pseudo Documentation

Release 0.5.2

Patryk Niedzwiedzinski

May 06, 2019

Contents

CHAPTER 1

Getting started

1.1 Install

1. Download executable of latest release.
2. Extract archive
3. Add folder with extracted files to PATH

If you're on macOs or Linux you probably have python3 installed. Then it will be easier to install it with *Pip*.

1.2 Install from source

1.2.1 Pip

You need to have python3.6 or greater. You can install pseudo with:

```
python3 -m pip install git+https://github.com/pniedzwiedzinski/pseudo.git
```

1.2.2 Docker

Download docker and follow instructions:

```
docker pull pniedzwiedzinski/pseudo

# Create alias
alias pdc='docker run -it --rm -v $(pwd):/home pseudo'
```

1.3 Usage

You can now use pseudo as:

```
pdc test.pdc
```

CHAPTER 2

Runtime

This module contains class of stream object used to iterate over input.

exception pseudo.stream.**EndOfFile**
Exception indicating that parsing ends.

class pseudo.stream.**Stream**(*input: str*)
Stream is an object used to iterate over input. It is a little bit similar to queue.

- input
List of lines of code.

- line
Number of current line. Counting from 1

- col
Number of current column. Counting from 1

eof() → bool
Returns true if next line is end of file and next char is end of line.

eol() → bool
Returns true if next char is end of line.

get_current_line()
Returns current line

next() → str
Move cursor to next column and return char from this postion.

next_line()
Move cursor to next line.

peek(*size: int = 0*) → Union[str, pseudo.type.base.EOL]
Returns next char without moving cursor. If next char does not exists it returns EOL instance.

Parameters **size** (–) – Size of shift, default *0*.

throw(*error: str*)
Used to display error messages with line number. It stops the execution.

CHAPTER 3

Runtime

After the pseudocode is parsed, runtime object (*pseudo.runtime.RunTime*) is initialized. All operations are evaluated in runtime context. In runtime there is also virtual memory (dict), which contains variables and their values.

This module contains RunTime class, which is a representation of runtime resources.

class pseudo.runtime.**MemoryObject**(*key: str*, *value: object*, *const: bool = False*)

This class is a representation of object in runtime memory.

- value

object, Stored object.

- const

bool, If true object is constant and cannot be changed.

getter()

This function returns value.

setter(*value: object*, *r*)

This function updates value.

class pseudo.runtime.**RunTime**

This class is a representation of computer resources like memory or processor. It is used to run instructions parsed by *pseudo.lexer.Lexer*.

- var

dict, In this object all variables will be stored.

Variable names:

Variable names consist of alphanumeric characters, starting with alphabetical char. Arrays are stored as independent keys i.e.:

```
{
    "T[1]": 1,
    "T[2]": 2,
}
```

Usage::

```
>>> instructions = [Statement("pisz", Int(42))]
>>> r = RunTime()
>>> r.run(instructions)
42
```

delete(*key: str*)
This function removes variable from memory.

eval(*instruction*)
Evaluate instruction.

get(*key: str*)
This function returns value of stored variable.

Parameters **key** (–) – str, Key under which value is stored.

run(*instructions: list*)
Run pseudocode instructions

save(*key: str*, *value: object*, *object_class=<class 'pseudo.runtime.MemoryObject'>*)
This functions is used to save variable's value in memory

Parameters

- **key** (–) – str, Unique key under which value will be stored. *T[1][10]* is also a key
- **value** (–) – object, Value to store.
- **object_class** (–) – class, Class of value.

static save_crash(*error_message: str*)
Save crash message to file and return path to it.

stdin(*key: str*)
This function reads value from standard input and stores it in given variable.

stdout(*value: object*)
This function writes value to standard output.

throw(*error_message: str*, *line_causing_error: str = ''*)
This function is used to tell user that a runtime error has occurred.

