

Madoko Reference

A Fast Scholarly Markdown Processor

2014-10-27 (version 0.7.6-beta)

Daan Leijen
Microsoft Research
daan@microsoft.com

Table of contents

1. Overview of Madoko	4
1.1. Madoko philosophy	4
1.2. Performance and License	5
2. Installation and usage	5
3. Syntax: Inline elements	6
3.1. Emphasis	6
3.2. Code	7
3.3. Sub- and super-script	7
3.4. Strike-out	7
3.5. Smart quotes, symbols, and direct links	7
3.5.1. Advanced: changing quotes	9
3.6. Links	10
3.7. Images	10
3.8. Footnotes	11
3.9. Escape sequences	11
3.9.1. Special escapes	12
4. Syntax: Block elements	12
4.1. Headings and rules	12
4.2. Identities and labels	14
4.2.1. A named heading	14
4.3. Figures and images	15
4.4. Lists	17
4.5. Definition lists	18
4.6. Block quotes	19
4.7. Code blocks	19
4.7.1. Syntax highlighting	20
4.7.2. Escaped code	21
4.7.3. Advanced: Pretty code alignment	23
4.7.4. Advanced: Customizing highlight colors	26
4.7.5. Advanced: Custom syntax highlighting	27
4.7.6. Advanced: taking over code blocks	28
4.8. Tables	28
4.8.1. Horizontal rules	30
4.8.2. Long tables	31
4.8.3. The width of columns	31
4.8.4. Colors	31
4.8.5. Book tables	32
4.9. Mathematics	33
4.9.1. Using packages	33
4.9.2. Math commands	35
4.9.3. Mathematics in HTML	35
4.9.4. Advanced: Generic LaTeX snippets	36
4.9.5. Advanced: Efficiency of math rendering	42
4.9.6. Advanced: Preformatted math	42

4.9.7. Setting all code to preformatted math	44
4.10. Table of contents	44
4.10.1. Advanced: Custom tables of contents	44
4.11. Bibliography and Citations	44
4.11.1. Bibliography styles	45
4.11.2. Citation styles	46
4.11.3. Bibliography tooltips and searches	46
4.12. Custom blocks	47
4.12.1. Predefined custom blocks	48
4.13. Special block elements	50
4.13.1. Advanced: including file fragments	51
5. Syntax: Metadata and Styling	52
5.1. Metadata	52
5.1.1. Special metadata keys	52
5.1.2. HTML keys	54
5.1.3. LaTeX keys	56
5.2. Entities	57
5.3. Attributes and Styling	58
5.3.1. Attributes	58
5.3.2. CSS formatting attributes	59
5.3.3. Float	62
5.4. Metadata rules	64
5.4.1. Names of predefined elements	64
5.4.2. Advanced: Styling in CSS	65
5.5. Numbering	65
5.5.1. Advanced: Custom numbering	65
5.5.2. Advanced: Reset counters	66
5.5.3. Advanced: Display format	66
5.6. Advanced: Replacement	67
5.6.1. Advanced: Regular expression replacement	67
5.6.2. Advanced recursion and replacement	69
6. References	70
A. Appendix	70
A.1. Command line options	70
A.2. Slide shows and presentations	72
A.2.1. Using Reveal.js	72
A.2.2. Using Beamer	72
A.3. Advanced: Customizing citations	73
A.4. Advanced: Not using BibTeX	73
A.5. Advanced: packages in dynamic math mode	75
A.6. Advanced: Using Prettify to highlight code	75
A.7. Advanced styling in LaTeX	77
A.8. Special attribute keys	79
A.9. Special attribute classes	80
A.10. Unicode characters	81
A.10.1. Unicode in LaTeX	82

A.10.2. Unicode font selection in LaTeX	82
A.11. Recognized character entities	83
A.12. Definitions of predefined custom blocks	89

1. Overview of Madoko

Madoko is a fast javascript [Markdown](#) processor written in [Koka](#). It started out as a demo program for the new, strongly typed, [Koka](#) language and the name comes from “*Markdown in Koka*”.

Madoko can both be run local as a command-line program, or as a full online experience on [Madoko.net](#) with storage and collaboration through [dropbox](#).

1.1. Madoko philosophy

The main design goal of Madoko is to enable light-weight creation of high-quality scholarly¹ and industrial documents for the web and print, while maintaining John Gruber’s Markdown philosophy of simplicity and focus on plain text readability.

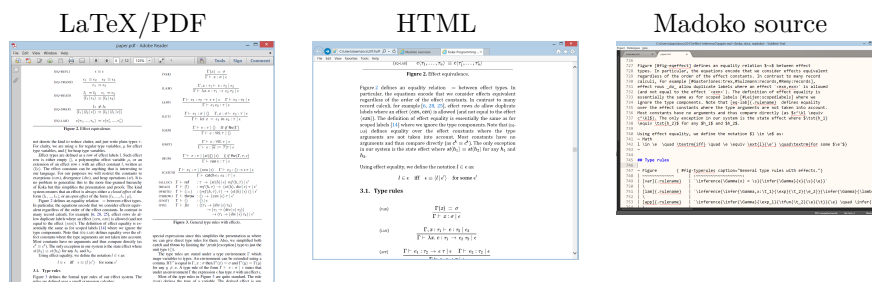
The popularity of Markdown is not accidental, and it is great for writing prose: it is super simple and straightforward to create good looking HTML documents. But for more serious use Markdown falls short in several areas, and Madoko provides many essential additions for larger documents:

- Extensive support for labeling and in-document [references](#).
- [Tables](#) with custom borders, column alignment, multicolumn spans, colors etc.
- Great [citation](#) support, using standard BibTeX entries and styles.
- Excellent support for static [syntax highlighting](#) code fragments.
- Automatic [numbering](#) of sections, figures, etc.
- Title page and [table-of-contents](#) generation.
- Support for [custom blocks](#), like **Theorem**, **Abstract**, **Figure**, etc.
- Styling for both HTML and LaTeX output through [standard CSS attributes](#).
- Powerful styling through attributes, [metadata rules](#) and [replacement rules](#).
- Excellent support for [advanced mathematics](#) and powerful packages like TikZ and PSTricks with hi-resolution images in web pages.
- Create great [presentations](#) for both the web and in print.
- Compatible with most common Markdown extensions, such as [footnotes](#), [mathematics](#), [attributes](#), etc.

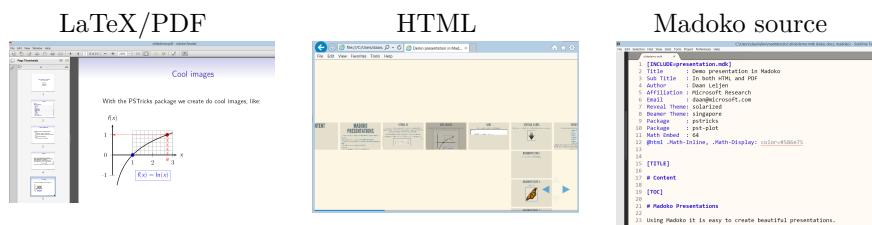
Instead of a plethora of backends, Madoko concentrates on generating either HTML or high-quality PDF files through [LaTeX](#). There has been a lot of effort in Madoko to make the LaTeX generation robust and customizable while

¹There is actually an effort to define [scholarly markdown](#).

integrating well with the various academic document- and bibliography styles. This makes it great for writing articles using just Madoko and get both a high-quality print format (PDF) and a great looking HTML page. Look at this scientific article for an example:



or this slide-show presentation:



Also, you can look at the [PDF](#) and [source](#) of this document. In the future, we plan to support e-books through the [ePub](#) format too.

1.2. Performance and License

Madoko is a javascript program (written in [Koka](#)) that runs on [Node.js](#). It is about 30% faster than [Marked](#) (one of the fastest Javascript markdown implementations), and about 6 to 8 times faster than [Showdown](#) and [Markdown.js](#). Madoko is also available as a .NET executable on windows.

Madoko is free software available under the [Apache 2.0 license](#) at madoko.codeplex.com.

2. Installation and usage

The recommended way to use Madoko now is online on Madoko.net in combination with [Dropbox](#) to store your files and collaborate and share with other users.

Nevertheless, Madoko works great as a command line tool as well, and it is good to have a solid backup. The easiest way to use Madoko is as a command line tool is by using Node.js (which works on many platforms, like Windows,

MacOSX, Linux etc). Madoko can also run inside a web browser or as a .NET executable. Installation under Node.js is very easy:

- Ensure you have [Node.js](#) installed on your system (and ensure that the Node installation directory is in your `PATH`).
- Open a command line window and run the Node package manager to install Madoko:
`npm install madoko -g`

and you are done. Translating a markdown document is done simply as:

- `madoko -v mydoc.mdk`

which generates `mydoc.html`. The `-v` flag gives more verbose output. To also generate a PDF file, use:

- `madoko --pdf -vv --odir=out mydoc`

where `--odir` puts all output files in the `out` directory. To generate a PDF, you need to have LaTeX installed on your system, which is also required for mathematics and bibliographies. We recommend the full [TeXLive](#) LaTeX system as it is available for Windows, Linux and MacOSX, and is used on the [Madoko.net](#) server as well.

