# SnakeMD

The Renegade Coder

Mar 31, 2023

# CONTENTS

1 Installation 3

2 Usage 5

3 Documentation 7

4 Version History 25

Python Module Index 29

Index 31

SnakeMD is a library for generating markdown files using Python. Use the links below to navigate the docs.

CHAPTER

ONE

# INSTALLATION

The quick and dirty way to install SnakeMD is to use pip:

```
pip install SnakeMD
```

CHAPTER

# TWO

# USAGE

SnakeMD is a Python library for building markdown documents. You can use it by importing the SnakeMD module into your program directly:

```
import snakemd
```

This way, you'll have access to all of the classes available in the SnakeMD module. From here, you can take advantage of a handy function to create a new document:

```
doc = snakemd.new_doc()
```

This will create a new Document object. Alternatively, you can import the Document class directly:

```
from snakemd import Document
```

From here, you can instantiate the Document class:

```
doc = Document()
```

While there is nothing in our document currently, we can render an empty one as follows:

```
doc.dump("README")
```

This will create an empty README.md file in our working directory. Of course, if we want something more interesting, we'll have to add some content to our document. To start, we'll add a title to the document:

```
doc.add_heading("Why Use SnakeMD?")
```

From here, we can do pretty much anything we'd like. Some quick actions might be to include a paragraph about this library as well as a list of reasons why you might use it:

```
p = doc.add_paragraph(
  """
  SnakeMD is a library for generating markdown, and here's
  why you might choose to use it:
  """
)
doc.add_unordered_list([
    "SnakeMD makes it easy to create markdown files.",
    "SnakeMD has been used to back of The Renegade Coder projects."
])
```

One thing that's really cool about using SnakeMD is that we can build out the structure of a document before we modify it to include any links. For example, you might have noticed that we saved the output of the add_paragraph() method from above. Well, as it turns out, all of the document methods return the objects that are generated as a result of their use. In this case, the method returns a Paragraph object which we can modify. Here's how we might insert a link to the docs:

```
p.insert_link("SnakeMD", "https://snakemd.therenegadecoder.com")
```

And if all goes well, we can output the results by outputting the document like before. Or, if we just need to see the results as a string, we can convert the document to a string directly:

```
print(doc)
```

And this is what we'll get:

```
# Why Use SnakeMD?

[SnakeMD](https://snakemd.therenegadecoder.com) is a library for generating markdown,␣
↪and here's why you might choose to use it:

- SnakeMD makes it easy to create markdown files.
- SnakeMD has been used to back of The Renegade Coder projects.
```

Feel completion, here is a complete working program to generate the document from above in a file called README.md:

```
import snakemd

doc = snakemd.new_doc()

doc.add_heading("Why Use SnakeMD?")
p = doc.add_paragraph(
  """
  SnakeMD is a library for generating markdown, and here's
  why you might choose to use it:
  """
)
doc.add_unordered_list([
    "SnakeMD makes it easy to create markdown files.",
    "SnakeMD has been used to back of The Renegade Coder projects."
])
p.insert_link("SnakeMD", "https://snakemd.therenegadecoder.com")

doc.dump("README")
```

As always, feel free to check out the rest of the usage docs for all of the ways you can make use of SnakeMD. If you find an issues, make sure to head over to the GitHub repo and let us know.

CHAPTER

# THREE

# DOCUMENTATION

The documentation page lists out all of the relevant classes and functions for generating markdown documents in Python.

## 3.1 The Document API

SnakeMD is designed with different types of users in mind. The main type of user is the person who wants to design and generate markdown quickly without worrying too much about the format or styling of their documents. To help this type of user, we've developed a high-level API which consists of a single function, `snakemd.new_doc()`. This function returns a Document object that is ready to be modified using any of the convenience methods available in the Document class. Both the `snakemd.new_doc()` function and the Document class are detailed below.

### 3.1.1 Module

The SnakeMD module contains all of the functionality for generating markdown files with Python. To get started, check out *Usage* for quickstart sample code.

snakemd.**new_doc**(*name: Optional[str] = None*) → *Document*

Creates a new SnakeMD document. This is a convenience function that allows you to create a new markdown document without having to import the Document class. This is useful for anyone who wants to take advantage of the procedural interface of SnakeMD. For those looking for a bit more control, each element class will need to be imported as needed.

```
doc = snakemd.new_doc()
```

New in version 0.9.0.

**Parameters**

**name** (*str*) – the file name of the document without the extension

Deprecated since version 0.13.0: parameter is now optional and will be removed in 1.0.0

**Returns**

a new Document object

### 3.1.2 Document

**Note:** All of the methods described in the Document class are assumed to work without any *snakemd.Element* imports. In circumstances where methods may make use of Elements, such as in `add_table()`, the snakemd module will be referenced directly in the sample source code.

