

# cmarker.typ

<https://github.com/SabrinaJewson/cmarker.typ>

This package enables you to write CommonMark Markdown, and import it directly into Typst.

simple.typ	simple.md
<pre>#import "@preview/cmarker:0.1.2"  #cmarker.render(read("simple.md"))</pre>	<pre># We can write Markdown!  *Using* __lots__ ~of~ `fancy` [features](https://example.org/).</pre>
<div>simple.pdf</div> <div><p><b>We can write Markdown!</b></p><p><i>Using lots of fancy features.</i></p></div>	

This document is available in [Markdown](#) and [rendered PDF](#) formats.

## 1. API

We offer a single function:

```
render(
  markdown,
  smart-punctuation: true,
  blockquote: none,
  math: none,
  h1-level: 1,
  raw-typst: true,
  html: (:),
  scope: (:),
  show-source: false,
) -> content
```

The parameters are as follows:

- `markdown`: The [CommonMark](#) Markdown string to be processed. Parsed with the [pulldown-cmark](#) Rust library. You can set this to `read("somefile.md")` to import an external markdown file; see the [documentation for the read function](#).
  - Accepted values: Strings and raw text code blocks.
  - Required.
- `smart-punctuation`: Automatically convert ASCII punctuation to Unicode equivalents:
  - nondirectional quotations (" and ') become directional (“ and ”);
  - three consecutive full stops (...) become ellipses (...);
  - two and three consecutive hyphen-minus signs (-- and ---) become en and em dashes (– and —).

Note that although Typst also offers this functionality, this conversion is done through the Markdown parser rather than Typst.

- Accepted values: Booleans.
- Default value: true.
- `blockquote`: A callback to be used when a blockquote is encountered in the Markdown, or none if blockquotes should be treated as normal text. Because Typst does not support blockquotes natively, the user must configure this.
  - Accepted values: Functions accepting content and returning content, or none.
  - Default value: none.

For example, to display a black border to the left of the text one can use:

```
box.with(stroke: (left: 1pt + black), inset: (left: 5pt, y: 6pt))
```

| which displays like this.

- `math`: A callback to be used when equations are encountered in the Markdown, or none if it should be treated as normal text. Because Typst does not support LaTeX equations natively, the user must configure this.
  - Accepted values: Functions that take a boolean argument named `block` and a positional string argument (often, the `mitex` function from [the mitex package](#)), or none.
  - Default value: none.

For example, to render math equation as a Typst math block, one can use:

```
#import "@preview/mitex:0.2.4": mitex
#cmarker.render(`$\int_1^2 x \mathrm{d} x$`, math: mitex)
```

which renders as:  $\int_1^2 x \, dx$

- `h1-level`: The level that top-level headings in Markdown should get in Typst. When set to zero, top-level headings are treated as text, `##` headings become = headings, `###` headings become == headings, et cetera; when set to 2, `#` headings become == headings, `##` headings become === headings, et cetera.
  - Accepted values: Integers in the range [0, 255].
  - Default value: 1.
- `raw-typst`: Whether to allow raw Typst code to be injected into the document via HTML comments. If disabled, the comments will act as regular HTML comments.
  - Accepted values: Booleans.
  - Default value: true.

For example, when this is enabled, `<!--raw-typst #circle(radius: 10pt) -->` will result in a circle in the document (but only when rendered through Typst). See also `<!--typst-begin-exclude-->` and `<!--typst-end-exclude-->`, which is the inverse of this.

- `html`: The list of HTML elements that `cmarker` will support. Even if you provide an empty dictionary, the following elements will be supported by default: `<sub>`, `<sup>`, `<mark>`, `<h1>`–`<h6>`, `<hr>`, `<a>`, `<em>`, `<strong>`, `<s>`, `<br>`, `<img>`. You can override these if you want.
  - Accepted values: Dictionaries whose values can be:
    - `("normal", (attrs, body) => [/ * ... * /])`: Defines a normal element, where `attrs` is a dictionary of strings, `body` is content, and the function returns content.
    - `("void", (attrs) => [/ * ... * /])`: Defines a void element (e.g. `<br>`, `<img>`, `<hr>`).
    - `("raw-text", (attrs, body) => [/ * ... * /])`: Defines a raw text element (e.g. `<script>`, `<style>`), where `body` is a string.