For PDF output, we added an extra verbose flag in order to see any warnings LaTeX produces. A full description of all command line options can be found in [Appendix A.1](#).

3. Syntax: Inline elements

Madoko is fully compatible with basic [Markdown syntax](#) and passes the entire test suite. It also implements most extensions, like [Github flavored markdown](#), [PanDoc](#), [Markdown Extra](#), and [multi-markdown](#), and it adds quite a few features itself to make it really useful for writing academic and industrial documents.

We assume that the reader is familiar with [basic markdown syntax](#).

In Madoko, *tab*'s are considered to be equivalent to 4 spaces. It is therefore best to configure your editor to view tabs as 4 spaces wide or documents may look off.

3.1. Emphasis

Enclose words in asteriks (*) or underscores (_) to emphasize them. Use double asteriks or underscores for strong emphasis:

Here is `_some emphasis_`, or using `*asteriks*`.
 Or use `__strong emphasis__`, or like `**this**`.

Here is *some emphasis*, or using *asteriks*. Or use **strong emphasis**, or like **this**.

3.2. Code

You can include pre-formatted text in regular text using back-quotes (```).

For emphasis, use the `` tag in HTML.
We can use back-quotes by using multiple ``back`quotes``.

For emphasis, use the `` tag in HTML. We can use back-quotes by using multiple `back`quotes`.

Section 4.7 discusses code in more detail including syntax highlighting.

3.3. Sub- and super-script

Using tilde (`~`) and hat (`^`), you can format inline text as sub- and super-script respectively. Inside script, no white space is allowed to prevent mistakes. If you need white space you can still use an escaped space (`\`).

Here is how you write H²O or E=MC².
Please use escapes for longer script.

Here is how you write H₂O or E=MC². Please use escapes for longer script.

3.4. Strike-out

Enclose anything in two tildes and it will strike out the content:

There is a ~~strike out~~ here.

There is a ~~strike out~~ here.

3.5. Smart quotes, symbols, and direct links

Madoko will quote smartly using proper open and closing quotes for anything enclosed in single quotes (`'`), double quotes (`"`), or french quotes (`<<` and `>>`).

```
"double quoted"
```

```
'single quoted'
```



```
<<guillemet quoted>>
<http://www.google.com>
```

```
“double quoted”
’single quoted’
«guillemet quoted»
http://www.google.com
```

Note that text enclosed in < and > brackets gets interpreted as direct link. Madoko will only smart quote single quotes if the last quote is not directly followed by a letter. This often prevents wrong quotation with words like “can’t”:

```
'really, I can't do this', he said.
I can't and mustn't do this.
```

```
‘really, I can’t do this’, he said. I can’t and mustn’t do this.
```

Madoko will also replace multiple dashes and dots to a proper symbol:

```
Please distinguish a minus sign, -, from the _en dash_ which
is used to separate spans or pages, like 1--20, and the
_em dash_ which is even longer and sometimes used for
quotation attribution. --- Oscar Wilde.
Three dots ... should be close together.
```

```
Please distinguish a minus sign, -, from the en dash which is used to sep-
arate spans or pages, like 1–20, and the em dash which is even longer and
sometimes used for quotation attribution. — Oscar Wilde.
Three dots ... should be close together.
```

3.5.1. Advanced: changing quotes

When replacing quotes, Madoko just inserts the right entity, like an `“`; for a left double quote (“). As described in Section 5.2 we can redefine entities, and use that to change the quotes. For example, in some obscure countries, like the Netherlands, people like to start a quotation at „the bottom” instead. You can get this behaviour by adding the following `metadata` rule:

```
ldquo: &bdquo;
```

Or Japanese style:

```
lsquo: &#12300;
```

```
rsquo: &#12301;
```

```
ldquo: &#12302;
```

rdquo: `』`

to single quote like this .

3.6. Links

Madoko has three kinds of links, reference links, inline links, and direct links. The inline links have the linked text between square brackets and the URL follows between parenthesis, while direct links are simply enclosed between angled brackets:

```
Here is a link to [Google] (http://www.google.com).
Or as a direct link: <http://www.google.com>.
```

Here is a link to [Google](http://www.google.com). Or as a direct link: <http://www.google.com>.

Generally, reference links are preferred. Here, the link is defined separately after the body of text such that it looks less cluttered:

```
Here is a link to [Google] again.
We can also [Change the text][Google].
```

```
[Google]: http://www.google.com "Google"
```

Here is a link to [Google](http://www.google.com) again. We can also [Change the text](#).

Note how `[Google]` is simply a shorthand for `[Google][Google]`. The title in double quotes in the link definition is optional. Link definitions do not have to follow the text immediately and can be defined anywhere in the document.

3.7. Images

Images are included using a regular link prefixed with an exclamation mark (!).

```
A butterfly: ![bfly].
```

```
[bfly]: butterfly.png "A Monarch" { width=100px }
```

A butterfly:  .

This example also shows the use of attributes (Section 5.3.1) where we can specify the `width`, `height`, or `vertical-align` of the image.

[madoko.net](#): You can use to the toolbox menu to include images or just drag&drop them directly into the editor. Just watch the filesize as images larger than about 1mb are rejected by the Madoko server.

3.8. Footnotes

Footnotes are written as regular link definitions prefixed with a hat (^) character. This is the syntax [originally proposed](#) by John Gruber.

Here is a footnote^[^fn].

[^fn]: This is the content of the example footnote.
 You can continue a footnote by indenting content.
 And notice the backlink.

Here is a footnote².

²This is the content of the example footnote. You can continue a footnote by indenting content. And notice the backlink.

3.9. Escape sequences

If you want to use a special character directly without a special Markdown meaning, precede it with a backslash (\). For example, to use a star (*) without causing emphasis, you can write *.

Madoko will never escape a letter or digit and always keep the backslash, so \a becomes “\a” while \& becomes just “&”. Thus, you can safely write windows style file names without needing an escape:

Would you like c:\foo\bar to be deleted? Yes\No.
 Here are some other escapes: \, \#, *, \|, and \0.

Would you like c:\foo\bar to be deleted? Yes\No. Here are some other escapes: \, #, *, |, and \0.

This approach is different than that of Markdown which only escapes a specific set of characters, while Madoko escapes everything that is not a letter or digit. The advantage of the approach of Madoko is that this is easy to remember, while trying to remember a specific set of special escape characters is near impossible. This is similar to the [PanDoc](#) approach to escape sequences.

3.9.1. Special escapes

Some characters are translated specially when escaped. If you escape a space (`\`), it is translated as a non-breakable space, while a backslash at the end of a line causes a hard line break to be inserted. The latter is recommended over using the standard Markdown way of using two spaces at the end of a line because it leaves visual clue that a line break occurs.

```
Here is non\ breakable space and a hard\
line break with a \* star.
```

```
Here is non breakable space and a hard
line break with a * star.
```

Finally, the escape sequence `\/` translates to nothing; this can be very useful to separate certain constructs. For example, emphasis is suppressed if the underscores appear inside a word, as in `my_example_here`. Using the empty escape sequence we can emphasize inside words too: to get *myexamplehere*, we can simply write `my\/_example_\/here`.

4. Syntax: Block elements

4.1. Headings and rules

Headings are written by prefixing with one or more hash characters (`#`):

```
# A level one heading
## A level two heading
### A level three heading
...
##### Up to level six
```

It is also possible to write level one and level two headers using a line of three or more equal (`=`) or dash (`-`) characters:

```
A level one heading
=====
```

```
A level two heading
-----
```

But hash-headings are generally preferred since a three or more dashes (or underscores or asteriks) are also used for horizontal rules:

Above...

and below the line.

Above...

and below the line.

4.2. Identities and labels

Madoko has extensive support for numbering and referencing elements in a document. Elements are given an identity using attributes (see Section 5.3.1). For example, here we give an identifier `myheading` to a heading:

```
### A named heading { #myheading }

And we can refer to it

* Using an explicit [link](#myheading)
  (or [reference](#myheading)).
* Or using an implicit link to Section [#myheading].
* Or we can just see its label, namely &myheading;.
```

4.2.1. A named heading

And we can refer to it

- Using an explicit [link](#) (or [reference](#)).
- Or using an implicit link to Section 4.2.1.
- Or we can just see its label, namely 4.2.1.

Using an implicit link is generally recommended. When typesetting a reference such as Section 4.2.1, Madoko automatically inserts a non-breakable space between “Section” and the reference³. Unlike LaTeX there is no need to insert such non-breakable space yourself.