For the average user, the document object is the only object in the library necessary to create markdown files. Document objects are automatically created from the *new_doc()* function of the SnakeMD module.

**class snakemd.Document**(*name: Optional[str] = None*)

Bases: `object`

A document represents a markdown file. Documents store a collection of blocks which are appended with new lines between to generate the markdown document. Document methods are intended to provided convenience when generating a markdown file. However, the functionality is not exhaustive. To get the full range of markdown functionality, you can take advantage of the *add_block()* function to provide custom markdown block.

**Parameters**
**name** (*str*) – the file name of the document without the extension

Deprecated since version 0.13.0: parameter is now optional and will be removed in 1.0.0

**add_block**(*block:* Block) → *Block*

A generic function for appending blocks to the document. Use this function when you want a little more control over what the output looks like.

```
doc.add_block(Heading("Python is Cool!", 2))
```

New in version 0.14.0: replaces *add_element()*

**Parameters**
**block** (Block) – a markdown block (e.g., Table, Heading, etc.)

**Returns**
the Block added to this Document

**add_checklist**(*items: Iterable[str]*) → *MDList*

A convenience method which adds a simple checklist to the document.

```
doc.add_checklist(["Okabe", "Mayuri", "Kurisu"])
```

New in version 0.10.0.

**Parameters**
**items** (*Iterable[str]*) – a "list" of strings

**Returns**
the MDCheckList added to this Document

**add_code**(*code: str*, *lang: str = 'generic'*) → *Code*

A convenience method which adds a code block to the document:

```
doc.add_code("x = 5")
```

Changed in version 0.2.0: returns Paragraph generated by this method instead of None

Changed in version 0.15.0: returns Code block generated by this method instead of Paragraph

**Parameters**

- **code** (*str*) – a preformatted code string
- **lang** (*str*) – the language for syntax highlighting

**Returns**
the Code block added to this Document

**add_element**(*element: Element*) → *Element*

A generic function for appending elements to the document. Use this function when you want a little more control over what the output looks like.

```
doc.add_element(Heading(Inline("Python is Cool!"), 2))
```

Changed in version 0.2.0: Returns Element generated by this method instead of None.

Deprecated since version 0.14.0: replaced in favor of *add_block()*

**Parameters**
**element** (Element) – a markdown object (e.g., Table, Heading, etc.)

**Returns**
the Element added to this Document

**add_header**(*text: str*, *level: int = 1*) → Header

A convenience method which adds a simple header to the document:

```
doc.add_header("Welcome to SnakeMD!")
```

Changed in version 0.2.0: returns Header generated by this method instead of None.

Deprecated since version 0.13.0: use *add_heading()* instead

**Parameters**

- **text** (*str*) – the text for the header
- **level** (*int*) – the level of the header from 1 to 6

**Returns**
the Header added to this Document

**add_heading**(*text: str*, *level: int = 1*) → *Heading*

A convenience method which adds a simple heading to the document:

```
doc.add_heading("Welcome to SnakeMD!")
```

New in version 0.13.0: replaces *add_header()*

**Parameters**

- **text** (*str*) – the text for the heading
- **level** (*int*) – the level of the heading from 1 to 6

**Returns**
the Heading added to this Document

**add_horizontal_rule**() → *HorizontalRule*

A convenience method which adds a horizontal rule to the document:

```
doc.add_horizontal_rule()
```

New in version 0.2.0.

**Returns**
the HorizontalRule added to this Document

**add_ordered_list**(*items: Iterable[str]*) → *MDList*

A convenience method which adds a simple ordered list to the document:

```
doc.add_ordered_list(["Goku", "Piccolo", "Vegeta"])
```

Changed in version 0.2.0: Returns MDList generated by this method instead of None.

**Parameters**
**items** (*Iterable[str]*) – a "list" of strings

**Returns**
the MDList added to this Document

**add_paragraph**(*text: str*) → *Paragraph*

A convenience method which adds a simple paragraph of text to the document:

```
doc.add_paragraph("Mitochondria is the powerhouse of the cell.")
```

Changed in version 0.2.0: Returns Paragraph generated by this method instead of None.

**Parameters**
**text** (*str*) – any arbitrary text

**Returns**
the Paragraph added to this Document

**add_quote**(*text: str*) → *Paragraph*

A convenience method which adds a blockquote to the document:

```
doc.add_quote("Welcome to the Internet!")
```

Changed in version 0.2.0: Returns Paragraph generated by this method instead of None.