CHAPTER 4

Type

4.1 Basic types

When building abstract syntax tree each value is represented as node. This node must be a subclass of *Value* class.

base: This module contains base class for all objects representing value in AST

class pseudo.type.base.**EOL**
Representation of newline.

class pseudo.type.base.**Value**(*value*, *line=''*)
Node containing a value.

- **value**
  Value of instance.

- **line**
  Line of pseudocode representation

numbers: This module contains functions and classes for handling numeric objects in pseudocode.

class pseudo.type.numbers.**Int**(**args*, ***kwargs*)
Int value node.

pseudo.type.numbers.**is_digit**(*c*) → bool
Checks if given char is a digit.

pseudo.type.numbers.**read_number**(*lexer*) → pseudo.type.numbers.Int
Read a number from the stream.

string: This module contains everything about strings.

class pseudo.type.string.**String**(*value*, *line=''*)
String value node.

pseudo.type.string.**read_string**(*lexer*) → pseudo.type.string.String
Read a string from the stream.

bool: This module contains everything about bool in pseudocode.

class pseudo.type.bool.**Bool**(*value*, *line=''*)
Bool value node.

static eval_operation(*operator: str*, *left: pseudo.type.base.Value*, *right: pseudo.type.base.Value*, *r*)
Eval bool operation.

Parameters
- **operator** (–) – str, bool operator
- **left** (–) – Value
- **right** (–) – Value

pseudo.type.bool.**read_bool**(*keyword: str*) → pseudo.type.bool.Bool
Parse bool str to Bool object.

4.2 Advanced types

Here are some advanced types used to build abstract syntax tree.

variable: This module contains all about variables in pseudocode.

class pseudo.type.variable.**Assignment**(*target: pseudo.type.variable.Variable*, *value: pseudo.type.base.Value*, *object_class=<class 'pseudo.runtime.MemoryObject'>*, *line: str = ''*)
Node for representing assignments.

– target
Target variable.

– value
Value to assign.

class pseudo.type.variable.**Increment**(*key: str*, *line: str = ''*)
Representing incrementation of iterator.

– key
str, Key of iterator.

– line
str, Representation of operation in pseudocode.

class pseudo.type.variable.**Variable**(*value*, *indices=[]*)
Node for representing variables.

– value
Name of the variable.

– indices
List of indices.

conditional: This module contains everything about conditional statements in pseudocode.

class pseudo.type.conditional.**Condition**(*condition*, *true*, *false=None*, *line=''*)
Node for representing conditional expressions (if).

– condition
Condition to check

– true
List to evaluate if condition is true

- **false**
  List to evaluate if condition is false (optional)

- **line**
  String of pseudocode representation

pseudo.type.conditional.**read_if**(*lexer*, *indent_level: int = 0*) → pseudo.type.conditional.Condition

Read if statement from stream. This function expect stream cursor be after if keyword:

```
'if condition then'
   ^
```

Parameters

- **lexer** (–) – Lexer object used to apply lexing rules.
- **indent_level** (–) – int, Indicates level of indentation passed to children.

operation: This module contains classes for representing operations in AST.

class pseudo.type.operation.**Operation**(*operator*, *left*, *right*)
Operation node.

class pseudo.type.operation.**Operator**(*value*, *line=''*)
Opartor class for representing mathematical operator.

pseudo.type.operation.**is_operator**(*c*) → bool
Checks if given char is an allowed operator.

pseudo.type.operation.**read_operator**(*stream*)
Return operator from input stream.

Parameters **stream** (–) – Input stream

Writing actual code might be hard to understand for new-learners. Pseudocode is a tool for writing algorithms without knowing how to code. This module contains classes and methods for parsing pseudocode to AST and then evaluating it.

Example

If you installed this module with pip you can run pseudocode from file, i.e. to run *test.pdc* file type:

```
$ pdc test.pdc
```

If you want to parse it by your own you will need *pseudo.lexer.Lexer* instance.:

```
from pseudo.lexer import Lexer

lex = Lexer("x := 12")

expression = lex.read_next()
print(expression)
```

If lexer reach the end of input, the *pseudo.stream.EndOfFile* exception will be raised.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

T

V