SnakeMD

The Renegade Coder

May 11, 2022

CONTENTS

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

INSTALLATION

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

```
pip install SnakeMD
```

USAGE

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

```python
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:

```python
doc = snakemd.new_doc("README")
```

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

```python
from snakemd import Document
```

From here, you can instantiate the Document class:

```python
doc = Document("README")
```

The argument we pass to the constructor is the name of the document, and we will use that name to reader a document called README.md:

```python
doc.output_page()
```

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:

```python
doc.add_header("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:

```python
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.
```

As always, feel free to check out the Documentation for all of the ways you can make use of SnakeMD.

DOCUMENTATION

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

3.1 snakemd

The snakemd module contains all of the functionality for generating markdown files with Python. To get started, check out Usage for information. Otherwise, see the rest of this document for details on all the functionality provided in snakemd.

snakemd.**new_doc**(*name: str*) → *snakemd.generator.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("README")
```

> New in version 0.9.0.

> > **Parameters name** – the file name of the document without the extension

> > **Returns** a new Document object

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

> Bases: `snakemd.generator.InlineText`

> A checkable box, based of InlineText. Supports all formats available via InlineText (eg. url, bold, italics, etc.)

> > **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*)

> > > • **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

- **checked** (*bool*) – the checkbox state, checked or not; set to True to render checkbox as checked

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.

> **Returns** the InlineText object as a string

class snakemd.generator.**Document**(*name: str*)

Bases: `object`

A document represents a markdown file. Documents store a collection of elements 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_element()` function to provide custom markdown elements.

> **Parameters** `name` – the file name of the document without the extension

add_checklist(*items: Iterable[str]*) → *snakemd.generator.MDCheckList*

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'*) → *snakemd.generator.Paragraph*

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.

> **Parameters**
>
> - **code** (`str`) – a preformatted code string
>
> - **lang** (`str`) – the language for syntax highlighting
>
> **Returns** the Paragraph added to this Document

add_element(*element:* snakemd.generator.Element) → *snakemd.generator.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(Header(InlineText("Python is Cool!"), 2))
```

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

> **Parameters** `element` (`Element`) – a markdown object (e.g., Table, Header, etc.)
>
> **Returns** the Element added to this Document

add_header(*text: str*, *level: int = 1*) → *snakemd.generator.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.

  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_horizontal_rule() → *snakemd.generator.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]*) → *snakemd.generator.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*) → *snakemd.generator.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*) → *snakemd.generator.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_table(*header: Iterable[str]*, *data: Iterable[Iterable[str]]*, *align:*
   *Optional[Iterable[*snakemd.generator.Table.Align*]] = None*) → *snakemd.generator.Table*

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

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

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

> **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
>
> **Returns** the Table added to this Document

add_table_of_contents(*levels: range = range(2, 3)*) → *snakemd.generator.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 header elements 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 header levels to be included in the table of contents
>
> **Returns** the TableOfContents added to this Document

add_unordered_list(*items: Iterable[str]*) → *snakemd.generator.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.

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

Generates the markdown file. Assumes UTF-8 encoding.

> **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 elements.

> **Returns** the document as a markdown string

scramble() → None

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

class snakemd.generator.**Element**

Bases: `object`

An element is defined as a standalone section of a markdown file. All elements are to be surrounded by empty lines. Examples of elements include paragraphs, headers, tables, and lists.

render() → str

Renders the element as a markdown string. This function is called by __str__ for all classes which inherit Element.

> **Raises** `NotImplementedError` – interface method never to be implemented

> **Returns** the element as a markdown string

verify() → *snakemd.generator.Verification*

Verifies that the element is valid markdown.

> **Raises** `NotImplementedError` – interface method never to be implemented

> **Returns** a verification object from the violator

class snakemd.generator.**Header**(*text: Union[*snakemd.generator.InlineText*, str], level: int*)

Bases: `snakemd.generator.Element`

A header is a text element which serves as the title for a new section of a document. Headers come in six main sizes which correspond to the six headers sizes in HTML (e.g., <h1>).

> **Parameters**
>
> - **text** (`Union[InlineText, str]`) – the header text
> - **level** (`int`) – the header level between 1 and 6 (rounds to closest bound if out of range)

demote() → None

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

promote() → None

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

render() → str

Renders the header in markdown according to the level provided.

> **Returns** the header as a markdown string

verify() → *snakemd.generator.Verification*

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

New in version 0.2.0.

> **Returns** a verification object from the violator

class snakemd.generator.**HorizontalRule**

   Bases: *snakemd.generator.Element*

   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.

   render() → str

      Renders the horizontal rule using the three dash syntax.

      New in version 0.2.0.

         Returns the horizontal rule as a markdown string

   verify() → *snakemd.generator.Verification*

      Verifies the structure of the HorizontalRule element. 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

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

   Bases: object

   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.

      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*)

         • **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() → *snakemd.generator.InlineText*

      Adds bold styling to self.

      Changed in version 0.7.0: Modified to return previous bold state

         Returns self

   code() → *snakemd.generator.InlineText*

      Adds code style to self.

      New in version 0.7.0.

         Returns self

is_text() → bool

Checks if this InlineText 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 InlineText object represents a URL.

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

italicize() → *snakemd.generator.InlineText*

Adds italics styling to self.