**Parameters**
**text** (*str*) – the text to be quoted

**Returns**
the Paragraph added to this Document

**add_raw**(*text: str*) → *Raw*

A convenience method which adds text as-is to the document:

```
doc.add_raw("X: 5\nY: 4\nZ: 3")
```

**Parameters**
**text** (*str*) – some text

**Returns**
the Raw block added to this Document

**add_table**(*header: Iterable[str], data: Iterable[Iterable[str]], align: Optional[Iterable[Align]] = None, indent: int = 0*) → *Table*

A convenience method which adds a simple table to the document:

```
doc.add_table(
    ["Place", "Name"],
    [
        ["1st", "Robert"],
        ["2nd", "Rae"]
    ],
    [snakemd.Table.Align.CENTER, snakemd.Table.Align.RIGHT],
    0
)
```

Changed in version 0.2.0: Returns Table generated by this method instead of None.

Changed in version 0.4.0: Added optional alignment parameter

Changed in version 0.11.0: Added optional indentation parameter

**Parameters**

- **`header`** (*`Iterable[str]`*) – a "list" of strings
- **`data`** (*`Iterable[Iterable[str]]`*) – a "list" of "lists" of strings
- **`align`** (*`Iterable[Table.Align]`*) – a "list" of column alignment values; defaults to None
- **`indent`** (*`int`*) – indent size for the whole table

**Returns**

the Table added to this Document

**`add_table_of_contents`**(*levels: range = range(2, 3)*) → *TableOfContents*

A convenience method which creates a table of contents. This function can be called where you want to add a table of contents to your document. The table itself is lazy loaded, so it always captures all of the heading blocks regardless of where the table of contents is added to the document.

```
doc.add_table_of_contents()
```

Changed in version 0.2.0: Fixed a bug where table of contents could only be rendered once.

Changed in version 0.8.0: Added optional levels parameter

**Parameters**

**`levels`** (*`range`*) – a range of heading levels to be included in the table of contents

**Returns**

the TableOfContents added to this Document

**`add_unordered_list`**(*items: Iterable[str]*) → *MDList*

A convenience method which adds a simple unordered list to the document.

```
doc.add_unordered_list(["Deku", "Bakugo", "Kirishima"])
```

Changed in version 0.2.0: Returns MDList generated by this method instead of None.

**Parameters**

**`items`** (*`Iterable[str]`*) – a "list" of strings

**Returns**

the MDList added to this Document

**check_for_errors**() → None

A convenience method which can be used to verify the integrity of the document. Results will be printed to standard out.

New in version 0.2.0.

**dump**(*name: str*, *dir: str | os.PathLike = ''*, *ext: str = 'md'*, *encoding: str = 'utf-8'*) → None

Outputs the markdown document to a file. This method assumes the output directory is the current working directory. Any alternative directory provided will be made if it does not already exist. This method also assumes a file extension of md and a file encoding of utf-8, all of which are configurable through the method parameters.

```
doc.dump("README")
```

New in version 0.13.0: Replaces the `output_page()` method

**Parameters**

- **name** (*str*) – the name of the markdown file to output without the file extension
- **dir** (*str | os.PathLike*) – the output directory for the markdown file; defaults to ""
- **ext** (*str*) – the output file extension; defaults to "md"
- **encoding** (*str*) – the encoding to use; defaults to utf-8

**output_page**(*dump_dir: str = ''*, *encoding: str = 'utf-8'*) → None

Generates the markdown file. Assumes UTF-8 encoding.

Deprecated since version 0.13.0: Use `dump()` instead

**Parameters**

- **dump_dir** (*str*) – the path to where you want to dump the file
- **encoding** (*str*) – the encoding to use

**render**() → str

Renders the markdown document from a list of blocks.

Deprecated since version 0.14.0: removed in favor of __str__

**Returns**
the document as a markdown string

**scramble**() → None

A silly method which mixes all of the blocks in this document in a random order.

## 3.2 The Element API

For users who want a little more control over how their markdown is formatted, SnakeMD provides a low-level API constructed of elements.

### 3.2.1 Element Interface

**class** snakemd.**Element**

Bases: `ABC`

A generic element interface which provides a framework for all types of elements in the collection. In short, elements should be able to be verified.

**render**() → str

Renders the element as a markdown string. This function now just calls the __str__ method directly.

Deprecated since version 0.14.0: replaced with the default dunder method `__str__()`

**Returns**
the element as a markdown string

**abstract verify**() → *Verification*

Verifies that the element is valid markdown.

**Returns**
a verification object from the violator

Elements are then broken down into two main types: block and inline.

### 3.2.2 Block Elements

SnakeMD block elements borrow from the idea of block-level elements from HTML. And because Markdown documents are constructed from a series of blocks, users of SnakeMD can seemlessly append their own custom blocks using the `add_block()` method. To make use of this method, blocks must be imported and constructed manually, like the following Heading example:

```
from snakemd import Heading
doc.add_block(Heading("Hello, World!", 2))
```

The remainder of this section introduces the Block interface as well as all of the Blocks currently available for use.

