# Flask-Misaka Documentation

*Release 1.0.0*

**David Baumgold**

**Jan 17, 2019**

# Contents

Flask-Misaka provides a pleasant interface between the Flask web framework and the Misaka Markdown parser.[1]

[1] (Technically, Misaka is just a Python binding to the Hoedown library, which is written in C.)

CHAPTER 1

# Installation

Install the extension with:

```
$ pip install Flask-Misaka
```

CHAPTER 2

# Usage

Just import the *markdown()* function and use it!

```
>>> from flask.ext.misaka import markdown
>>> markdown("A *simple* example.")
Markup(u'<p>A <em>simple</em> example.</p>\n')
```

To use Markdown in your templates, you just need to import the *Misaka* class and wrap your Flask instance with it:

```
from flask import Flask
from flask.ext.misaka import Misaka

app = Flask(__name__)
Misaka(app)
```

or use the application factory pattern:

```
md = Misaka()
app = Flask(__name__)
md.init_app(app)
```

And then the `markdown` filter will be available in your Jinja2 templates. You can pass variables in your template through it:

```
{{ text|markdown }}
```

Or, you can use the `filter` tag to write your template directly in Markdown and have Jinja dynamically interpret it for you!

```
{% filter markdown %}
I'm writing my templates in *Markdown!*
{% endfilter %}
```

# CHAPTER 3

# API

flask_misaka.**markdown**(*text*, *renderer=None*, ***options*)

Parses the provided Markdown-formatted text into valid HTML, and returns it as a `flask.Markup` instance.

**Parameters**

- **text** – Markdown-formatted text to be rendered into HTML
- **renderer** – A custom misaka renderer to be used instead of the default one
- **options** – Additional options for customizing the default renderer

**Returns** A `flask.Markup` instance representing the rendered text

**class** flask_misaka.**Misaka**(*app=None*, *renderer=None*, ***defaults*)

**__init__**(*app=None*, *renderer=None*, ***defaults*)

Set the default options for the *render()* method. If you want the `markdown` template filter to use options, set them here.

A custom misaka renderer can be specified to be used instead of the default one.

**init_app**(*app*)

Registers the rendering method as template filter.

**Parameters** **app** – a `flask.Flask` instance.

**render**(*text*, ***overrides*)

It delegates to the *markdown()* function, passing any default options or renderer set in the *__init__()* method.

The `markdown` template filter calls this method.

**Parameters**

- **text** – Markdown-formatted text to be rendered to HTML
- **overrides** – Additional options which may override the defaults

**Returns** A `flask.Markup` instance representing the rendered text

CHAPTER 4

# Options

Misaka is very customizable, and supports many Markdown extensions. Flask-Misaka provides a nicer API for these extensions. All functions in the public API (except `Misaka.init_app()`) accept the following boolean arguments, all of which default to False:

Table 1: Flask-Misaka options

| Option Name | Description |
|---|---|
| `autolink` | Parse links even when they are not enclosed in <> characters. Autolinks for the http, https and ftp protocols will be automatically detected. Email addresses are also handled, and http links without protocol, but starting with `www`. |
| `fenced_code` | Blocks delimited with 3 or more ~ or backticks will be considered as code, without the need to be indented. An optional language name may be added at the end of the opening fence for the code block. |
| `underline` | Treat text surrounded by underscores (like `_this_`) as underlined, rather than emphasized. |
| `highlight` | Treat text surrounded by double equal signs (like `==this==`) as highlighted. |
| `quote` | Parse inline quotes (like `"this"`). This allows the renderer to control how they are rendered. |
| `math` | Parse inline LaTeX-style math blocks (like `$$this$$`). |
| `math_explicit` | Parse inline LaTeX-style math blocks with a single dollar, e.g. `$x + y = 3$` |
| `disable_indented_code` or `no_indented_code` | Ignore indented code blocks |
| `no_intra_emphasis` | Do not parse emphasis inside of words. Strings such as `foo_bar_baz` will not generate  tags. |
| `space_headers` | A space is always required between the hash at the beginning of a header and its name, e.g. `#this is my header` would not be a valid header. |
| `strikethrough` | Two ~ characters mark the start of a strikethrough, e.g. `this is ~~good~~ bad`. |
| `superscript` | Parse superscripts after the ^ character; contiguous superscripts are nested together, and complex values can be enclosed in parenthesis, e.g. `this is the 2^(nd) time`. |
| `tables` | Parse PHP-Markdown tables. |
| `hard_wrap` *or* `wrap` | Insert HTML <br> tags inside on paragraphs where the origin Markdown document had newlines (by default, Markdown ignores these newlines). |
| `footnotes` | Parse Markdown footnotes. |
| `escape` | Escape all HTML tags, regardless of what they are. |
| `skip_html` *or* `no_html` | Do not allow any user-inputted HTML in the output. |
| `use_xhtml` *or* `xhtml` | Output XHTML-conformant tags. |
| `smartypants` | Post-process rendered markdown text with SmartyPants. |

Any option that starts with `no_` can also be passed as its inverse set to False. For example, `no_html=True` and `html=False` have exactly the same effect, just as `no_intra_emphasis=True` and `intra_emphasis=False` have exactly the same effect.

**Note:** To override an option, you must use exactly the same option name as you used to originally set the option. If you set `html=False` as a default, you must override it with `html=True`: using `no_html=False` or `skip_html=False` will not work, even though they all refer to the same thing.

# Python Module Index

## f

# Index

## Symbols



## F



## I



## M



## R