

# Markdown syntax for writing documents

Markdown comes with a simple syntax to create headers, links, images, and paragraphs with formatted text, lists. However, that's not enough. Some Markdown extensions bring more syntaxes to create complex elements or new layouts that help writing documents easier, faster, and look professional.

[#markdown](#)

---

Last update: 2021-06-03 16:34:50

# Table of Content

1. Meta-data
2. Code blocks
3. Inline code
4. Admonition
5. Attribute list
  - 5.1. Block attribute
  - 5.2. Element attribute
6. Lists
  - 6.1. Ordered list
  - 6.2. Unordered list
  - 6.3. Task list
  - 6.4. Definition
  - 6.5. Sane list
7. Images
8. Tabs
9. Tables
10. Formatting
  - 10.1. Caret
  - 10.2. Mark
  - 10.3. Tildes
  - 10.4. Critic
11. HTML block
12. Icons & Emojis
  - 12.1. Emojis
  - 12.2. Icons
  - 12.3. Use in template
13. Escape All
14. Sane Headers

## 15. Special characters

### 15.1. Smarty Pants

### 15.2. Smart Symbols

## 16. Snippets

## 17. Custom block


## 18. Footnotes

## 19. Tips

### 19.1. Small headers

### 19.2. Ignore formatting

### 19.3. Look up an icon or emoji

 For basic markdown syntax, refer to [Markdown Guide](#).

## 1. Meta-data

The [Meta-Data extension](#) adds a syntax for defining meta-data of a document. It is inspired by and follows the syntax of [MultiMarkdown](#). Meta-data is the additional information that can be used to briefly describe the content of a post, such as the title, the short description, tags, and sometimes the banner image.

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
```

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

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

The metadata can be used in the template and the page content<sup>1</sup>. In jinja syntax, each page is represented as a *page* object, then the meta-data field `{{ page.meta.title }}` will be replaced by the string `The page title`.

MkDocs uses the title of each post to show in the navigation sidebar. In case the title is long, it can make the side look a bit messy. A solution is to use a short title on navigation, and a long title on the post. Read more in [Customize theme](#).

## 2. 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 standard format which supports to add *id*, *class* or custom *key=value* is as below:

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

or in a simple syntax:

```
```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-th* to *z-th*. Line numbers are always referenced starting at 1 ignoring what the line number is started labeling at the number set by the option `linenums="n"`.

Example:

```
```cpp 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  }
```

Some styles will be added to the code blocks to make it look a bit bigger and wrap long lines. Read more in [Customize theme](#).

### 3. 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 my code`` or ``#!language my code``.

This will render this line ``#!python [x for x in range(1, 10) if x % 2]`` to a fully colored inline Python code: `[x for x in range(1, 10) if x % 2]`.

### 4. Admonition

The [Admonition extension](#) adds [rST-style admonitions](#) to Markdown documents. This block displays its content in a block, with or without a title.

Enable the extension:

```
markdown_extensions:
    - admonition
```

Admonitions are created using the following syntax. The title is optional, use an empty string to remove the title:

```
!!! type "Title"

    Content of the admonition is indented

!!! type ""

    There is no title
```

 **Title**

Content of the admonition is indented

There is no title

## 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

With some additional styles, it is possible to create an admonition without title but still has icon, by using `.nt` class and an empty title (use `"&nbsp;"`, or `"\ "` if [Escape All](#) extension is used). Read more in [Customize theme](#).

```
!!! info nt "&nbsp;"
```

This admonition has icon as an inline element with the content



This admonition has icon as an inline element with the content

## 5. Attribute list

The [Attribute Lists extension](#) adds a syntax to add attributes on the various HTML elements in markdown's output.

Enable the extension:

```
markdown_extensions:
  - attr_list
```

An example of using attribute list might look like this:

```
{ #id .class key='value' }
```

## 5.1. 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.

```
This is a paragraph colored in teal by using a block attribute.
{style="color: teal;"}
```

This is a paragraph colored in **teal** by using a block attribute.

## 5.2. Element attribute

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

```
This is a _green_{style="color:green"} word.\
This is a <span>non-markdown span</span>{style="color:green"} word therefore the
attribute is not parsed.
```

This is a **green** word.

This is a non-markdown span{style="color:green"} word therefore the attribute is not parsed.