#### Block Interface

All markdown blocks inherit from the Block interface.

**class** snakemd.**Block**

Bases: *`Element`*

A block element in Markdown. A block is defined as a standalone element starting on a newline. Examples of blocks include paragraphs (i.e., <p>), headings (e.g., <h1>, <h2>, etc.), tables (i.e., <table>), and lists (e.g., <ol>, <ul>, etc.).

New in version 0.14.0: replaced the *`Element`* class

## Verification

**Warning:** The verification object and its implementation throughout SnakeMD is in beta, and it should not be used to verify production markdown.

Because of the increase in control granted to users by blocks, there are opportunities where invalid markdown can be generated. In an attempt to provide a method of verifying the structure of the markdown, a `verify()` method has been provided for all elements. The result of a call to `verify()` is a verification object which is defined as folows:

`class snakemd.`**`Verification`**

Bases: `object`

Verification is a helper object for storing errors generated when creating a markdown document. This object is largely used internally to verify the contents of a document, but can be accessed through the various verify() methods throughout the library by the user. A convenience method is provided in Document for listing all of the errors. Otherwise, a handful of methods are available here for interacting with the Verification object directly.

New in version 0.2.0.

**`absorb`**(*verification:* Verification) → None

Absorbs an existing verification object in self. This is helpful when you have many verification objects that you'd like to aggregate.

**Parameters**

**`verification`** (`Verification`) – the verification object to absorb

**`add_error`**(*violator: object*, *error: str*) → None

Documents a verification error.

**Parameters**

- **`violator`** (*`object`*) – the object which produced the error
- **`error`** (*`str`*) – the error message detailing the error

**`passes_inspection`**() → bool

Assuming this object has already been used to verify something, this function will determine if that verificatioc succeeded.

**Returns**

True if there are no errors; False otherwise

The remainder of this page outlines the various elements that can be added to a markdown document.

## Code

`class snakemd.`**`Code`**(*code: str* | snakemd.elements.Code, *lang: str = 'generic'*)

Bases: *`Block`*

A code block is a standalone block of syntax-highlighted code. Code blocks can have generic highlighting or highlighting based on their language.

New in version 0.15.0.

**`verify`**() → *Verification*

Verifies that the element is valid markdown.

**Returns**
a verification object from the violator

## Heading

`class snakemd.Heading(text: Union[str, Inline, Iterable[snakemd.elements.Inline | str]], level: int)`

Bases: *Block*

A heading is a text block which serves as the title for a new section of a document. Headings come in six main sizes which correspond to the six headings sizes in HTML (e.g., <h1>).

**Parameters**

- **text** (*str | Inline | Iterable[Inline | str]*) – the heading text
- **level** (*int*) – the heading level between 1 and 6 (rounds to closest bound if out of range)

`demote()` → None

Demotes a heading down a level. Fails silently if the heading is already at the lowest level (i.e., <h6>).

`get_text()` → str

Returns the heading text free of any styling.

New in version 0.15.0.

**Returns**
the heading as a string

`promote()` → None

Promotes a heading up a level. Fails silently if the heading is already at the highest level (i.e., <h1>).

`verify()` → *Verification*

Verifies that the provided heading is valid. This mainly returns errors associated with the Inline element provided during instantiation.

New in version 0.2.0.

**Returns**
a verification object from the violator

## HorizontalRule

`class snakemd.HorizontalRule`

Bases: *Block*

A horizontal rule is a line separating different sections of a document. Horizontal rules really only come in one form, so there are no settings to adjust.

New in version 0.2.0.

`verify()` → *Verification*

Verifies the structure of the HorizontalRule block. Because there is no way to customize this object, it is always valid. Therefore, this method returns an empty Verification object.

New in version 0.2.0.

**Returns**
a verification object from the violator

### MDList

**class** snakemd.**MDList**(*items: Iterable[str* | snakemd.elements.Inline | snakemd.elements.Paragraph | snakemd.elements.MDList*]*, *ordered: bool = False*, *checked: Union[None, bool, Iterable[bool]] = None*)

Bases: *Block*

A markdown list is a standalone list that comes in three varieties: ordered, unordered, and checked.

Changed in version 0.4.0: Expanded constructor to accept strings directly

**Parameters**

- **items** (*Iterable[str | Inline | Paragraph | MDList]*) – a "list" of objects to be rendered as a list
- **ordered** (*bool*) – the ordered state of the list; set to True to render an ordered list (i.e., True -> 1. item)
- **checked** (*None | bool | Iterable[bool]*) – the checked state of the list; set to True, False, or an iterable of booleans to enable the checklist feature.

**verify**() → *Verification*

Verifies that the markdown list is valid. Mainly, this checks the validity of the containing Inline items. The MDList class has no way to instantiate it incorrectly, beyond providing the wrong data types.

