*, *sep*)
endBy(p, sep) parses zero or more occurrences of *p*, separated and ended by *sep*. Returns a list of values returned by *p*.

parsec.**endBy1**(*p*, *sep*)
endBy1(p, sep) parses one or more occurrences of 'p, separated and ended by *sep*. Returns a list of values returned by *p*.

parsec.**ends_with**(*pa*, *pb*)
Ends with a specified parser, and at the end parser hasn't consumed any input. Implements the operator of *(<)*.

parsec.**eof**()
Parses EOF flag of a string.

parsec.**excepts**(*pa*, *pb*)
Fail *pa* though matched when the consecutive parser *pb* success for the rest text.

parsec.**exclude**(*p: parsec.Parser*, *exclude: parsec.Parser*)
Fails parser p if parser *exclude* matches

parsec.**fix**(*fn*)
Allow recursive parser using the Y combinator trick.

Note that this version still yields the stack overflow problem, and will be fixed in later version.

See also: https://github.com/sighingnow/parsec.py/issues/39.

parsec.**generate**(*fn*)
Parser generator. (combinator syntax).

parsec.**joint**(**parsers*)
Joint two or more parsers, implements the operator of *(+)*.

parsec.**letter**()
Parse a letter in alphabet.

parsec.**lookahead**(*p: parsec.Parser*)
Parses without consuming

parsec.**many**(*p*)
Repeat a parser 0 to infinity times. DO AS MUCH MATCH AS IT CAN. Return a list of values.

parsec.**many1**(*p*)
Repeat a parser 1 to infinity times. DO AS MUCH MATCH AS IT CAN. Return a list of values.

parsec.**mark**(*p*)
Mark the line and column information of the result of the parser *p*.

parsec.**none_of**(*s*)
Parses a char NOT from specified string.

parsec.**one_of**(*s*)
Parses a char from specified string.

parsec.**optional**(*p*, *default_value=None*)
'Make a parser as optional. If success, return the result, otherwise return default_value silently, without raising any exception. If default_value is not provided None is returned instead.

parsec.**parse**(*p*, *text*, *index=0*)
Parse a string and return the result or raise a ParseError.

parsec.**parsecapp**(*p*, *other*)
Returns a parser that applies the produced value of this parser to the produced value of *other*.

There should be an operator *(<*>)*, but that is impossible in Python.

parsec.**parsecmap**(*p*, *fn*)
Returns a parser that transforms the produced value of parser with *fn*.

parsec.**regex**(*exp*, *flags=0*)
Parses according to a regular expression.

parsec.**result**(*p*, *res*)
Return a value according to the parameter *res* when parse successfully.

parsec.**sepBy**(*p*, *sep*)
sepBy(p, sep) parses zero or more occurrences of p, separated by *sep*. Returns a list of values returned by *p*.

parsec.**sepBy1**(*p*, *sep*)
sepBy1(p, sep) parses one or more occurrences of *p*, separated by *sep*. Returns a list of values returned by *p*.

parsec.**sepEndBy**(*p*, *sep*)
sepEndBy(p, sep) parses zero or more occurrences of *p*, separated and optionally ended by *sep*. Returns a list of values returned by *p*.

parsec.**sepEndBy1**(*p*, *sep*)
sepEndBy1(p, sep) parses one or more occurrences of *p*, separated and optionally ended by *sep*. Returns a list of values returned by *p*.

parsec.**separated**(*p*, *sep*, *mint*, *maxt=None*, *end=None*)
Repeat a parser *p* separated by *s* between *mint* and *maxt* times.

- When *end* is None, a trailing separator is optional.
- When *end* is True, a trailing separator is required.
- When *end* is False, a trailing separator will not be parsed.

MATCHES AS MUCH AS POSSIBLE.

Return list of values returned by *p*.

parsec.**skip**(*pa*, *pb*)
Ends with a specified parser, and at the end parser consumed the end flag. Implements the operator of *(<<)*.

parsec.**space**()
Parses a whitespace character.

parsec.**spaces**()
Parses zero or more whitespace characters.

parsec.**string**(*s*)
Parses a string.

parsec.**times**(*p*, *mint*, *maxt=None*)
Repeat a parser between *mint* and *maxt* times. DO AS MUCH MATCH AS IT CAN. Return a list of values.

parsec.**try_choice**(*pa*, *pb*)
Choice one from two parsers with backtrack, implements the operator of *(^)*.

parsec.**unit**(*p: parsec.Parser*)
Converts a parser into a single unit. Only consumes input if the parser succeeds

CHAPTER 3

Usage and Contributor Guide

This project is open-source under The MIT License and you may use, distribute and modify this code with out limitation. The source code can be found at the site hosted on github.

- You can install this library using pip with the following command:

  ```
  pip install parsec
  ```

  or download it from Pypi and then install manually.
- Looking for specific information? Try the *detailed documentation*.
- Report bugs with parsec.py in our Issue tracker.
- Contribute your code to help me improve this library with Pull Request.

Python Module Index

p

Index

S

T

U

V