MarkdownToLaTeX

Release 0.0.2

gitcordier

Jun 23, 2023

CONTENTS:

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

1.1 Contents

1.1.1 Usage

Installation

To use **MarkdownToLaTeX**, first install it using pip:

```
$ pip3 install markdowntolatex
```

1.1.2 API

Markdown

Unicode UTF-8 code points of characters that play a special role: LF, #, , … , and so on. Note for Windows users: the CRLF EOL is not taken into account. Please switch to LF.

`markdowntolatex.markdown.encoding.`**`ASTERISK = 42`**

*, code point: U+002A.

`markdowntolatex.markdown.encoding.`**`BACKSLASH = 92`**

\, code point: U+005C.

`markdowntolatex.markdown.encoding.`**`DOLLAR = 36`**

$, code point: U+0024.

`markdowntolatex.markdown.encoding.`**`HASH = 35`**

#, code point: U+0023.

markdowntolatex.markdown.encoding.**LATEX_BACKSLASH = [92, 98, 97, 99, 107, 115, 108, 97, 115, 104]**

backslah = (\, b, a, c, k, s, l, a, s, h)

markdowntolatex.markdown.encoding.**LATEX_NEWLINE = [92, 92]**

newline = (\, \)

markdowntolatex.markdown.encoding.**LF = 10**

LF, code point: U+000A.

markdowntolatex.markdown.encoding.**SPACE = 32**

‘ ‘ , code point: U+0020.

markdowntolatex.markdown.encoding.**SPECIAL_CHARACTERS = {10, 32, 35}**

LF, hash, space are the *special characters*.

markdowntolatex.markdown.encoding.**UNDERSCORE = 95**

_,code point: U+005F.

The Markdown parser.

class markdowntolatex.markdown.parser.**Parser**(*dialect='Github'*)

interpret(*read*)

The automaton that rules the markdown parsing.

Parameters
read – the currently read character, as an int.

Type
int

Raise
ValueError iff read is not a legal code point.

TODO: Define the legal range for code points. TODO: Implement ValueError.

is_count_even(*key*)

Param
key, a key for the dictionary *count*.

Type
str

Returns
True iff self.count[*key*] is an even interger.

Return type
Boolean

Raise
TypeError iff self.count[*key*] is not an integer.

TODO: Implement exceptions management.

is_count_odd(*key*)

Param
key, a key for the dictionary *count*.

Type
str

Returns
True iff self.count[*key*] is an odd interger.

Return type
Boolean

Raise
TypeError iff self.count[*key*] is not an integer.

TODO: Implement exceptions management.

`is_count_positive`(*key*)

Param
key, a key for the dictionary *count*.

Type
str

Returns
True iff self.count[*key*] is positive.

Return type
Boolean

Raise
TypeError iff self.count[*key*] is not an integer.

TODO: Implement exceptions management.

`is_count_positive_even`(*key*)

Param
key, a key for the dictionary *count*.

Type
str

Returns
True iff self.count[*key*] is positive even.

Return type
Boolean

Raise
TypeError iff self.count[*key*] is not an integer.

TODO: Implement exceptions management.

`is_count_positive_odd`(*key*)

Param
key, a key for the dictionary *count*.

Type
str

Returns
True iff self.count[*key*] is positive even.

Return type
Boolean

Raise
TypeError iff self.count[*key*] is not an integer.

TODO: Implement exceptions management.

`static is_regular_character`(*read*)

A character is a regular character **iff** it is not a *special character*.

`static latex`(*command*, *card=1*, *line_break=False*)

Given a LaTex command *command* and a cardinality *card*, returns the sequence *command* + ... + *command* (*card* time(s)).

Optionally, appends n (n = 0, 1, 2, ...) line break(s) at the end of the sequence **iff** 'line_break' is set to n.

Note that (1) *line_break=True* means *line_break=1*, (2) *line_break=False* means *line_break=0*.

TODO: Exceptions (card) :param command: plain text that denotes a LaTeX command ("backslash", "newline", ...). :param card: the number of times *command* is repeated. :type card: int :param line_break: Appended line break(s). :type line_break: int :return: A LaTeX command, as a list of code points. :rtype: list

`reset_count`(*key*)

Resets self.count[*key*] to 0.

Param
key a key for the dictionary *count*.

Type
str

Raise
KeyError iff key is not a key for count.

`set_count`(*key*, *n*)

`set_level`(*level*)

the inner **level** is updated with respect to the given value of *level*.

`set_state`(*state*)

state is assigned the value of *state*.

`update_count`(*key*, *n=1*)

Given a counting number n=1, 2, 3, ..., @increases self.count[*key*] by n.

Param
key, a key for the dictionary *count*.

Type
str

Param
n the value that adds to self.count[*key*].

Type
int

Raise
KeyError iff key is not a key for count.