New in version 0.2.0.

**Returns**

a verification object from the violator

### Paragraph

**class** snakemd.**Paragraph**(*content: Iterable[*snakemd.elements.Inline | *str]*, *code: bool = False*, *lang: str = 'generic'*, *quote: bool = False*)

Bases: *Block*

A paragraph is a standalone block of text. Paragraphs can be formatted in a variety of ways including as code and blockquotes.

Changed in version 0.4.0: Expanded constructor to accept strings directly

**Parameters**

- **content** (*Iterable[Inline | str]*) – a "list" of text objects to render as a paragraph
- **code** (*bool*) – the code state of the paragraph; set True to convert the paragraph to a code block (i.e., True -> `` `code` ``)

  Deprecated since version 0.15.0: replaced in favor of the *Code* block
- **lang** (*str*) – the language of the code snippet; invalid without the code flag set to True

  Deprecated since version 0.15.0: replaced in favor of the *Code* block
- **quote** (*bool*) – the quote state of the paragraph; set True to convert the paragraph to a blockquote (i.e., True -> > quote)

**add**(*text:* snakemd.elements.Inline | *str*) → None

Adds a text object to the paragraph.

Changed in version 0.4.0: Allows adding of strings directly

**Parameters**
: **`text`** – a custom Inline element

**`insert_link`**(*target: str*, *url: str*, *count: int = -1*) → *Paragraph*

A convenience method which inserts links in the paragraph for all matching instances of a target string. This method is modeled after `str.replace()`, so a count can be provided to limit the number of insertions. This method will not replace links of text that have already been linked. See replace_link() for that behavior.

```
paragraph.insert_link("Here", "https://therenegadecoder.com")
```

New in version 0.2.0.

Changed in version 0.5.0: Changed function to insert links at all instances of target rather than just the first instance

**Parameters**

- **`target`** (*str*) – the string to link
- **`url`** (*str*) – the url to link
- **`count`** (*int*) – the number of links to insert; defaults to -1 (all)

**Returns**
self

**`is_text`**() → bool

Checks if this Paragraph is a text-only block. If not, it must be a quote or code block.

New in version 0.3.0.

**Returns**
True if this is a text-only block; False otherwise

**`replace`**(*target: str*, *replacement: str*, *count: int = -1*) → *Paragraph*

A convenience method which replaces a target string with a string of the users choice. Like insert_link, this method is modeled after `str.replace()` of the standard library. As a result, a count can be provided to limit the number of strings replaced in the paragraph.

New in version 0.5.0.

**Parameters**

- **`target`** (*str*) – the target string to replace
- **`replacement`** (*str*) – the Inline object to insert in place of the target
- **`count`** (*int*) – the number of links to insert; defaults to -1

**Returns**
self

**`replace_link`**(*target: str*, *url: str*, *count: int = -1*) → *Paragraph*

A convenience method which replaces matching URLs in the paragraph with a new url. Like insert_link() and replace(), this method is also modeled after `str.replace()`, so a count can be provided to limit the number of links replaced in the paragraph. This method is useful if you want to replace existing URLs but don’t necessarily care what the anchor text is.

New in version 0.7.0.

**Parameters**

- **`target`** (*str*) – the string to link

- **url** (*str*) – the url to link
- **count** (*int*) – the number of links to replace; defaults to -1 (all)

**Returns**
self

**verify**() → *Verification*

Verifies that the Paragraph is valid.

New in version 0.2.0.

**Returns**
a verification object from the violator

**verify_urls**() → dict[str, bool]

Verifies all URLs in the paragraph. Results are returned in a dictionary where the URLs are mapped to their validity.

**Returns**
a dictionary of URLs mapped to their validity

### Raw

class snakemd.**Raw**(*text: str*)

Bases: *Block*

Raw blocks allow a user to insert text into the Markdown document without an processing. Use this block to insert raw Markdown or other types of text (e.g., Jekyll frontmatter).

New in version 0.14.0.

**verify**() → *Verification*

Verifies that the element is valid markdown.

**Returns**
a verification object from the violator

### Table

class snakemd.**Table**(*header: Iterable[str* | Inline | Paragraph*], body: Iterable[Iterable[str* | Inline | Paragraph*]] = []*, *align: Iterable[*Align*] = None*, *indent: int = 0*)

Bases: *Block*

A table is a standalone block of rows and columns. Data is rendered according to underlying Inline items.

Changed in version 0.4.0: Added optional alignment parameter and expanded constructor to accept strings

Changed in version 0.11.0: Added optional indentation parameter for the whole table

Changed in version 0.12.0: Made body parameter optional

**Parameters**

- **header** – the header row of labels
- **body** – the collection of rows of data
- **align** – the column alignment
- **indent** – indent size for the whole table

**class Align**(*value*)

Bases: Enum

Align is an enum only used by the Table class to specify the alignment of various columns in the table.

