Markdown syntax for writing documents

Install markdown extensions, use new markdown syntax, create better layout, write content faster.

#markdown

Last update: April 23, 2021

Table of Content

For basic markdown syntax, refer to Markdown Guide.

1. Meta-data

The Meta-Data extension adds a syntax for defining meta-data about a document. It is inspired by and follows the syntax of MultiMarkdown.

Enable the extension:

```
markdown_extensions:
    - meta
```

Meta-data consists of a series of keywords and values defined at the beginning of a markdown document like this:

```
title: The page title
description: The summary of the page content
tags:
    page
    markdown
```

Alternatively, YAML style, using triple-dash `---`, can be used to mark the start and the end of the meta-data section:

```
---
title: The page title
description: The summary of the page content
tags:
    - page
    - markdown
---
```

The metadata can be used in the template and page content. In jinja syntax, each page is represented as a *page* object, then `Markdown syntax` will be replaced by the page's title.

Read more about using metadata in page content, and customize the theme.

2. Admonition

The Admonition extension adds rST-style admonitions to Markdown documents.

Enable the extension:

```
markdown_extensions:
    - admonition
```

Admonitions are created using the following syntax:

```
!!! type "Title"
    Content of the admonition is indented.
```

Title

Content of the admonition is indented.

Using custom class and style to make admonition which does not have title but still have icon:

```
!!! type notitle "&nbsp;"
    Content of the admonition is indented.
```

Content of the admonition is indented.

Other *types*:

note, seealso

Content of the admonition is indented.

abstract, summary, tldr

Content of the admonition is indented.

info, todo

Content of the admonition is indented.

tip, hint, important

Content of the admonition is indented.

success, check, done

Content of the admonition is indented.

question, help, faq

Content of the admonition is indented.

warning, caution, attention

Content of the admonition is indented.

failure, fail, missing

Content of the admonition is indented.

danger, error

Content of the admonition is indented.

bug

Content of the admonition is indented.

example

Content of the admonition is indented.

quote, cite

Content of the admonition is indented.

3. Attribute

The Attribute Lists extension adds a syntax to define attributes on the various HTML elements in markdown's output.

Enable the extension:

```
markdown_extensions:
    - attr_list
```

An example attribute list might look like this:

```
{ #someid .someclass somekey='somevalue' }
```

Block attribute:

To define attributes for a block level element, the attribute list should be defined on the last line of the block by itself.

```
Lorem Ipsum is simply dummy text of the printing and typesetting industry.
Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,
when an unknown printer took a galley of type and scrambled it to make a type
specimen book.
{style="font-style:italic;"}
```

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

Element attribute:

To define attributes on inline elements, the attribute list should be defined immediately after the inline element generated by markdown with no white space.

```
This is a _green_{style="color:green"} word.
This is a <span>green</span>{style="color:green"} word.
```

This is a *green* word. This is a non-green{style="color:green"} word.

4. Lists

Markdown supports Ordered and Unordered lists. Extensions provide 2 more kinds of list as Tasks and Definitions. The Sane Lists extension alters the behavior of the Markdown List syntax to be less surprising. Sane Lists do not allow the mixing of list types. In other words, an ordered list will not continue when an unordered list item is encountered and vice versa.

Enable the extension:

```
markdown_extensions:
    - pymdownx.tasklist
    - def_list
    - sane_lists
```

Ordered list:

```
1. Ordered item 1
    1. Child 1
    2. Child 2
2. Ordered item 2
```

1. Ordered item 1
    a. Child 1
    b. Child 2
2. Ordered item 2

Unordered list:

```
* Unordered item 1
    * Child 1
    * Child 2
* Unordered item 2
```

- Unordered item 1
    - Child 1
    - Child 2
- Unordered item 2

Sane list:

```
1. Ordered item 1
2. Ordered item 2

* Unordered item 1
* Unordered item 2
```

1. Ordered item 1
2. Ordered item 2

- Unordered item 1
- Unordered item 2

Tasks:

```
- [x] item 1
    * [x] item A
    * [ ] item B
- [ ] item 2
```

- ✓ item 1
    - ✓ item A
    - ☐ item B
- ☐ item 2

Definitions:

```
Roses
:   are red

Violets
:   are blue
```

Roses

: are red

Violets

: are blue

5. Code blocks

The SuperFences extension provides a number of features including allowing the nesting of fences, and ability to specify custom fences to provide features like flowcharts, sequence diagrams, or other custom blocks.