Raise
TypeError iff n is not an integer

Raise
ValueError iff n is not a positive integer.

update_heading(*read*, *head=None*, *prefix=None*, *tail=None*, *suffix=None*)

Given *read**, updates the heading.

Parameters

- **read** (*int*) – The read character.
- **prefix** (*list*) – An optional prefix
- **tail** (*list*) – An optional tail
- **suffix** (*list*) – An optional suffix (if no **tail**)

update_mode(*mode=None*)

Sets **self.mode** to **mode**.

Param
mode, a mode identifier.

Type
str

Raise
ValueError iff **mode** is not a legal mode.

TODO: Implement exceptions management.

update_text(*read*, *head=None*, *prefix=None*, *tail=None*, *suffix=None*)

Given *read**, updates the text.

Parameters

- **read** (*int*) – The read character.
- **prefix** (*list*) – An optional prefix
- **tail** (*list*) – An optional tail
- **suffix** (*list*) – An optional suffix (if no **tail**)

Updates the (core) text that will be injected into the LaTeX code source.

update_tree()

Updates the current tree by appending a node at the upper right extremity. If the tree does not exist yet, the node becomes the tree.

class markdowntolatex.markdown.tree.**Tree**(*height*, *level*)

Tree is the tree

Parameters

- **height** (*int*) – The tree's height at instantiation. Positive integer.
- **level** – The tree's level at instantiation. Positive integer.
- **type** – int

TODO: Exceptions

add_branch(*level*, *subtree*)

Adds branch (a "subtree") to **self.tree**.

The height of the tree is updated accordingly.

`find_diagonal_up()`

Find the upper diagonal element and returns it.

Returns
The subtree (should be a **Node**)

Return type
Tree

TODO: Warning if diagonal element is not a **Node**

`get_level_max()`

return: *level_max* (see mathsheet)

`to_string(arg, whitespaces=True)`

TODO: !

`update_diagonal(subtree)`

TODO: Compare with **add_branch**.

`update_heading(read)`

updates the curent heading with *read*.

`update_text(read)`

updates the curent text heading with *read*.

LaTeX

The LaTeX part. **Document** is an abstraction for a *document*. A **Document** is instantiated from a preferences file.

IF preferences contains a single **JSON** file, **THEN** such file is the preferences file.

ELIF preferences contains more than one **JSON** file, **THEN** raise **FileNotFoundError**.

ELIF preferences contains no **JSON** file, **THEN** look for *user_preferences* input:

IF input is in **{None, ''}**, **THEN** use the default preferences.

ELIF input is a valid path, **THEN** preferences are set.

ELSE raise **FileNotFoundError**

`class markdowntolatex.latex.document.Document(user_preferences=None)`

Abstraction for document.

user_preferences is a path to a JSON preferences file.

Parameters
user_preferences – A path to a *preferences* file.

Type
str

Raise
FileNotFoundError if the path is not valid.

Raise
JSONDecodeError if the preferences file being deserialized is not a valid JSON document.

get_latex()

From Markdown to LaTeX.

Creates the latex code after the parsing is done.

Formally speaking, this method is a getter, since a dictionary dict {'folder', 'document'} is returned.

Returns
A dictionary dict{folder, document}.

Return type
dict

get_markdownn(*arg*)

parse_markdown()

Parses the document.

User

The **User** package is dedicated to interaction with the user. It provides a command line interface (CLI) implementation and a record that keeps track of the user's choices.

Choice is an abstraction for the User's choice.

class markdowntolatex.user.choice.**Choice**

User's inputs are collected then recorded into a dictionary, which is the **Choice** instance itself.

markdown_to_latex()

xelatex_to_pdf()

markdowntolatex.user.choice.**latex_to_pdf**()

Performs the whole "LaTex to PDF" process.

When the binary is run, this method is called hunder the hood.

markdowntolatex.user.choice.**markdown_to_latex**()

Performs the whole "markdown to LaTex" process.

When the binary is run, this method is called hunder the hood.

markdowntolatex.user.choice.**markdown_to_pdf**()

Performs the whole "Markdown to PDF" process.

When the binary is run, this method is called hunder the hood.

We now come to the

Command line interface definition. A dictionary **ARGUMENT** stores all arguments definitions.

markdowntolatex.user.cli.**MAX_NUMBEROF_INPUTS = 3**

The maximal number of input strings we expect from the user: Two arguments (help, preferences), one value (for preferences). Three, then

Utilities

Misc functions.

markdowntolatex.utilities.**get_file**(*name*, **prefix*, ***kwargs*)

Fetch a file *name* the from **package_data** folder (only).

Parameters
name (*str*) – The file's name.

Raises

- **ModuleNotFoundError** – If the package metadata do not mention the package name.
- **FileNotFoundError** – If the folder **package_data** do not exist.

Returns
The desired file, as a string or a byte array

Return type
str or byte array

PYTHON MODULE INDEX

m

INDEX

P

R

S

T

U

X