## 6. 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:
      custom_checkbox: true
  - def_list
  - sane_lists
```



Some styles need to be added to show lists in a better view. Read more in [Customize theme](#).

## 6.1. 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

## 6.2. Unordered list

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

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

## 6.3. Task list

```
- [x] item 1
  - [x] item a
  - [ ] item b
- [ ] item 2
```

- ✓ item 1
  - ✓ item a
  - ⊖ item b
- ⊖ item 2

## 6.4. Definition

```
Roses
: are red

Violets
: are blue
```

Roses  
are red

Violets  
are blue

## 6.5. 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

## 7. Images

There are some extensions to add a caption to an image. After testing, [markdown-captions](#) is a good one that uses the alternate text to make caption, accepts markdown in the alternate text.

Install the extension:

```
pip install -U markdown-captions
```

then enable it in the config file:

```
markdown_extensions:  
    - markdown_captions
```

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 can do that requirement well. Read more in [Customize theme](#) to learn how to enable that library.

```
![A photo from <https://picsum.photos>](https://picsum.photos/320/240)
```



*A photo from <https://picsum.photos>*

## 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 A"
```

```
Text A
```

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

```
Text B
```

```
=== "Tab 2"
```

```
Some other texts
```

**Tab 1**

Some texts

**Tab A**

Text A

**Tab B**

Text B

**Tab 2**

Some other texts

Styles for tabs need to change a little bit to have left margin in tab's content. Read more in [Customize theme](#).

## 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 1<sup>st</sup> line contains the column headers. The 2<sup>nd</sup> line is to control text alignment in a column: `:--`, `---`, `:---` and `---` are left, center, and right alignment. Styles for table need to change a little bit to show cell border. Read more in [Customize theme](#).

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

Syntax

Description

Test Text

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

## 10. Formatting

Some texts may have special formats, and here are extensions that can help to add some styles.

### 10.1. Caret

**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^^ the energy E=mc^2^
```

Insert the energy  $E=mc^2$

### 10.2. Mark

**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== and ==mark==me==all==
```

mark me and mark==me==all

### 10.3. Tildes

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

Enable the extension:

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

For example:

```
~~Delete~~ the existence of
CH~3~CH~2~OH
```

Delete the existence of CH<sub>3</sub>CH<sub>2</sub>OH

## 10.4. 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, use {++insert me++} and {--remove me--}.\
Denote a substitution with {~~substitute this~>with this~~}.
```

```
Highlight specific text with {==highlight me==}.\
Or even add {>>a comment<<}.
```

To insert or remove text, use insert me and ~~remove me~~.

Denote a substitution with ~~substitute this~~with this.

Highlight specific text with highlight me.

Or even add /\* a comment \*/.

## 11. 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 `this` 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 word__ by Markdown.
</div>
<div markdown="1">
This is a __bold__ word parsed by Markdown.
</div>
```

This is \_\_not parsed word\_\_ by Markdown.

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

## 12. Icons & Emojis

The [Emoji extension](#) adds support for inserting emoji via simple short names enclosed within colons `:short_name:`. This is accomplished by using a short name index to map easy-to-remember names to the corresponding emoji characters.

Enable the extension:

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

### 12.1. Emojis

Emojis can be written by putting the short-code of the emoji between two colons. Look up the short-codes at [Emojipedia](#).

`:smile:` 😊 `:heart:` ❤️

### 12.2. 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 `-`.

For example:

The short-code `:material-account-circle:` will be converted to an svg image element with the path `.icons/material/account-circle.svg` which eventually shows the icon 👤 on the webpage.

## 12.3. Use in template

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

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

to show the Twitter icon .

## 13. Escape All

The `Escape All` extension makes the backslash `\` character escape everything after it, except things in code blocks of any kind.

Enable the extension:

```
markdown_extensions:
  - pymdownx.escapeall:
      hardbreak: true
      nbsp: true
```

There are two special escapes among all of these escapes though: escaping *space* characters and escaping *newline* characters:

- Enable `nbsp` to convert an escaped space into a non-breaking space: `&nbsp;` .
- Enable `hardbreak` to convert an escaped newline to a hard break `<br>` . The advantage of hard break is that the backslash is visually seen in the document, opposed to the Markdown's default method of two spaces at the end of a line.

For example:

```
This is a line with one space.\
This is a new line with some \ \ \ \ spaces.
```

This is a line with one space.

This is a new line with some    spaces.

## 14. Sane Headers

The `Sane Headers` is an extension that alters the default hashed headers extension to require headers to have spaces after the hashes `#` in order to be recognized as headers. This allows for other extension syntaxes to use `#` in their syntaxes as long as no spaces follow the `#` at the beginning of a line.