New in version 0.4.0.

**CENTER = 3**

**LEFT = 1**

**RIGHT = 2**

**add_row**(*row: Iterable[str* | snakemd.elements.Inline | snakemd.elements.Paragraph*]*) → None

Adds a row to the end of table.

New in version 0.12.0.

**verify**()

Verifies the integrity of the markdown table. There are various ways a user could instantiate this object improperly. For example, they may provide a body with roes that are not all equal width. Likewise, the header may not match the width of the body. Inline elements may also be malformed.

New in version 0.2.0.

**Returns**

a verification object from the violator

### 3.2.3 Inline Elements

One of the benefits of creating the Block elements directly over using the Document methods is the control users now have over the underlying structure and style. Now, instead of being bound to the string inputs, users can provide Inline elements directly. For example, maybe we want to be able to link a Heading. This is not exactly possible through the Document methods as even the returned Heading object has no support for linking. However, with Inline elements, we can create a custom Heading to our standards:

```
from snakemd import Heading
doc.add_block(Heading(Inline("Hello, World!", "https://snakemd.io"), 2))
```

The remainder of this section introduces the Inline class.

#### Inline

**class** snakemd.**Inline**(*text: str*, *url: Optional[str] = None*, *bold: bool = False*, *italics: bool = False*, *strikethrough: bool = False*, *code: bool = False*, *image: bool = False*)

Bases: *Element*

The basic unit of text in markdown. All components which contain text are built using this class instead of strings directly. That way, those elements capture all styling information.

New in version 0.14.0: replaced the InlineText

**Parameters**

- **text** (*str*) – the inline text to render
- **url** (*str*) – the link associated with the inline text

- **bold** (*bool*) – the bold state of the inline text; set to True to render bold inline text (i.e., True -> **bold**)
- **italics** (*bool*) – the italics state of the inline text; set to True to render inline text in italics (i.e., True -> *italics*)
- **strikethrough** (*bool*) – the strikethrough state of the inline text; set to True to render inline text with a strikethrough (i.e., True -> ~~strikethrough~~)
- **code** (*bool*) – the italics state of the inline text; set to True to render inline text as code (i.e., True -> *code*)
- **image** (*bool*) – the image state of the inline text; set to True to render inline text as an image; must include url parameter to render

**bold**() → *Inline*

Adds bold styling to self.

Changed in version 0.7.0: Modified to return previous bold state

**Returns**
self

**code**() → *Inline*

Adds code style to self.

New in version 0.7.0.

**Returns**
self

**is_text**() → bool

Checks if this Inline is a text-only element. If not, it must be an image, a URL, or an inline code snippet.

New in version 0.2.0.

**Returns**
True if this is a text-only element; False otherwise

**is_url**() → bool

Checks if the Inline object represents a URL.

**Returns**
True if the object has a URL; False otherwise

**italicize**() → *Inline*

Adds italics styling to self.

New in version 0.7.0.

**Returns**
self

**link**(*url: str*) → *Inline*

Adds URL to self.

New in version 0.7.0.

**Parameters**
**url** (*str*) – the URL to apply to this Inline element

**Returns**
self

`render()` → str

Renders self as a string. In this case, inline text can represent many different types of data from stylized text to inline code to links and images.

Deprecated since version 0.14.0: replaced with the default dunder method `__str__()`

**Returns**
the Inline object as a string

`reset()` → *Inline*

Removes all settings from self (e.g., bold, code, italics, url, etc.). All that will remain is the text itself.

New in version 0.7.0.

**Returns**
self

`strikethrough()` → *Inline*

Adds strikethrough styling to self.

New in version 0.12.0.

**Returns**
self

`unbold()` → *Inline*

Removes bold styling from self.

Changed in version 0.7.0: Modified to return previous bold state

**Returns**
self

`uncode()` → *Inline*

Removes code style from self.

New in version 0.7.0.

**Returns**
self

`unitalicize()` → *Inline*

Removes italics styling from self.

New in version 0.7.0.

**Returns**
self

`unlink()` → *Inline*

Removes URL from self.

New in version 0.7.0.

**Returns**
self

`unstrikethrough()` → *Inline*

Remove strikethrough styling from self.

New in version 0.12.0.

**Returns**
self

**verify**() → *Verification*

Verifies that the Inline object is valid.

New in version 0.2.0.

**Returns**

a verification object containing any errors that may have occured

**verify_url**() → bool

Verifies that a URL is a valid URL.