New in version 0.7.0.

> **Returns** self

link(*url: str*) → *snakemd.generator.InlineText*

Adds URL to self.

New in version 0.7.0.

> **Parameters url** (`str`) – the URL to apply to this text 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.

> **Returns** the InlineText object as a string

reset() → *snakemd.generator.InlineText*

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

unbold() → *snakemd.generator.InlineText*

Removes bold styling from self.

Changed in version 0.7.0: Modified to return previous bold state

> **Returns** self

uncode() → *snakemd.generator.InlineText*

Removes code style from self.

New in version 0.7.0.

> **Returns** self

unitalicize() → *snakemd.generator.InlineText*

Removes italics styling from self.

New in version 0.7.0.

> **Returns** self

unlink() → *snakemd.generator.InlineText*

>   Removes URL from self.

>   New in version 0.7.0.

> >   **Returns** self

verify() → *snakemd.generator.Verification*

>   Verifies that the InlineText 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

class snakemd.generator.**MDCheckList**(*items: Iterable[Union[str,* snakemd.generator.InlineText, snakemd.generator.Paragraph, snakemd.generator.MDList*]], checked: bool = False*)

Bases: `snakemd.generator.MDList`

A markdown CheckBox list has boxes that can be clicked.

New in version 0.10.0.

>   **Parameters**

>   - **items** (`Iterable[Union[str, InlineText, Paragraph, MDList]]`) – a "list" of objects to be rendered as a Checkbox list

>   - **checked** (`bool`) – the state of the checkbox; set to True to render a checked box (i.e., True -> - [x] item)

render() → str

>   Renders the markdown Check Box list according to the settings provided. For example, if the the checked flag is set, a checked list will be rendered in markdown.

> >   **Returns** the list as a markdown string

class snakemd.generator.**MDList**(*items: Iterable[Union[str,* snakemd.generator.InlineText, snakemd.generator.Paragraph, snakemd.generator.MDList*]], ordered: bool = False*)

Bases: `snakemd.generator.Element`

A markdown list is a standalone list that comes in two varieties: ordered and unordered.

Changed in version 0.4.0: Expanded constructor to accept strings directly

>   **Parameters**

>   - **items** (`Iterable[Union[str, InlineText, 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)

render() → str

>   Renders the markdown list according to the settings provided. For example, if the the ordered flag is set, an ordered list will be rendered in markdown.

> >   **Returns** the list as a markdown string

verify() → *snakemd.generator.Verification*

>   Verifies that the markdown list is valid. Mainly, this checks the validity of the containing InlineText 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

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

>   Bases: `snakemd.generator.Element`
>
>   A paragraph is a standalone element 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[Union[InlineText, 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` ``)
>   >
>   > - **lang** (`str`) – the language of the code snippet; invalid without the code flag set to True
>   >
>   > - **quote** (`bool`) – the quote state of the paragraph; set True to convert the paragraph to a blockquote (i.e., True -> > quote)

>   **add**(*text: Union[snakemd.generator.InlineText, str]*) → None
>
>   >   Adds a text object to the paragraph.
>   >
>   >   Changed in version 0.4.0: Allows adding of strings directly
>   >
>   >   > **Parameters** `text` – a custom text element

>   **insert_link**(*target: str*, *url: str*, *count: int = - 1*) → *snakemd.generator.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 element. If not, it must be a quote or code block.
>   >
>   >   New in version 0.3.0.

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

render() → str

> Renders the paragraph as markdown according to the settings provided. For example, if the code flag is enabled, the paragraph will be rendered as a code block. If both flags are enabled, code takes precedence.
>
> Changed in version 0.4.0: No longer assumes spaces between InlineText items
>
> > **Returns** the paragraph as a markdown string

replace(*target: str*, *replacement: str*, *count: int = - 1*) → *snakemd.generator.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 InlineText 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*) → *snakemd.generator.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() → *snakemd.generator.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

class snakemd.generator.**Table**(*header: Iterable[Union[str,* InlineText*,* Paragraph*]]*, *body: Iterable[Iterable[Union[str,* InlineText*,* Paragraph*]]]*, *align: Iterable[*Align*] = None*)

> Bases: `snakemd.generator.Element`

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

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

> **Parameters**
>
> - **header** – the header row of labels
>
> - **body** – the collection of rows of data
>
> - **align** – the column alignment

class Align(*value*)

> Bases: `enum.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**

render() → str

> Renders a markdown table from a header "list" and a data set.
>
> Changed in version 0.4.0: Modified to support column alignment and pipes on both sides of the table
>
> > **Returns** a table as a markdown string

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. InlineText elements may also be malformed.
>
> New in version 0.2.0.
>
> > **Returns** a verification object from the violator

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

> Bases: `snakemd.generator.Element`
>
> A Table of Contents is an element containing an ordered list of all the *<h2>* headers in the document by default. A range can be specified to customize which headers (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 header levels to include in the table of contents; defaults to range(2, 3)

render() → str

> Renders the table of contents using the Document reference.
>
> > **Returns** the table of contents as a markdown string

verify() → *snakemd.generator.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

class snakemd.generator.**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:* snakemd.generator.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

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:

- **v0.10.1**

    – Enforced UTF-8 encoding in the output_page method

- **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

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

S

T

U

V