Enable the extension:

```
markdown_extensions:
    - pymdownx.saneheaders
```

In default, both of these are treated as headers:

```
## Header

##Also a Header
```

With SaneHeaders, only the first is a header:

```
## Header

##Not a Header
```

## 15. Special characters

The [Smarty Pants](#) converts ASCII dashes, quotes and ellipses to their HTML entity equivalents.

The [Smart Symbols](#) adds syntax for creating special characters such as trademarks, arrows, fractions, etc.

Enable the extension:

```
markdown_extensions:
    - smarty:
        smart_angled_quotes: true
    - pymdownx.smartsymbols
```

### 15.1. Smarty Pants

```
'single quote'
"double quote"
<<angle quote>>
ellipses ...
N-dash --
M-dash ---
```

- ‘single quote’
- “double quote”
- «angle quote»
- ellipses ...
- N-dash –
- M-dash —


### 15.2. Smart Symbols

```
trademark (tm)
copyright (c)
registered (r)
in care of c/o
plus or minus +/-
arrows --> <-- <-->
not equal !=
fractions 1/4
ordinal numbers 1st 2nd 3rd 4th 5th
```

- trademark ™
- copyright ©
- registered ®
- in care of %
- plus or minus ±
- arrows → ← ↔
- not equal ≠
- fractions ¼ ½ ¾
- ordinal numbers 1<sup>st</sup> 2<sup>nd</sup> 3<sup>rd</sup> 4<sup>th</sup> 5<sup>th</sup>

## 16. Snippets

The [Snippets](#) inserts the content of a file into the markdown document. It is great for situations where there is a content that needs to be inserted into multiple documents.

 Snippets is run as a preprocessor, so if a snippet is found in a fenced code block etc., it will still get processed.

Enable the extension:

```
markdown_extensions:
  - pymdownx.snippets
```

There are two modes of inserting snippets: single line and block. Single line mode accepts a single file name, and block accepts multiple files.

Single line format is done by placing the following markup for the single line notation:

```
--8<-- "filename.ext"
```

In block format, it is important to note that empty lines are preserved for formatting.

```
--8<--
filename.md
the below empty line is preserved.log

filename.log
--8<--
```

To temporarily ignore a file, comment it out by pre-pending the path with semicolon ; and a space. This works for both single line and block format:

```
--8<-- "; skip.md"
--8<--
include.md
; skip.md
--8<--
```

## 17. Custom block

The [Custom Blocks extension](#) defines a common markup to create parameter-supported 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:

```
::: type class param=value

    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: 0.9em;
  color: gray;
```

```
margin-bottom: -1.2em;
}
```

will generate:

main.c

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

There are some more custom block created with new layout and style, such as `row` and `col`. Read more in [Customize theme](#).

## 18. 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`.
```

Footnotes<sup>2</sup> have a label<sup>3</sup> and the footnote's content.

## 19. Tips

There are some tips when writing document in Markdown which help to format the content in a good layout.

### 19.1. Small headers

At small header levels `<h5>` and `<h6>`, the header text is smaller than the body text, and the header is transformed to all capitalized characters. Use a hard break and a bold text instead. This make text clear to be read, and have a good space to the previous paragraph.

```
\
**Item**
```

## 19.2. Ignore formatting

When using [Prettier extension](#) to format the documents, some block can be ignored from formatting by adding directives. Refer to the [Prettier - Ignore Code](#). Note the this extension only format the markdown file, and when it is rendered to HTML, it will be displayed in HTML rendered visual.

In Markdown, use block directive before a block that needs to be preserved:

```
<!-- prettier-ignore -->
Do not format this
```

In case of a big block or multiple blocks, use the range directive:

```
<!-- prettier-ignore-start -->
MY	AWESOME	AUTO-GENERATED	TABLE
a	b	c	d
<!-- prettier-ignore-end -->
```

## 19.3. Look up an icon or emoji

Material for MkDocs provides a tool to look up an icon or an emoji by searching a name. It has fast copy to clipboard when selecting on the wanted icon. Please go there <https://squidfunk.github.io/mkdocs-material/reference/icons-emojis/>.

- 
1. Use [mkdocs-macros](#) plugin to use Jinja template directly in the Markdown content. ↩
  2. This is a footnote content. ↩
  3. A footnote on the label `fn.` ↩