- ("escapable-raw-text", (attrs, body) => [/ \* ... \* /]): Defines an escapable raw text element (e.g. <textarea>), where body is a string.
  - (attrs, body) => [/ \* ... \* /]: Shorthand for ("normal", (attrs, body) => [/ \* ... \* /]).
- Default value: (:).
- scope: When raw-typst is enabled, this is a dictionary providing the context in which the evaluated Typst code runs. It is useful to pass values in to code inside <!-- raw-typst --> blocks.
  - Accepted values: Any dictionary.
  - Default value: (:).
- show-source: A debugging tool. When set to true, the Typst code that would otherwise have been displayed will be instead rendered in a code block.
  - Accepted values: Booleans.
  - Default value: false.

This function returns the rendered content.

## 2. Resolving Paths Correctly

Because of how Typst handles paths, elements like images will by default resolve relative to the project root of cmarker itself and not your project.

To fix this, one can override the image function in the scope the Typst code is evaluated.

```
#import "@preview/cmarker:0.1.2"

#cmarker.render(
  read("yourfile.md"),
  scope: (image: (path, alt: none) => image(path, alt: alt))
)
```

## 3. Supported Markdown Syntax

We support CommonMark with a couple extensions.

- Paragraph breaks: Two newlines, i.e. one blank line.
- Hard line breaks (used more in poetry than prose): Put two spaces at the end of the line.
- *\*emphasis\** or `_emphasis_`: *emphasis*
- **\*\*strong\*\*** or `__strong__`: **strong**
- ~~~strikethrough~~~: ~~strikethrough~~
- [links](https://example.org): [links](https://example.org)
- **###** Headings, where # is a top-level heading, ## a subheading, ### a sub-subheading, etc
- ``inline code blocks``: inline code blocks
- `````
  - out of line code blocks

Syntax highlighting can be achieved by specifying a language after the opening backticks:

```
```rust
let x = 5;
```
```

giving:

```
let x = 5;
```

- `---`, making a horizontal rule:

- - Unordered
  - lists
- Unordered
- Lists
- 1. Ordered
  1. Lists
- 1. Ordered
- 2. Lists
- $x + y$  or  $x + y$ : math equations, if the math parameter is set.
- > blockquotes, if the blockquote parameter is set.
- Images: ![Some tiled hexagons](examples/hexagons.png), giving



- Tables:

```
Column 1	Column 2
Row 1 Cell 1	Row 1 Cell 2
Row 2 Cell 1	Row 2 Cell 2
```

| Column 1     | Column 2     |
|--------------|--------------|
| Row 1 Cell 1 | Row 1 Cell 2 |
| Row 2 Cell 1 | Row 2 Cell 2 |

- HTML, e.g. `<sub>subscript</sub>` for <sub>subscript</sub>.

## 4. Interleaving Markdown and Typst

Sometimes, you might want to render a certain section of the document only when viewed as Markdown, or only when viewed through Typst. To achieve the former, you can simply wrap the section in `<!--typst-begin-exclude-->` and `<!--typst-end-exclude-->`:

```
<!--typst-begin-exclude-->
Hello from not Typst!
<!--typst-end-exclude-->
```

Most Markdown parsers support HTML comments, so from their perspective this is no different to just writing out the Markdown directly; but `cmarker.typ` knows to search for those comments and avoid rendering the content in between.

Note that when the opening comment is followed by the end of an element, `cmarker.typ` will close the block for you. For example:

```
> <!--typst-begin-exclude-->
> One
```

Two

In this code, “Two” will be given no matter where the document is rendered. This is done to prevent us from generating invalid Typst code.

Conversely, one can put Typst code inside a HTML comment of the form `<!--raw-typst [...]-->` to have it evaluated directly as Typst code (but only if the `raw-typst` option to render is set to `true`, otherwise it will just be seen as a regular comment and removed):

```
<!--raw-typst Hello from #text(fill:blue)[Typst]!-->
```

## 5. Limitations

Although I tried my best to escape everything correctly, I won't provide a hard guarantee that everything is fully sandboxed even if you set `raw-typst: false`. That said, Typst itself is well-sandboxed anyway.

## 6. Development

- Build the plugin with `./build.sh`, which produces the `plugin.wasm` necessary to use this.
- Compile examples with `typst compile examples/{name}.typ --root ..`
- Compile this README to PDF with `typst compile README.typ`.