Of course, we can refer to *any* element that has an identity, like equations, figures, tables, etc. Here is an example with an equation (see Section 4.9):

```
In Equation [#euler] we define [Euler]'s number $e$:

~ Equation { #euler }
```

³Madoko will still use a regular space if you use an explicit line break between “Section” and the reference in the source.

```
e = \lim_{n\to\infty} \left( 1 + \frac{1}{n} \right)^n
```

```
[Euler]: http://en.wikipedia.org/wiki/Euler's_number
```

In Equation (1) we define Euler’s number e :

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n} \right)^n \quad (1)$$

An implicit link such as `[#myheading]` is a shorthand for an explicit link of the form `[&myheading;](#myheading)`, i.e. a local reference to `#myheading` which displays the entity `&myheading`. In Madoko, an entity name like `&myheading` is replaced by the *label* value of the referred block (see Section 5.2). For headers, this is by default the header number (as we will see in Section 5.5) but you can set any label you’d like. Using a minus (-) in the attributes clears all attributes, which is used in the following example to suppress the default numbering:

```
### An unnumbered heading { - }
### A labeled heading      { - #myheading1 label="my label" }
Let's refer to Section [#myheading1].
```

An unnumbered heading

A labeled heading

Let’s refer to Section [my label](#).

Also, any local link, like “Section 4.2.1” or a citation like “[2]”, displays a tooltip when hovering above it. The tooltip content is determined by the `caption` attribute of the target element. For headings, this is by default the heading text.

[madoko.net](#): In the toolbox menu you can insert local references under the # menu which shows all labels in the document.

4.3. Figures and images

Figures can be included using the **Figure** custom block (see Section 4.12):

Figure [#fig-monarch] shows how to put an image in a figure.

```
~ Figure { #fig-monarch caption="A Monarch butterfly." }  
The ![monarch] image.  
~
```




Figure 1. A Monarch butterfly.

```
[monarch]: butterfly.png {width=100px vertical-align=middle}
```

Figure 1 shows how to put an image in a figure.

A figure can be given an identity and referred to just like headings. The `caption` attribute gives the caption of the figure which is also used in the table-of-figures. A **Figure** can have a `page-align` attribute that can be set to `top`, `bottom`, `page`, `here` and `forcehere`. This is ignored in the HTML output but used in LaTeX to influence where a figure is placed on a page. The `.wide` attribute is used in LaTeX to have figures span both columns in a two-column document class.

4.4. Lists

Unordered lists are created simply by using an asteriks (*), plus (+), or dash (-) as item markers preceded by an empty line (or otherwise it is considered part of a paragraph):

```
Groceries:
```

- ```
* Banana
* Bread
 - white
 - whole grain
* Basil
```

---

```
Groceries:
```

- ```
• Banana
• Bread
  - white
  - whole grain
• Basil
```

Each item marker must be followed by a space. You can created ordered lists using numbers followed by a dot and space. Also long list items can be wrapped by indenting the text for each item:

- ```
1. An ordered list example.
```

```

 With some longer items.
2. An another item
 with more text.

```

- 
1. An ordered list example. With some longer items.
  2. An another item with more text.

Normally, the content of each list item is just treated as text and not as a paragraph. If there are any blank lines in the entire list, each item is typeset with a surrounding paragraph which makes the list a bit looser for HTML output:

```

3. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
 Aliquam hendrerit mi posuere lectus.

3. Vestibulum enim wisi,
 viverra nec, fringilla in, laoreet vitae, risus.

3. Donec sit amet nisl. Aliquam semper ipsum sit amet velit.
 Suspendisse id sem consectetur libero luctus adipiscing.

```

---

```

3. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam
 hendrerit mi posuere lectus.

4. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.

5. Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Sus-
 pendisse id sem consectetur libero luctus adipiscing.

```

Note how the specific numbers used in the list do not matter, except that the first item determines the start number of the enumeration.

For ease of formatting, unordered lists get a class assigned corresponding to their first bullet:

1. A bullet `*` gets class `list-star` (default display as `•`).
2. A bullet `+` gets class `list-plus` (default display as `■`).
3. A bullet `-` gets class `list-dash` (default display as `-`).

This can be used in CSS or Latex to customize how the bullets are displayed.

## 4.5. Definition lists

Definition lists are used to define terms. Each term must fit on one line and is followed by one or more definitions. Each definition starts with one or two spaces followed by a tilde (`~`) or colon (`:`). Each definition must be indented by four spaces.

**Abstract syntax**

- ~ The conceptual structure (illustrated by the pictures) is called the abstract syntax of the language.

**Concrete syntax**

- ~ The particular details and rules for writing expressions as strings of characters is called the concrete syntax.
- ~ Perhaps some other meaning too?

---

**Abstract syntax**

The conceptual structure (illustrated by the pictures) is called the abstract syntax of the language.

**Concrete syntax**

The particular details and rules for writing expressions as strings of characters is called the concrete syntax.

Perhaps some other meaning too?

## 4.6. Block quotes

Block quotes are written just as in email using > angle brackets:

Let's start a block quote:

```
> Of life's two chief prizes, beauty and truth,
> I found the first in a loving heart and the
> second in a laborer's hand.\
>    --- Khalil Gibran
```

---

Let's start a block quote:

Of life's two chief prizes, beauty and truth, I found the first in a loving heart and the second in a laborer's hand.  
— Khalil Gibran

Just like lists, block quotes can be nested and contain other block elements.

## 4.7. Code blocks

You can write preformatted code simply by indenting the code with a tab character or at least *four* spaces. For example:

```
a link in html
```

---

```
a link in html
```

Another way to write code is to use *fenced* code blocks. These start with at least three backticks (```) and goes on to the first line containing the same number of backticks. Moreover, you can include the language name after the backticks:

```
``` html
<a href="foo.com">a link in html</a>
```
```

---

```
a link in html
```

#### 4.7.1. Syntax highlighting

As we can see from the previous example, Madoko will automatically perform (static) syntax highlighting for any code fragment that has the `language` key set. For a fenced code block, Madoko automatically adds the `language` key when the language is specified.

Internally, Madoko uses the Monarch library to do syntax highlighting. This means that also the PDF output through LaTeX will be fully highlighted (oh what a joy to no longer having to remember the vagaries of the `listing` package). The standard library contains some default language definition, like Java, JavaScript, C#, HTML, Ruby, and Python.

```
A Haskell keyword like `instance`{language=haskell} can be highlighted inline.
Or some Ruby code block:
``` Ruby
# ruby pi - how to calculate pi with ruby.
num = 4.0
pi = 0
plus = true
den = 1
while den < 10000000
  if plus
    pi = pi + num/den
    plus = false
  else
    pi = pi - num/den
    plus = true
  end
  den = den + 2
end

puts "PI = #{pi}"    # calculated value of pi
```

```
...
```

A Haskell keyword like `instance` can be highlighted inline. Or some Ruby code block:

```
# ruby pi - how to calculate pi with ruby.
num = 4.0
pi = 0
plus = true
den = 1
while den < 10000000
  if plus
    pi = pi + num/den
    plus = false
  else
    pi = pi - num/den
    plus = true
  end
  den = den + 2
end

puts "PI = #{pi}" # calculated value of pi
```

If you use a particular language a lot in your document, you may want to set it as the default language using a metadata rule. For example:

```
~Pre,~Code: language=JavaScript
```

If you want to disable syntax highlighting for a particular code fragment, you can set the language key to empty. To disable highlighting completely, set the `Highlight` metadata key to `False`.

For HTML, it is also possible to use dynamic syntax highlighting with the [Prettify](#) library, see [Appendix A.6](#) for more information.

4.7.2. Escaped code

Madoko allows you to escape back to inline madoko syntax inside code by bracketing inside `\(` and `\)`. This can be nice to insert special looking symbols in the code while maintaining full automatic syntax highlighting for all the surrounding code:

```
``` javascript
function sqr(x) {
 var \(π\) = 3.141593;
 return x*x /* ensures \(_result_ \ge 0\) on return.*/
}
```
```

```
function sqr( x ) {
  var  $\pi$  = 3.141593;
  return x*x /* ensures result  $\geq 0$  on return.*/
}
```

This is especially powerful in combination with replace attributes (Section 5.6). For example, in this document, we have the following metadata rules:

```
~Pre,~Code: replace="/==>/\(&rArr;\)/g"
```

which makes it easy to insert an implication symbol in code:

```
Look at implication (1) in the following code:
```javascript
function implies(x,y) { x ==> y } \((**1**)\)
```
```

Look at implication (1) in the following code:

```
function implies(x,y) { x  $\Rightarrow$  y } (1)
```

Each escaped piece of code is wrapped in a `code-escaped` element. This can be used to for example render all escaped text in a serif font:

```
~code-escaped: font-family=serif
```

Finally, you can use the `.noescape` class to suppress escaping.

When encountering a replaced piece of text, the syntax highlighter treats it by default as whitespace. It is possible to specify though what characters the syntax highlighter sees using the bar format `.`. For example, to colorize our implication symbol as a keyword, we can do:

```
Look at highlighted implication (1) in the following code:
```javascript
function implies(x,y) { x \(\return|\⇒\) y } \((**1**)\)
```
```

Look at highlighted implication (1) in the following code:

```
function implies(x,y) { x  $\Rightarrow$  y } (1)
```

4.7.3. Advanced: Pretty code alignment

When writing high quality articles containing code fragments, it often looks best when using a serif font instead of monospace. However, it becomes hard to align the code properly in such cases. Madoko offers a special `pretty` mode for

advanced code alignment inspired by the excellent [lhs2tex](#) tool by Andres Löh.

We use a proportional font, but everything preceded by 2 or more spaces is aligned properly:

```
``` Haskell { .pretty }
data Exp = Number Int
 | Add Exp Exp
 | Subtract Exp Exp
 | Multiply Exp Exp
 | Divide Exp Exp
 | Variable String -- added
 deriving (Eq)
...