Highlighting can be further controlled via the Highlight extension.

Enable the extension:

```
markdown_extensions:
    - pymdownx.superfences
    - pymdownx.highlight
```

The format is as below:

~~~
```{ .language  #id .class key="value" linenums="n" hl_lines="x y-z"}
codeblock content
```
~~~

or in simpler syntax which does not support *id*, *class* or custom *key=value*:

~~~
``` language  linenums="n" hl_lines="x y-z"
codeblock content
```
~~~

Option `linenums="n"` creates line numbers starting from *n*.

Option `hl_lines="x y-z"` highlights the *x-th* line and lines in the range from *y* to *z*. Line numbers are always referenced starting at 1 ignoring what the line number is labeled by the option `linenums="n"`.

~~~
``` c linenums="2" hl_lines="1 4-5"
#include <stdio.h>

int main(void) {
    printf("Hello world!\n");
    return 0;
}
```
~~~

```
2  #include <stdio.h>
3
4  int main(void) {
5      printf("Hello world!\n");
6      return 0;
7  }
```

6. Inline code

the InlineHilite extension is an inline code highlighter inspired by CodeHilite.

Enable the extension:

```
markdown_extensions:
    - pymdownx.inlinehilite
```

Borrowing from CodeHilite's existing syntax, InlineHilite utilizes the following syntax to insert inline highlighted code:

```
`:::language mycode`
```

or

```
`#!language mycode`
```

This will render as below example:

```
A line of python code: `#!python [x for x in range(1, 10) if x % 2]`.
```

A line of python code: `[x for x in range(1, 10) if x % 2]`.

7. Images

There are some extensions to add a caption to an image. After testing, markdown-captions is a good one that uses alt text to make caption, accepts markdown in alt text.

```
![Caption](link to image)
```

Some images have big size that does not show the detail, therefore, it's better to zoom in by clicking on them, and pan the image on the screen. The view-bigimg library do that requirement well. Enable it by following Customize theme.

For example:

```
![Lorem Picsum](https://picsum.photos/1280/720)
```

Lorem Picsum

8. Tabs

Tabbed extension provides a syntax to easily add tabbed Markdown content.

Enable the extension:

```
markdown_extensions:
    - pymdownx.tabbed
```

Tabs start with `===` to signify a tab followed by a quoted title. Consecutive tabs are grouped into a tab set.

```
=== "Tab 1"
    Some texts

=== "Tab 2"
    Some other texts

    Inner tabs:
    === "Tab A"
        Text A

    === "Tab B"
        Text B
```

Tab 1

Some texts

Tab 2

Some other texts

Inner tabs:

Tab A

Text A

Tab B

Text B

9. Tables

The Tables extension adds the ability to create tables in Markdown documents.

Enable the extension:

```
markdown_extensions:
    - tables
```

Markdown Tables are written in pipe-line format: row is on one line, cell is inline text only. The 1st line contains the column headers. The 2nd line is to control text alignment in a column: `:---`, `:---:` and `---:` are left, center, and right alignment.

```
| Syntax     | Description  |   Test Text |
| :--------- | :----------: | ----------: |
| Left align | Center align | Right align |
| Some texts |  Some texts  |  Some texts |
```

Syntax	Description	Test Text
Left align	Center align	Right align
Some texts	Some texts	Some texts

10. Formatting

Carets:

Caret optionally adds two different features which are syntactically built around the `^` character.

Enable the extension:

```
markdown_extensions:
    - pymdownx.caret
```

The double carets `^^` inserts `<ins></ins>` tags, and the single caret `^` inserts `<sup></sup>` tags.

```
^^Insert me^^
H^2^0
```

Insert me
$H^2 0$

Marks:

Mark adds the ability to insert `<mark></mark>` tags.

Enable the extension:

```
markdown_extensions:
    - pymdownx.mark
```

The syntax requires the text to be surrounded by double equal signs `==`.

```
==mark me==
==smart==mark==
```

mark me
smart==mark

Tildes:

The PyMdown extension optionally adds two different features which are syntactically built around the `~` character: delete using double tildes `~~` which inserts `<del></del>` tags and subscript using single tilde `~` which inserts `<sub></sub>` tags.

