<https://python.plainenglish.io/writing-an-interpreter-in-python-from-scratch-part-2-fdddc591941>

<https://gist.github.com/umangshrestha/890cc1b61169f7938f236722cfbc59a5#file-regex_tokens-py>

#-------------------------------------------------------------------------------

# lexer.py

#

# A generic regex-based Lexer/tokenizer tool.

# See the if \_\_main\_\_ section in the bottom for an example.

#

# Eli Bendersky (eliben@gmail.com)

# This code is in the public domain

# Last modified: August 2010

#-------------------------------------------------------------------------------

import re

import sys

class Token(object):

""" A simple Token structure.

Contains the token type, value and position.

"""

def \_\_init\_\_(self, type, val, pos):

self.type = type

self.val = val

self.pos = pos

def \_\_str\_\_(self):

return '%s(%s) at %s' % (self.type, self.val, self.pos)

class LexerError(Exception):

""" Lexer error exception.

pos:

Position in the input line where the error occurred.

"""

def \_\_init\_\_(self, pos):

self.pos = pos

class Lexer(object):

""" A simple regex-based lexer/tokenizer.

See below for an example of usage.

"""

def \_\_init\_\_(self, rules, skip\_whitespace=True):

""" Create a lexer.

rules:

A list of rules. Each rule is a `regex, type`

pair, where `regex` is the regular expression used

to recognize the token and `type` is the type

of the token to return when it's recognized.

skip\_whitespace:

If True, whitespace (\s+) will be skipped and not

reported by the lexer. Otherwise, you have to

specify your rules for whitespace, or it will be

flagged as an error.

"""

# All the regexes are concatenated into a single one

# with named groups. Since the group names must be valid

# Python identifiers, but the token types used by the

# user are arbitrary strings, we auto-generate the group

# names and map them to token types.

#

idx = 1

regex\_parts = []

self.group\_type = {}

for regex, type in rules:

groupname = 'GROUP%s' % idx

regex\_parts.append('(?P<%s>%s)' % (groupname, regex))

self.group\_type[groupname] = type

idx += 1

self.regex = re.compile('|'.join(regex\_parts))

self.skip\_whitespace = skip\_whitespace

self.re\_ws\_skip = re.compile('\S')

def input(self, buf):

""" Initialize the lexer with a buffer as input.

"""

self.buf = buf

self.pos = 0

def token(self):

""" Return the next token (a Token object) found in the

input buffer. None is returned if the end of the

buffer was reached.

In case of a lexing error (the current chunk of the

buffer matches no rule), a LexerError is raised with

the position of the error.

"""

if self.pos >= len(self.buf):

return None

else:

if self.skip\_whitespace:

m = self.re\_ws\_skip.search(self.buf, self.pos)

if m:

self.pos = m.start()

else:

return None

m = self.regex.match(self.buf, self.pos)

if m:

groupname = m.lastgroup

tok\_type = self.group\_type[groupname]

tok = Token(tok\_type, m.group(groupname), self.pos)

self.pos = m.end()

return tok

# if we're here, no rule matched

raise LexerError(self.pos)

def tokens(self):

""" Returns an iterator to the tokens found in the buffer.

"""

while 1:

tok = self.token()

if tok is None: break

yield tok

if \_\_name\_\_ == '\_\_main\_\_':

rules = [

('\d+', 'NUMBER'),

('[a-zA-Z\_]\w+', 'IDENTIFIER'),

('\+', 'PLUS'),

('\-', 'MINUS'),

('\\*', 'MULTIPLY'),

('\/', 'DIVIDE'),

('\(', 'LP'),

('\)', 'RP'),

('=', 'EQUALS'),

]

lx = Lexer(rules, skip\_whitespace=True)

lx.input('erw = \_abc + 12\*(R4-623902) ')

try:

for tok in lx.tokens():

print(tok)

except LexerError as err:

print('LexerError at position %s' % err.pos)

<https://onlinegdb.com/3_8hrrehL>

<https://gist.github.com/eliben/5797351>

|  |
| --- |
| import ply.lex as lex |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| tokens = [ 'NAME','NUMBER','PLUS','MINUS','TIMES','DIVIDE', 'EQUALS' ] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| t\_ignore = ' \t' |
|  |

|  |
| --- |
| t\_PLUS = r'\+' |
|  |

|  |
| --- |
| t\_MINUS = r'-' |
|  |

|  |
| --- |
| t\_TIMES = r'\\*' |
|  |

|  |
| --- |
| t\_DIVIDE = r'/' |
|  |

|  |
| --- |
| t\_EQUALS = r':=' |
|  |

|  |
| --- |
| t\_NAME = r'[a-zA-Z\_][a-zA-Z0-9\_]\*' |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def t\_NUMBER(t): |
|  |

|  |
| --- |
| r'\d+' |
|  |

|  |
| --- |
| t.value = int(t.value) |
|  |

|  |
| --- |
| return t |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Error handling rule |
|  |

|  |
| --- |
| def t\_error(t): |
|  |

|  |
| --- |
| print("Illegal character '%s'" % t.value[0]) |
|  |

|  |
| --- |
| t.lexer.skip(1) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lex.lex() # Build the lexer |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| lex.input("x = 3 - 4 + 5 \* 6") |
|  |

|  |
| --- |
| while True: |
|  |

|  |
| --- |
| tok = lex.token() |
|  |

|  |
| --- |
| if not tok: break |
|  |

print str(tok.value) + " - " + str(tok.type)

https://www.geeksforgeeks.org/how-to-validate-form-using-regular-expression-in-javascript/