```

We use a proportional font, but everything preceded by 2 or more spaces is aligned properly:

```
data Exp = Number Int
 | Add Exp Exp
 | Subtract Exp Exp
 | Multiply Exp Exp
 | Divide Exp Exp
 | Variable String – added
 deriving (Eq)

```

Even though the font used is proportional everything which is preceded by 2 or more spaces is aligned. If something is indented more, there is some default indentation being added. For example:

```
``` haskell { .pretty replace="/->/\(->|&rarr;\)/g" }
substitute1 :: (String, Int) -> Exp -> Exp
substitute1 (var, val) exp = subst exp where
  subst (Number i)      = Number i
  subst (Add a b)       = Add (subst a) (subst b)
  subst (Subtract a b)  = Subtract (subst a) (subst b)
  subst (Multiply a b)  = Multiply (subst a) (subst b)
  subst (Divide a b)    = Divide (subst a) (subst b)
  subst (Variable name) = if var == name
                          then Number val
                          else Variable name
...

```



```

substitute1 :: (String, Int) → Exp → Exp
substitute1 (var, val) exp = subst exp where
  subst (Number i)      = Number i
  subst (Add a b)       = Add (subst a) (subst b)
  subst (Subtract a b)  = Subtract (subst a) (subst b)
  subst (Multiply a b)  = Multiply (subst a) (subst b)
  subst (Divide a b)    = Divide (subst a) (subst b)
  subst (Variable name) = if var == name
                          then Number val
                          else Variable name

```

Note that we used a replacement expression to replace arrows with their proper symbol. To make alignment and syntax highlighting work properly, we used the bar format. In this case we wrote `\(->|` which makes the replacement count for 2 characters for alignment purposes (and uses `->` for the syntax highlighter). In LaTeX the samples generally align very nice. Here is how the above example is rendered in LaTeX (where we defined `\mdTokenPrettyType` to show a teal color):

```

substitute1 :: (String, Int) → Exp → Exp
substitute1 (var, val) exp = subst exp where
  subst (Number i)      = Number i
  subst (Add a b)       = Add (subst a) (subst b)
  subst (Subtract a b)  = Subtract (subst a) (subst b)
  subst (Multiply a b)  = Multiply (subst a) (subst b)
  subst (Divide a b)    = Divide (subst a) (subst b)
  subst (Variable name) = if var == name
                          then Number val
                          else Variable name

```

Finally, for really pretty code, it generally helps to add more replacements. For example, here is a sample from the [lhs2tex](#) manual:

```

```` Haskell { .pretty \
 replace="//->/\(->|⇉\)/\(\(?![()])/\(λ\)/g" \
 replace="/_ (?=[a-zA-Z])/\(_|&lowline;\)/g" \
 replace="/([a-zA-Z])(\d)\b/\(\\1~\\2~\\)/g" }
rep_alg = (_ -> \m -> Leaf m
 , \lfun rfun -> \m -> let lt = lfun m
 rt = rfun m
 in Bin lt rt "hi"
)
replace_min t = (cata_tree rep_alg t) (cata_tree min_alg t)
````

```

```

rep_alg      = ( \_      → λm → Leaf m
                , λlfun rfun → λm → let lt = lfun m
                                   rt = rfun m
                                   in Bin lt rt "hi"
                )
replace_min t = (cata_tree rep_alg t) (cata_tree min_alg t)

```

In LaTeX this sample is rendered as:

```

rep_alg      = ( \_      → λm → Leaf m
                , λlfun rfun → λm → let lt = lfun m
                                   rt = rfun m
                                   in Bin lt rt "hi"
                )
replace_min' t = (cata_tree rep_alg t) (cata_tree min_alg t)

```

Note that to improve the rendering, we added replacements for underscores inside identifiers with `&lowline`; and subscripted digits that end an identifier. Also, we replaced right arrows and lambda bindings.

4.7.4. Advanced: Customizing highlight colors

The syntax highlighter for a language assigns specific class names to each token. Using these classes we can customize how they are displayed. The standard class names that are often used include:

identifier, operators, keyword, string, escape, comment, commentdoc, constant, entity, tag, info, warn, error, debug, regexp, attribute, value, constructor, namespace, header, type, predefined, invalid, code, codekeyword, typevar, delimiter, number, variable, meta, bold, and italic.

To customize the appearance in HTML, just use a CSS style declaration:

```

<style>
.token.identifier { font-style: italic }
.token.string.escape { font-weight: bold }
</style>

```

In LaTeX, just define a `\mdTokenToken` command. For example:

```

~ TexRaw
\defcommand{\mdTokenIdentifier}[1]{\textit{#1}}
\defcommand{\mdTokenEscape}[1]{\textbf{#1}}
~

```

We use `\defcommand` which is like `\newcommand` but doesn't fail if the command was already defined. For identifiers in *pretty* code, the token rendering is determined by `mdTokenPrettyToken` commands. By default, these tokens are not colorized in LaTeX but this is easy to customize:

```

~ TexRaw
\defcommand{\mdTokenPrettyKeyword}[1]{\textcolor{blue}{\textsf{#1}}}
~

```

4.7.5. Advanced: Custom syntax highlighting

Madoko can actually load syntax specifications dynamically to perform syntax highlighting. For example, suppose you are a blogger that is writing about the new Javascript module proposal, and you would like to highlight the new `module` or `export` keywords. Unfortunately, regular Javascript highlighting does no such thing:

```
// Javascript 5 modules
module Math {
  export function sum(x, y) {
    return x + y;
  }
  export var pi = 3.141593;
}
```

In Madoko, you can add your own syntax highlighter quite easily. First, register your new language definition in the metadata:

```
Colorizer: javascript
Colorizer: javascript5
```

Because we are going to extend `javascript`, we also mention that language. Now you need a file `javascript5.json` which will contain the new language definition as JSON:

```
{ "name": "javascript5",
  "extend": "javascript",
  "extraKeywords": ["module", "export", "import"]
}
```

This is generally all that is needed in most situations, but if you like to get fancy, there is extensive [documentation](#) about writing syntax extensions. (or use `Colorizer: standard-language` to see the definitions of standard languages that come with Madoko (like Javascript, Java, CSharp, Ruby, and Python).

Et voilà, with our new language definition we can properly highlight our previous example:

```
``` javascript5
// Javascript 5 modules
module Math {
 export function sum(x, y) {
 return x + y;
 }
 export var pi = 3.141593;
}
```
```

```
// Javascript 5 modules
module Math {
  export function sum(x, y) {
    return x + y;
  }
  export var pi = 3.141593;
}
```

As an aside, note that colorization is done statically by Madoko, and not dynamically when a user views your document (and therefore, it works in LaTeX/PDF too).

There are many built-in languages in Madoko, and you can look at their JSON definitions. Here are a few: [Java](#), [Dafny](#), [C#](#), [Javascript](#), [HTML](#), [LaTeX](#), [BibTeX](#), [Madoko](#), [Python](#), [Ruby](#), [SMT](#), [Koka](#), [C++](#), etc. If you write your own JSON description, you can simply include it yourself through the `Colorizer` metadata key.

4.7.6. Advanced: taking over code blocks

It is possible to ‘take over’ code blocks and use them for example for mathematics. For example, here is how to make all code blocks use pre-formatted math (Section 4.9.6):

```
~Pre,~Code: input=mathpre
```

Moreover, you can selectively only take over different kinds of code. Use `.code n` for inline code with n quotes, and `.pre-indented` or `pre-fenced` for indented or fenced code blocks. For example, here is how to use a default language only for indented and double quoted (```) code:

```
.pre-indented,.code2: language=koka
```

4.8. Tables

Madoko significantly extends the table syntax of basic Markdown. In particular, it is easy to add horizontal or vertical lines, to control cell alignment, use multiple column spans, colorize rows, etc. The basic rules for formatting a table are:

- Every line of the table should start and end with a `|` (or `+`) and columns are separated by `|` (or `+`) too. If you need a `|` character in cell content, use an escaped bar instead (`\|`).
- Every row can be on one line only, and there can be no blank lines.
- The table can optionally start with one or more *header* rows.
- A cell can span multiple columns by using multiple bars to end the cell, like `||` in the previous example.
- The table should always have a *column specifier* row that separates the header from the body of the table (or marks the start of the body in case there is no header). The content of each cell in the separator is just dashes (`-`) or tildes (`~`)

Here is an example of a plain table from “[Just a Theory](#)”.

| id | name | description | price |
|------|-----------------------|--------------------------|--------|
| 1 | gizmo | Takes care of doohickies | 1.99 |
| 2 | doodad | Collects *gizmos* | 23.80 |
| 10 | dojigger | Foo | 102.98 |
| 1024 | Self-explanatory, no? | | 0.99 |

The column specifier row is the second row in the previous example, and it determines column alignment and vertical lines in a table:

- Columns can be aligned by using a : in a separator row column: one on the right or left aligns to the right or left, while a colon on both sides will center the column.
- If a column specifier uses a plus (+) instead of a bar (|) to separate the column, a vertical line is used to separate the columns. To distinguish the use of a + for a table instead of as a list item, there should be *no whitespace* following a + when used this way!
- If a column specifier cell uses dashes -, a horizontal line is drawn. By using tildes (~) instead, no line is drawn for that column specifier cell.

In the next example, we suppress the horizontal line after the header, but add some vertical lines. Also, we use two header rows.

| id | name | description | price |
|------|-----------------------|--------------------------|--------|
| 1 | gizmo | Takes care of doohickies | 1.99 |
| 2 | doodad | Collects *gizmos* | 23.80 |
| 10 | dojigger | Foo | 102.98 |
| 1024 | Self-explanatory, no? | | 0.99 |

madoko.net: You can press **Alt-Q** to reformat tables (or paragraphs) and align all columns. Also, pressing **enter** in a table will add a new row (use **ctrl-enter** to use a regular line break if needed).

4.8.1. Horizontal rules

We can draw horizontal rules in a table by using a row where every cell just contains dashes (-). By using equal signs (=) we get a double horizontal line. Here is a a table with no header and an outer border:

```

+:----:|-----+:-----+:-----+:
| centered gizmos  || Takes care of doohickies  | 1.99 |
| 2 | doodad | Collects *gizmos* | 23.80 |
| 10 | dojigger | Escaped \| and \+ | 102.98 |
1024	thingamabob	Self-explanatory, no?	0.99

```

| | | |
|------------------|--------------------------|--------|
| centered gizmos | Takes care of doohickies | 1.99 |
| 2 doodad | Collects <i>*gizmos*</i> | 23.80 |
| 10 dojigger | Escaped \ and \+ | 102.98 |
| 1024 thingamabob | Self-explanatory, no? | 0.99 |

And finally a complex table with all kinds of alignment, multiple column spans and horizontal rules – imagine trying to draw this example table in HTML or LaTeX without consulting the manual.

```

| ----- | ----- | ----- | ----- |
| id | name | description | price |
+-----+-----+:-----+:~~~~~+:
1	gizmo		
2	doodad	Collect *gizmos*	23.80
=====	-----	=====	-----
1024	thingamabob	Self-explanatory	0.99

```

| id | name | description | price |
|------|-------------|-------------------------|-------|
| 1 | gizmo | | |
| 2 | doodad | Collect <i>*gizmos*</i> | 23.80 |
| 1024 | thingamabob | Self-explanatory | 0.99 |

Note how we started the previous table with a horizontal line, where the column specifier row is on the third line.

4.8.2. Long tables

If a table is expected to be long and cross multiple pages, the table should be followed by an attribute declaration that sets the `.long` class. This is used in LaTeX output to switch to a `longtable` environment which supports tables that can be broken over multiple pages.

4.8.3. The width of columns

We can specify the width (and other attributes) of a column by adding an `attribute` specification in the column specification row. In the following example, we use a column of fixed width, and one of a relative width with respect to the overall width of the table.

```

+:----:|---{width=2cm}---+:-----{width=60%}-----:++-----:
centered gizmos		Take care of doo hickies	1.99
2	doodad	Collects *gizmos*	23.80
10	dojigger	Foo	102.98
1024	thingabob	Self-explanatory, no?	0.99

```

| | | |
|-----------------|--------------------------|--------|
| centered gizmos | Take care of doo hickies | 1.99 |
| 2 doodad | Collects <i>gizmos</i> | 23.80 |
| 10 dojigger | Foo | 102.98 |
| 1024 thingabob | Self-explanatory, no? | 0.99 |

4.8.4. Colors

Often we need to colorize certain rows or columns for clarity. Just like `width` we can specify a background color for a column in the column specifier row.

Moreover, we can specify after the table more attributes that can style the table further. In particular, any attributes starting with `th-` or `tr-` apply to the header or body rows respectively. When we add `even-` or `odd-` the attribute only applies to even or odd rows. The following example shows this in action:

```

| ---- | ----- | ----- | ----- |
| id | name | description | price |
+:----:++{background-color=Silver}---+:-----:|-----:++
centered gizmos		Take care of doo hickies	1.99
2	doodad	Collects *gizmos*	23.80
10	dojigger	Foo	102.98
1024	thingabob	Self-explanatory, no?	0.99
{ tr-odd-background-color=Gainsboro \

```

| | | |
|-----------------|--------------------------|--------|
| centered gizmos | Take care of doo hickies | 1.99 |
| 2 doodad | Collects <i>gizmos</i> | 23.80 |
| 10 dojigger | Foo | 102.98 |
| 1024 thingabob | Self-explanatory, no? | 0.99 |

4.8.5. Book tables

1. Never use vertical rules in a table, and,
2. Never use double horizontal rules.

An example of a 'book table':

```

| ----- | ----- | ----- |
|          | Item          |          |
| \ / ----- | ----- |          |
Animal	Description	&ensp;Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo&ensp;	frozen	8.99
-----	-----	-----
{ .booktable }

```

An example of a 'book table':

| Item | | |
|-----------|-------------|------------|
| Animal | Description | Price (\$) |
| Gnat | per gram | 13.65 |
| | each | 0.01 |
| Gnu | stuffed | 92.50 |
| Emu | stuffed | 33.33 |
| Armadillo | frozen | 8.99 |

Note how we used the empty escape sequence `\` in front of the second dashed row in the header, or otherwise Madoko would interpret that row as the column specification row. The `.booktable` class ensures more vertical space between the rules, and a heavier top, mid-, and bottom rule. The exact styling is specified using a (built-in) [metadata rule](#) which is shown in [Appendix A.12](#).

4.9. Mathematics

Madoko documents can include mathematics in standard LaTeX syntax. Generally, inline math should be typeset between `$` characters, while block equations should use the `Equation` block syntax.

A famous equation is `$E = mc^2$`, but another famous one is:
`~ Equation {#eq-gaussian}`
`\int_{-\infty}^{\infty} e^{-a x^2} d x = \sqrt{\frac{\pi}{a}}`
`~`
 and we can refer to Equation `[#eq-gaussian]` like any heading.

A famous equation is $E = mc^2$, but another famous one is:

$$\int_{-\infty}^{\infty} e^{-ax^2} dx = \sqrt{\frac{\pi}{a}} \quad (2)$$

and we can refer to Equation [\(2\)](#) like any heading.

Block equations can also be included use the `Math` custom block which is just like the standard LaTeX display environment (using `$$` or `\[`). However, the `Equation` block is preferred as it takes care of numbering, and alignment. See also [Section 4.12](#) on custom blocks that support mathematics, like `Theorem`, `Lemma`, `Proof`, etc. Madoko does not support the `\(`, `\[`, and `$$` commands to enclose mathematics since those are escape sequences in Madoko.

4.9.1. Using packages

By default, Madoko supports most of the AMS mathematics commands, i.e. `amsmath`, `amsfonts`, and `amssymb`. However, sometimes you need specific packages. You can import more packages using the `Package` metadata key. For

example, we can include the [amscd](#) package,

Package: [amscd](#)

to draw commutative diagrams:

```
~ Equation
\begin{CD}
S^{\{\mathcal{W}\}}_{\Lambda} \otimes T @>j>> T \\
@VVV @VVV{P} \\
(S \otimes T)/I @= (Z \otimes T)/J
\end{CD}
```

~

$$\begin{array}{ccc}
 S^{\mathcal{W}_{\Lambda}} \otimes T & \xrightarrow{j} & T \\
 \downarrow & & \downarrow P \\
 (S \otimes T)/I & \equiv & (Z \otimes T)/J
 \end{array} \tag{3}$$

As a hint to understand the previous example, note that `@VVV` denotes a downward arrow, `@>j>>` a right pointing arrow with a label `j` on top, and `@=` a long equality.

Here is an example of the more modern [xypic](#) package which we included in this document with the `curve` option:

Package: [\[curve\]xypic](#)

With this package, we can draw more complex diagrams.