Enable the extension:

```
markdown_extensions:
    - pymdownx.tilde
```

For example:

```
~~Delete me~~
CH~3~CH~2~OH
```

~~Delete me~~
CH_3CH_2OH

11. Critic

Critic is an extension that adds handling and support of Critic Markup which uses a special syntax to represent edits to a Markdown document. This extension runs before all other extensions to parse the critic edits.

Enable the extension:

```
markdown_extensions:
    - pymdownx.critic
```

Critic Markup uses special markup to insert, delete, substitute, highlight, and comment.

```
To insert or remove text you can use {++insert me++} and {--remove me--}
respectively. You can also denote a substitution with {~~substitute this~>with
this~~}.

You can also highlight specific text with {==highlight me==}. Or even comment,
which is generally done by highlighting text and following it with a comment: {
==highlight me==}{>>Add a comment<<}.
```

To insert or remove text you can use insert me and ~~remove me~~ respectively. You can also denote a substitution with ~~substitute this~~with this.

You can also highlight specific text with highlight me. Or even comment, which is generally done by highlighting text and following it with a comment: highlight me/* Add a comment */.

12. HTML block

The Markdown in HTML extension that parses Markdown inside of HTML block tags.

Enable the extension:

```
markdown_extensions:
    - md_in_html
```

By default, Markdown ignores any content within a raw HTML block-level element. With the *md-in-html* extension enabled, the content of a raw HTML block-level element can be parsed as Markdown by including a markdown attribute on the opening tag.

```
<div>
This is __not parsed__ by Markdown.
</div>
<div markdown="1">
This is a __bold__ word parsed by Markdown.
</div>
```

This is __not parsed__ by Markdown.

This is a **bold** word parsed by Markdown.

13. Icons & Emojis

The Emoji extension

Enable the extension:

```
markdown_extensions:
    - pymdownx.emoji:
        emoji_index: !!python/name:materialx.emoji.twemoji
        emoji_generator: !!python/name:materialx.emoji.to_svg
```

Emojis:

> Emojis can be written by putting the shortcode of the emoji between two colons. Look up the shortcodes at Emojipedia.

`:smile:` 😄 `:heart:` ❤️

Icons:

Icons can be used similarly to emojis, by referencing a valid path to any icon bundled with the theme, which are located in the *.icons* directory, and replacing `/` with `-`:

```
- :material-account-circle: – `.icons/material/account-circle.svg`
- :fontawesome-regular-laugh-wink: – `.icons/fontawesome/regular/laugh-
wink.svg`
- :octicons-octoface-24: – `.icons/octicons/octoface-24.svg`
```

👤 😉 🐱

Include in theme:

Using `include` function of Jinja to add an icon wrapped in a `twemoji` class:

```
<span class="twemoji">
{% include ".icons/fontawesome/brands/twitter.svg" %}
</span>
```

14. Custom block

The Custom Blocks extension defines a common markup to create parametrizable and nestable custom blocks.

Install the extension:

```
pip install -U markdown-customblocks
```

Enable the extension:

```
markdown_extensions:
    - customblocks
```

This extension parses markup structures like this one:

```
::: mytype "value 1" param2=value2
    Indented content
```

Example usage: Add the filename of a codeblock, to show where the it belongs.

Syntax:

~~~
::: file
    main.c

``` cpp
int main(void) {
    return 0;
}
```
~~~

with style:

```
.md-typeset .file > p {
    font-size: .9em;
    font-style: italic;
    color: darkred;
    margin-bottom: -1.2em;
}
```

will generate:

main.c

```
int main(void) {
    return 0;
}
```

15. Footnotes

The Footnotes extension adds syntax for defining footnotes in Markdown documents.

Enable the extension:

```
markdown_extensions:
    - footnotes
```

And use the following syntax:

```
Footnotes[^1] have a label[^fn] and the footnote's content.

[^1]: This is a footnote content.
[^fn]: A footnote on the label _fn_.
```

to get:

Footnotes[1] have a label[2] and the footnote's content.

Footnotes:

1. This is a footnote content. ↩
2. A footnote on the label *fn*. ↩