**Returns**

True if the URL is valid; False otherwise

## 3.3 The Template API

While the document and element APIs are available for folks who are already somewhat familiar with Markdown, a template system is slowly being developed for folks who are looking for a bit more convenience. Ultimately, these folks can expect support for typical document sections such as tables of contents, footers, and more.

### 3.3.1 Templates

To allow for templates to be integrated with documents seamlessly, the Template interface was developed to inherit directly from the Element interface, just like Block and Inline. Therefore, templates can also be verified.

`class snakemd.`**`Template`**

Bases: `Element`

A template element in Markdown. A template can be thought of as a subdocument or collection of blocks. The entire purpose of the Template interface is to provide a superclass for a variety of abstractions over the typical markdown features. For example, Markdown has no feature for tables of contents, but a template could be created to generate one automatically for the user. In other words, templates are meant to be conviences objects for our users.

New in version 0.14.0.

Below are a few existing templates.

### 3.3.2 TableOfContents

`class snakemd.`**`TableOfContents`**(*doc:* Document, *levels: range = range(2, 3)*)

Bases: `Template`

A Table of Contents is an block containing an ordered list of all the *<h2>* headings in the document by default. A range can be specified to customize which headings (e.g., *<h3>*) are included in the table of contents. This element can be placed anywhere in the document.

New in version 0.2.0.

Changed in version 0.8.0: Added optional levels parameter

**Parameters**

- **doc** (`Document`) – a reference to the document containing this table of contents
- **levels** (*list[int]*) – a range of integers representing the sequence of heading levels to include in the table of contents; defaults to range(2, 3)

**`verify()`** → *Verification*

A Table of Contents is generated through a circular reference to the Document it contains. There is no way to instantiate this incorrectly.

New in version 0.2.0.

**Returns**
a verification object from the violator

# CHAPTER FOUR

# VERSION HISTORY

**Note:** All versions of documentation are left in the condition in which they were generated. At times, the navigation may look different than expected.

In an effort to keep history of all the documentation for SnakeMD, we've included all old versions below as follows:

- **v0.15.0 [#97, #98, #99, #101]**
  - Moved README generation code to repo root as a script
  - Expanded Heading constructor to support list of strings and Inline objects
  - Migrated code block support from Paragraph class into new Code class
- **v0.14.0 [#84, #86, #89, #90, #91, #95]**
  - Added Raw block for user formatted text
  - Replaced InlineText with Inline
  - Added Block and Inline classes
  - Deprecated MDCheckList and CheckBox
  - Replaced render with bulit-in str method
- **v0.13.0 [#71, #74, #76, #78, #80, #82]**
  - Created a replacement method for output_page called dump
  - Renamed Header class to Heading
  - Included deprecation warnings for both output_page and header as well as others affected
- **v0.12.0 [#65, #66]**
  - Added support for table generation on-the-fly (#64)
  - Reworked documentation to include proper headings and organization
  - Added support for strikethrough on InlineText elements (#58)
- **v0.11.0 [#61, #62]**
  - Added support for table indentation
- **v0.10.1 [#59]**
  - Enforced UTF-8 encoding in the output_page method (#54)
- **v0.10.0 [#55, #56, #57]**

  - Added the CheckBox class for creating checkboxes
  - Added the MDCheckList class for creating lists of checkboxes
  - Added a Document method for implementing easy checklists
  - Updated README to include a new section on checklists
- **v0.9.3 [#50, #49]**
  - Added multiple versions of Python testing
  - Restricted package to Python version 3.8+
  - Added Markdown linting for main README
- **v0.9.0 [#47, #46, #45]**
  - Added convenience function for creating new Document objects (#40)
  - Ported documentation to Read the Docs (#43)
- **v0.8.1**
  - Fixed an issue where nested lists did not render correctly
- **v0.8.0**
  - Added range feature to Table of Contents (#41)
- **v0.7.0**
  - Added replace_link() method to Paragraph
  - Added various state methods to InlineText
  - Expanded testing
  - Lowered log level to INFO for verify URL errors
  - Added code coverage to build
- **v0.6.0**
  - Restructured api, so snakemd is the import module
  - Updated usage page to show more features
  - Fixed issue where base docs link would reroute to index.html directly
- **v0.5.0**
  - Added favicon to docs (#26)
  - Added mass URL verification function to Paragraph class (#27)
  - Expanded testing to ensure code works as expected
  - Changed behavior of insert_link() to mimic str.replace() (#19)
  - Added a replace method to Paragraph (#27)
  - Added plausible tracking to latest version of docs (#25)
- **v0.4.1**
  - Added support for Python logging library (#22)
  - Expanded support for strings in the Header, Paragraph, and MDList classes
  - Fixed an issue where Paragraphs would sometimes render unexpected spaces (#23)