```
~ Equation
\xymatrix @-0.5em{
U \ar@/_/[ddr]_y \ar@/^/[drr]^x \\
\ar@{.}>[dr]|-{\langle x,y \rangle} \\
& X \times_Z Y \ar[d]^q \ar[r]_p \\
& X \ar[d]_f \\
& Y \ar[r]^g \& Z }

```

~

$$\begin{array}{c}
 U \xrightarrow{x} X \\
 \text{\scriptsize (x,y)} \searrow \text{\scriptsize y} \\
 X \times_Z Y \xrightarrow{p} X \\
 \downarrow q \qquad \downarrow f \\
 Y \xrightarrow{g} Z
 \end{array}
 \tag{4}$$

As a hint to understand the above code, note that `\ar@/_/[ddr]_y` denotes an arrow, left curved (`@/_/`), going down, down, right (`[ddr]`), with a label `y` underneath. See the package [user guide](#) for more examples.

[madoko.net](#): You can by default include any package included in [TeXLive](#) (which are many). If you need to include a local package (or document class, bibliography etc) you can either use the toolbox menu or just drag&drop it into the editor.

4.9.2. Math commands

For math-heavy documents, it is convenient to define LaTeX commands for common operations. Such command definitions can be directly understood by LaTeX but need to be handled specially if the math is rendered dynamically in a HTML page. Madoko defines the custom block `MathDefs` to support mathematics definitions transparently across modes:

```

~ MathDefs
\newcommand{\infer}[3]{#1 \vdash #2\, : #3}
~
We infer  $\Gamma \vdash e : \tau$  for such expression  $e$ .

```

```

We infer  $\Gamma \vdash e : \tau$  for such expression  $e$ .

```

Often it is convenient to put all such definitions in a separate `.tex` file and include it in the document as:

```

~ MathDefs
[INCLUDE="mathdefs.tex"]
~

```

4.9.3. Mathematics in HTML

To typeset mathematics in HTML, Madoko can either typeset the math statically or dynamically:

- **static:** In the default static mode, Madoko uses LaTeX to generate static images for each formula. This is generally preferred because pages load fast, and it requires no javascript. The drawback is that you need to

have LaTeX installed on your system. See the [metadata section](#) for more information on fine-tuning static mathematics.

- **dynamic:** Madoko uses the [MathJax](#) JavaScript library to render the math on the client-side. This means you do not need LaTeX but it is generally quite slow for math-heavy documents and not all of the LaTeX commands and packages are available. This mode can be enabled in the metadata as:

```
Math Mode: dynamic
```

The [metadata section](#) contains more information on customizing MathJax.

You can view and experiment with the different rendering modes in the online [mathematics rendering demo](#).

4.9.4. Advanced: Generic LaTeX snippets

Besides just mathematics, you can include arbitrary LaTeX snippets using the custom **Snippet** block. In static math mode such snippets get included in the web page as hi-res images. This makes it possible to use powerful packages like [TikZ](#) and [PSTricks](#). To extract the images, Madoko uses [ImageMagick's Convert](#) program that should be installed on your system ([download](#)).

For the following example, we use PSTricks:

```
Package: pstricks
Package: pst-plot
```

to typeset a the graph of the natural logarithm⁴:

⁴Example comes from the [PSTricks example gallery](#).

```

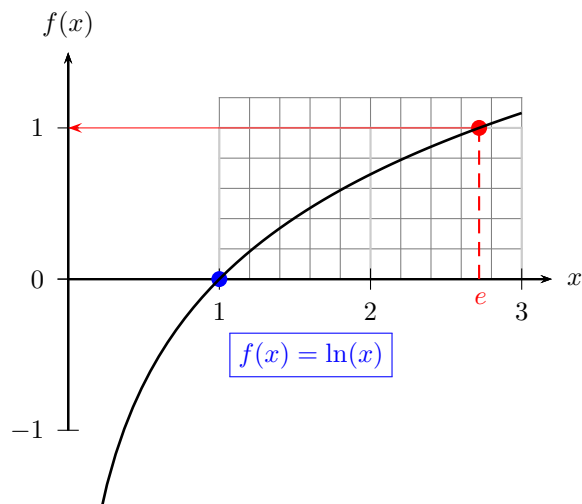
~ Center {padding=1ex}
~~ Snippet
\psset{unit=2cm}
\begin{pspicture*}(-0.5,-1.5)(4,2)
  \psgrid[subgriddiv=5,gridcolor=black!20%
    ,gridlabels=0pt](1,0)(3,1.2)
  \psaxes{->}(0,0)(0,-1)(3.2,1.5)
  \uput[0](3.2,0){$x$}\uput[90](0,1.5){$f(x)$}
  \pscircle*[linecolor=red](! Euler 1){3pt}
  \psline[linecolor=red,linestyle=dashed]%
    (! Euler 0)(! Euler 1)
  \psline[linecolor=red,linewidth=0.2pt,arrowscale=2]%
    {->}(! Euler 1)(0, 1)
  \uput[-90](! Euler 0){\color{red}$e$}

```

```

\pscircle*[linecolor=blue](1,0){3pt}
\psplot[linewidth=1pt]{0.2}{3}{ x ln }
\rput(1.6,-0.5){\color{blue}\fbox{$f(x)=\ln(x)$}}
\end{pspicture*}
~
~

```



The next two figures, Figure 2 and Figure 3 use the modern [TikZ/Pgf](#) package which we included as:

```

Package      : tikz
Tex Header   : \usetikzlibrary{decorations.pathreplacing%
                  ,decorations.pathmorphing}

```

Both examples come from the [TikZ example gallery](#).

The `pgfplots` library is an extension of TikZ/Pgf to display plots:

Package: `pgfplots`

Drawing plots can be done either using math formulas or by giving direct data points:

```

Using the `pgfplots` package we can draw nice plots:
~ Snippet
\begin{tikzpicture}
\begin{axis}[
  height=8cm,
  width=8cm,

```

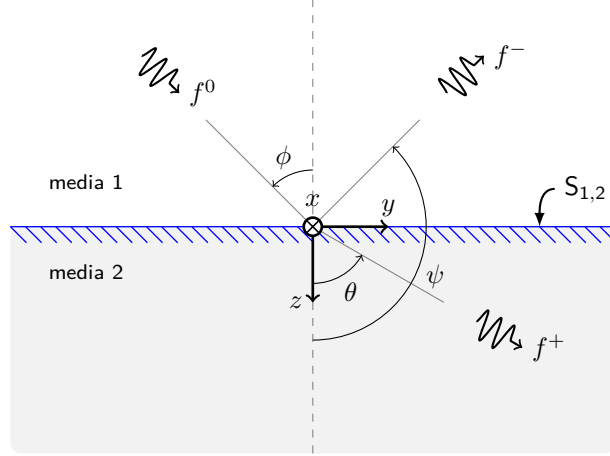


Figure 2. TikZ example by Edgar Fuentes of reflection and refraction of electromagnetic waves.

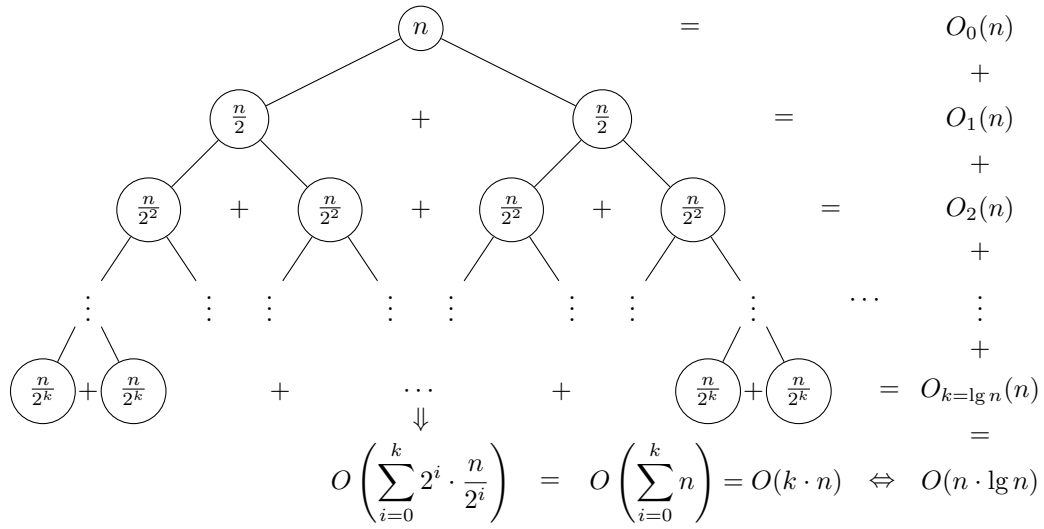


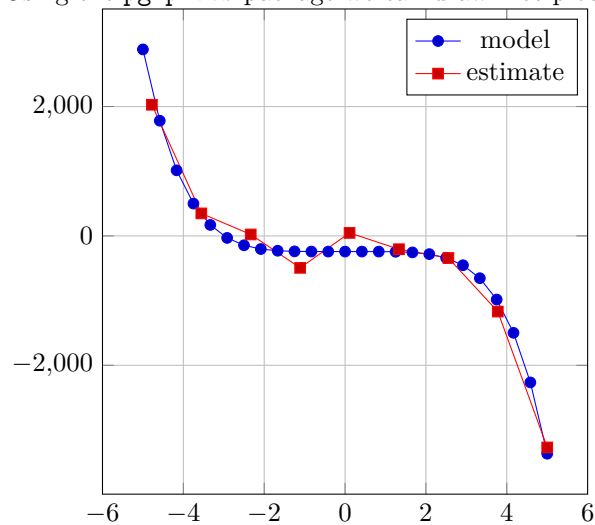
Figure 3. TikZ example by Manuel Kirsch of a merge sort recursion tree.