  - Added GitHub links to version history page
  - Added support for column alignment on tables (#4)
  - Fixed issue where tables sometimes wouldn't pretty print properly (#5)
- **v0.3.0 [#21]**
  - Gave documentation a major overhaul
  - Added support for paragraphs in MDList
  - Added is_text() method to Paragraph
  - Fixed issue where punctuation sometimes rendered with an extra space in front
- **v0.2.0 [#17]**
  - Added support for horizontal rules
  - Added automated testing through PyTest and GitHub Actions
  - Added document verification services
  - Added documentation link to README as well as info about installing the package
  - Fixed table of contents single render problem
  - Added a feature which allows users to insert links in existing paragraphs
- **v0.1.0**
  - Added support for links, lists, images, tables, code blocks, and quotes
  - Added a table of contents feature

# PYTHON MODULE INDEX

s

snakemd, 7

# INDEX

## A

absorb() (*snakemd.Verification method*), 14
add() (*snakemd.Paragraph method*), 16
add_block() (*snakemd.Document method*), 8
add_checklist() (*snakemd.Document method*), 8
add_code() (*snakemd.Document method*), 8
add_element() (*snakemd.Document method*), 9
add_error() (*snakemd.Verification method*), 14
add_header() (*snakemd.Document method*), 9
add_heading() (*snakemd.Document method*), 9
add_horizontal_rule() (*snakemd.Document method*), 9
add_ordered_list() (*snakemd.Document method*), 10
add_paragraph() (*snakemd.Document method*), 10
add_quote() (*snakemd.Document method*), 10
add_raw() (*snakemd.Document method*), 10
add_row() (*snakemd.Table method*), 19
add_table() (*snakemd.Document method*), 10
add_table_of_contents() (*snakemd.Document method*), 11
add_unordered_list() (*snakemd.Document method*), 11

## B

Block (*class in snakemd*), 13
bold() (*snakemd.Inline method*), 20

## C

CENTER (*snakemd.Table.Align attribute*), 19
check_for_errors() (*snakemd.Document method*), 11
Code (*class in snakemd*), 14
code() (*snakemd.Inline method*), 20

## D

demote() (*snakemd.Heading method*), 15
Document (*class in snakemd*), 8
dump() (*snakemd.Document method*), 12

## E

Element (*class in snakemd*), 13

## G

get_text() (*snakemd.Heading method*), 15

## H

Heading (*class in snakemd*), 15
HorizontalRule (*class in snakemd*), 15

## I

Inline (*class in snakemd*), 19
insert_link() (*snakemd.Paragraph method*), 17
is_text() (*snakemd.Inline method*), 20
is_text() (*snakemd.Paragraph method*), 17
is_url() (*snakemd.Inline method*), 20
italicize() (*snakemd.Inline method*), 20

## L

LEFT (*snakemd.Table.Align attribute*), 19
link() (*snakemd.Inline method*), 20

## M

MDList (*class in snakemd*), 16
module
    snakemd, 7

## N

new_doc() (*in module snakemd*), 7

## O

output_page() (*snakemd.Document method*), 12

## P

Paragraph (*class in snakemd*), 16
passes_inspection() (*snakemd.Verification method*), 14
promote() (*snakemd.Heading method*), 15

## R

Raw (*class in snakemd*), 18
render() (*snakemd.Document method*), 12
render() (*snakemd.Element method*), 13
render() (*snakemd.Inline method*), 20

replace() (*snakemd.Paragraph method*), 17
replace_link() (*snakemd.Paragraph method*), 17
reset() (*snakemd.Inline method*), 21
RIGHT (*snakemd.Table.Align attribute*), 19

## S

scramble() (*snakemd.Document method*), 12
snakemd
    module, 7
strikethrough() (*snakemd.Inline method*), 21

## T

Table (*class in snakemd*), 18
Table.Align (*class in snakemd*), 18
TableOfContents (*class in snakemd*), 22
Template (*class in snakemd*), 22

## U

unbold() (*snakemd.Inline method*), 21
uncode() (*snakemd.Inline method*), 21
unitalicize() (*snakemd.Inline method*), 21
unlink() (*snakemd.Inline method*), 21
unstrikethrough() (*snakemd.Inline method*), 21

## V

Verification (*class in snakemd*), 14
verify() (*snakemd.Code method*), 14
verify() (*snakemd.Element method*), 13
verify() (*snakemd.Heading method*), 15
verify() (*snakemd.HorizontalRule method*), 15
verify() (*snakemd.Inline method*), 21
verify() (*snakemd.MDList method*), 16
verify() (*snakemd.Paragraph method*), 18
verify() (*snakemd.Raw method*), 18
verify() (*snakemd.Table method*), 19
verify() (*snakemd.TableOfContents method*), 23
verify_url() (*snakemd.Inline method*), 22
verify_urls() (*snakemd.Paragraph method*), 18