```

    grid=major,
]
% math plot
\addplot {-x^5 - 242};
\addlegendentry{model}
% data plot
\addplot coordinates {
(-4.77778,2027.60977)
(-3.55556,347.84069)
(-2.33333,22.58953)
(-1.11111,-493.50066)
(0.11111,46.66082)
(1.33333,-205.56286)
(2.55556,-341.40638)
(3.77778,-1169.24780)
(5.00000,-3269.56775)
};
\addlegendentry{estimate}
\end{axis}
\end{tikzpicture}
~

```

Using the `pgfplots` package we can draw nice plots:



The final examples in Figure 4 also use the `pgfplotstable` library:

Package: `pgfplotstable`

to display bar charts.

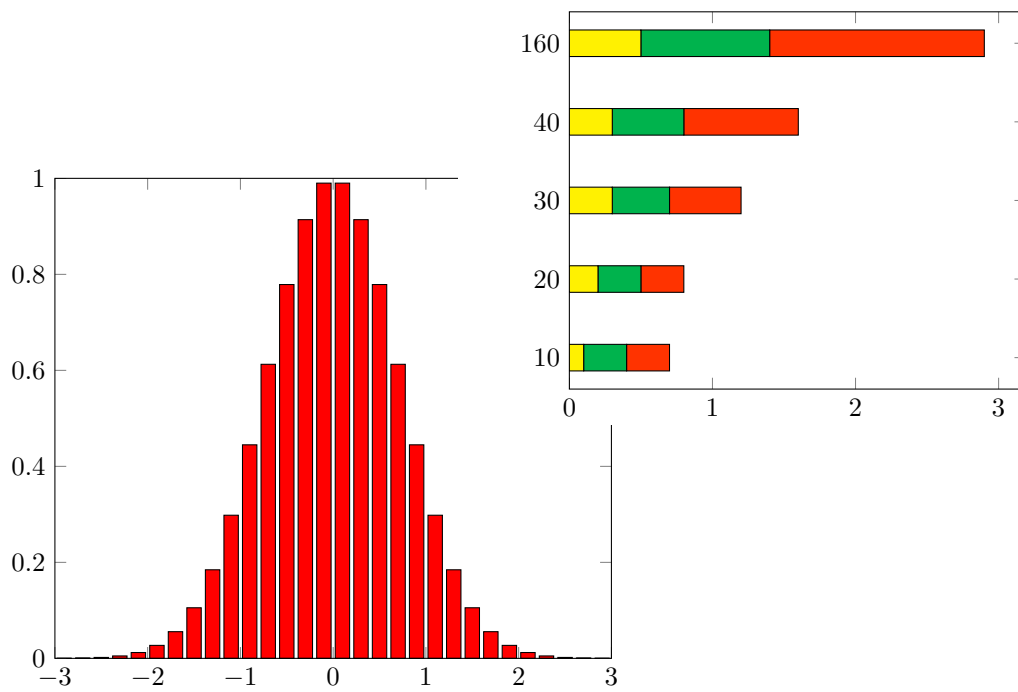


Figure 4. Bar chart examples by [Matt B.](#) and [Jake](#) using pgfplots.

4.9.5. Advanced: Efficiency of math rendering

Madoko is quite optimized to render mathematics efficiently, especially for math heavy documents which can easily contain thousands of small mathematics fragments. Madoko carefully processes only those fragments that need recompilation. If needed you can give the `-r` flag to rebuild everything from scratch.

Usually, Madoko uses plain `latex` to generate a `.dvi` file, and then uses `dvipng` to extract hi-resolution images, automatically taking care of proper baseline alignment.

However, certain snippets cannot work with plain `.dvi` and need postscript or PDF primitives, like the `TikZ` and `PSTricks` packages. In such cases, Madoko will render these snippets using `xelatex` to generate a `.pdf` file, and [ImageMagick's Convert](#) program to extract images. However, this is not done by default since this method is much (much!) slower than extracting from the `.dvi` file. The `.pdf` method is done automatically for any `Snippet`. You can force a `Snippet` to use the `dvi` method by setting `snippet-needspdf=false` in its attributes, and dually, you can force the `pdf` method for any equation by setting `math-needspdf=true`.

4.9.6. Advanced: Preformatted math

LaTeX math mode is great for regular mathematics but not so good if one tries to preserve whitespace or uses longer identifiers. This is actually quite common for computer science documents where mathematics is mixed with program code. Madoko supports a `MathPre` custom block that makes *preformatted* math much easier to typeset. In particular:

- Whitespace is preserved and spaces are replaced with a medium space (`\mathspace`) command (except when the spaces directly follow a LaTeX command). Indentation is done through a `\mathindent` command, while line breaks are automatic with a `\mathbr` command.
- A name (consisting of letters and digits) is typeset in a `\mathid` command so it will look like *function* instead of *function* (note the spacing between the *f* and *u* for example).
- If a name ends with digits, they are typeset as subscripts, where `x1` becomes x_1 .
- A name starting with an `@` character is typeset using a `\mathkw` command, where `@return` becomes `return`.
- Any text that is an argument of a `\begin`, `\end`, `\textxx` or `\mathxx` command, where `xx` is one of `tt`, `sf`, `sl`, `rm`, `it`, `kw`, `id`, `bb`, or `bf`, is kept unchanged. Also any name starting with a `#` character is kept unchanged.
- Ampersands can be used to align text.

Using this convention, we can easily typeset program code using nice symbols.

```
~ MathPre
@function sqr_\pi( num :int ) \{
  @return (num {\times} num {\times} \pi)
\}
~
```

```
function  $sqr_{\pi}(num : int)$  {
  return ( $num \times num \times \pi$ )
}
```

Here is an example of aligned text which also demonstrates the use of `replace` (see Section 5.6):

```
~ MathPre { replace="/->/\rightarrow/g" }
random &: () -> ndet double;
print  &: string -> io ();
error  &: \forall\langle\alpha\rangle string -> exn \alpha;
~
```

```
random : ()  $\rightarrow$  ndet double;
print  : string  $\rightarrow$  io ();
error  :  $\forall\langle\alpha\rangle$  string  $\rightarrow$  exn  $\alpha$ ;
```

Of course, you can often achieve a similar effect by using good replacers with `.pretty` code directly (Section 4.7.3):

```
``` koka { .pretty \
 replace="//->/\(->|\→\)\/\/</\(<|\⟨\)\/\/>/\(>|\⟩\)\/\/g" \
 replace="//\bforall\b\/\(\forall|\∀\)\/\/\balpha\b\/\(\alpha|\&\alpha;\)\/\/g" }
random : () -> ndet double;
print : string -> io ();
error : forall<alpha> string -> exn alpha;
```
```

```
random : ()  $\rightarrow$  ndet double;
print  : string  $\rightarrow$  io ();
error  :  $\forall\langle\alpha\rangle$  string  $\rightarrow$  exn  $\alpha$ ;
```

4.9.7. Setting all code to preformatted math

If you want to typeset all math using preformatted math, you can actually ‘take over’ the standard math blocks in Madoko and set the input of those to `mathpre`. For example:

```
.Math-Inline,.Math-Display: input=mathpre
```

These set both display- and inline-math to the `mathpre` input mode.

4.10. Table of contents

Generating a table of contents is easy, just include the special element `[TOC]` anywhere in your document and it will expand to a table of contents. You can use the metadata value `Toc Depth` to set how many levels deep the table of contents goes (by default 3).

You can also create a list of figures, by using `[TOC=tof]`. This will list all the `~Figure` block elements.

4.10.1. Advanced: Custom tables of contents

Custom tables of contents can be generated using the `toc` attribute. The `toc-depth` attribute specifies the depth of the element (by default 1), while the `toc-line` specifies the contents of the line displayed in the table of contents. For example, using [metadata rules](#) we can generate a table of contents for equations:

```
~Equation: toc=equations toc-line="&caption;"
```

Here we assume that the user adds a `caption` attribute when denoting equations (which will get expanded inside the `toc-line`). We can then render the table of equations anywhere in the document as:

```
[TOC=equations]
```

4.11. Bibliography and Citations

One of Madoko’s main design goals is to enable the creation of high-quality scholarly articles. As such, Madoko integrates closely with the standard [BibTeX](#) tool to generate bibliographies and references inside a document. Usually, BibTeX comes standard with any LaTeX installation.

You can simply use any existing BibTeX bibliography file (`.bib`). You can specify which bibliography files are to be used using `Bib metadata` entries:

```
Bib: ../mybib1
Bib: mybib2
```

As an example, you can view the bibliography for this document [here](#). Entries in the bibliography files can now be referenced using semi-colon separated references (as in [Pandoc](#)), for example

Read about LaTeX and TeX [`@Knuth:Tex`; `@Lamport:Latex`].

Read about LaTeX and TeX [4, 5].

Note that unlike LaTeX there is no need to explicitly insert an unbreakable space between the text and the citation, Madoko automatically takes care of this (as described in Section 4.2). If necessary, you can also include extra text for each entry:

Read more [The book `@Knuth:Tex`; \ `@Lamport:Latex` (chapter 4)].

Read more [The book 4, 5 (chapter 4)].

When Madoko finds such references, it writes them to a `.bib.aux` file (together with the needed bibliography files) that are read by BibTeX to generate the bibliography entries. BibTeX is called automatically by Madoko whenever the citations change. The generated bibliography entries are included in your document using the special `[BIB]` element, for example:

```
## References {-}
[BIB]
```

which you can see at the [end of this document](#).

4.11.1. Bibliography styles

The style of the bibliography entries is determined using the `Bib Style` key and can be any [BibTeX style](#) (by default `plainnat`):

Bib Style: `abbrvnat`

Madoko uses a very simple LaTeX parser to format the bibliography entries in Markdown. It can handle things like special characters and accents quite well and recognizes many formatting commands. Even though it is sufficient for bibliography entries in general, the Madoko LaTeX parser may not work for more fancy LaTeX commands in bibliography entries. However, we strive to make it work for any bibliography style and entries, so please [file a bug report](#) if you encounter situations where it does not work correctly.

Bibliography styles tested with Madoko include the following author-year styles: `apa`, `apalike`, `plainnat`, `abbrvnat`, `unsrtnat`, `newapa`, `chicago`, `named`, `agsm`, `dcu`, `kluwer`, `astron`, `bbs`, `cbe`, `humannat`, `humanbio`, `jtb`, `apsrev4-1`, `aipauth4-1` and others. Also, the following numeric styles have been tested: `eptcs`, `abbrv`, `plain`, `ieeetr`, `acm`, `unsrt`, `alpha`, `siam`, `apsrmp4-1`, `aipnum4-1` and others. Since tools like BibTeX and Madoko make numbering and linking automatic, it is generally preferred for modern documents to use a numeric citations style.

4.11.2. Citation styles

The citation style defaults to a *numeric* style. However, it can be set explicitly using the `Cite Style` metadata key:

`Cite Style: natural`

Valid citation styles are *natural*, *textual*, *super*, and *numeric* (default). The natural and textual style use author-year style citations, while the super and numeric styles use numbers.

| | | |
|----------------|------------------------------|-------------------------------------|
| <i>natural</i> | (Lamport, 1994; Knuth, 1984) | (default for author-year citations) |
| <i>textual</i> | Lamport (1994); Knuth (1984) | |
| <i>numeric</i> | [4, 5] | (default for numeric citations) |
| <i>super</i> | ^{4,5} | |

Note that numeric citations are sorted (and compressed) by default. Also full author-year style citations only work with BibTeX styles that support this, i.e. generally any style that works with the `natbib` package like `plainnat`. With author-year citations we can use modifiers to change how the citation is shown. For example, assuming a *natural* style:

| | | |
|----------------------|---|--------------------|
| <code>@Goo93</code> | (Goossens et al., 1993) | Natural |
| <code>+@Goo93</code> | (Goossens, Mittelbach, and Samarin, 1993) | Long – all authors |
| <code>-@Goo93</code> | (1993) | Short – just year |

Moreover, if you leave out the brackets, you force a *textual* style:

| | | |
|----------------------|--|--------------------|
| <code>@Goo93</code> | Goossens et al. (1993) | Textual style |
| <code>+@Goo93</code> | Goossens, Mittelbach, and Samarin (1993) | Long – all authors |
| <code>-@Goo93</code> | 1993 | Short – just year |
| <code>!@Goo93</code> | Goossens et al. | Just authors |

In a numeric style, most of these attributes have no effect and all the bracketed cases are displayed as “[2]”. For the four cases in textual style, we get “Goossens et al. [2]”, “Goossens, Mittelbach, and Samarin [2]”, “[2]”, and “Goossens et al.”.

See Appendix A.3 on how to customize citations styles further, and Appendix A.4 on how to write your bibliography entries by hand without using BibTeX.

4.11.3. Bibliography tooltips and searches

By default, the HTML backend shows a tooltip when hovering over a citation (try it [2]). This functionality is by default disabled in the LaTeX backend. However, you can enable tooltips in PDF too by setting the `.tex-tooltip` class through a metadata rule:


`~a: .tex-tooltip`

This will display a small yellow text balloon in the PDF file that shows a tooltip when hovering above it:

Safe state encapsulation using
by Launchbury and Sabry [5]. The

| | |
|---|-------|
| Title | are i |
| John Launchbury and Amr Sabry. Monadic state: | ate |
| Axiomatization and type safety. | but |

it is hard to state the stateful le,

Moreover, Madoko adds to each bibliography entry a magnifying glass icon () that links to a web search for that reference. You can customize the search engine that is used by setting the `Bib Search Url` metadata key:

`Bib Search Url: www.google.com`

If you set the `Bib Search Url` to `false` (or empty), Madoko will disable search icons. Again, this functionality is disabled by default in the LaTeX backend unless you set the `Bib Search Url` key explicitly.

4.12. Custom blocks

Madoko custom blocks are similar to the `div` element in HTML and allow the use of custom block elements that can be styled and processed in a particular way. A custom block starts on new line starting with one or more tildes (`~`) optionally followed by the block name and attributes. It ends at *the first line containing the same number of tildes* that started this block.

```
~ Note
Here is a note.
~
~~ { font-style=italic }
And some italic text in an unnamed block.
~~
```

Note. Here is a note.
And some italic text in an unnamed block.

Note that blocks with the same number of tildes do not nest, e.g. the following is wrong:

```
~ Note
~ Equation
e = mc^2
~
~
```

since the `Note` will end at the first lonely tilde, not the second one. As an aside, [git flavored markdown](#) uses three or more tildes for fenced code blocks. Since Madoko uses tildes for custom code blocks this cannot be used and Madoko only supports the more popular back-ticks (`````) for fenced code blocks.

Custom blocks work especially well with metadata rules (see [Section 5.4](#)) where we can define attributes that get applied to every occurrence of a custom block. For example, we could define the metadata rule:

`~Slanted: font-style=oblique`

and then every occurrence of a `Slanted` custom block would be typeset in a slanted font.

```
~ Slanted
Here is my slanted custom block
~
```

Here is my slanted custom block

The “number-of-tildes” rule to delimit custom blocks is convenient and works fine when nesting a small number of custom blocks, but for long blocks or deep nesting, this can easily lead to confusion. To alleviate this, a custom block can also start with one or more tildes, followed by `Begin blockname`. Such block continues until a corresponding number of tildes is found followed by `End blockname`. It is recommended to use this form of blocks for example in [metadata rules](#)

```
~ Begin Slanted
~ Begin Note
Here is a slanted note.
~ End Note
~ End Slanted
```

Note. *Here is a slanted note.*

4.12.1. Predefined custom blocks

Madoko defines quite a few common custom blocks. Their exact definitions can be found in [Appendix A.12](#).

- **Figure:** This is used to define figures (see [Section 4.3](#)). Recognizes the following attributes:
 - `caption=caption`: specify the caption of a figure.
 - `.wide`: if the class `wide` is set, the figure will span the width of a page in a two-column format (used in LaTeX).
 - `page-align=(top|bottom|page|here|forcehere)`: specify the alignment of the figure in a page (used for LaTeX output).
- **Equation:** Specify mathematical equations. See [Section 4.9](#) for its usage.
- **Snippet:** Generic LaTeX snippets, see [Section 4.9.4](#).
- **Bibitem, Bibliography:** Used for writing bibliography entries by hand. See [Section 4.11](#) for more information.

- **Note, Remark, Proof:** Used for notes, remarks, and proofs.
- **Abstract:** Defines the abstract of an article.
- **Framed:** A block with a solid border. Use the `tight` attribute to suppress a paragraph block around its content.
- **Center:** A block that centers its contents horizontally on the page.
- **Columns, Column:** Put `Column` blocks inside a `Columns` block, to typeset block elements next to each other on a page. Use the `width` attribute to control the width of each column.
- **TexRaw:** A block for raw TeX content that is passed directly to LaTeX.
- **Section:** Starts an explicit section. In the HTML5 backend expands to a `<section>` element. This is mainly used inside presentations using `reveal.js`.
- **Article, Aside, Nav:** Expand to the corresponding HTML5 elements.
- **HtmlRaw:** A block with raw HTML content that is pasted directly into the HTML output.
- **TexOnly:** A markdown block that is only processed for LaTeX output (and not shown in HTML output).
- **HtmlOnly:** A markdown block that is only processed for Html output (and not shown in LaTeX output).
- **MathPre:** A block with preformatted math (Section 4.9.6).
- **MathDefs:** A block with LaTeX math definitions (Section 4.9.2).
- **Theorem, Lemma, Proposition, Corollary, Example, Definition:** Each of these blocks is individually numbered and starts with the block name in bold. For example:

```

~ Lemma      {#LeftCosetsDisjoint}
Let  $H$  be a subgroup of a group  $G$ , and let  $x$  and
 $y$  be elements of  $G$ . Suppose that  $xH \cap yH$  is
non-empty. Then  $xH = yH$ .
~
~ Proof
Let  $z$  be some element of  $xH \cap yH$ . Then  $z = xa$ 
for some  $a \in H$ , and  $z = yb$  for some  $b \in H$ .
If  $h$  is any element of  $H$  then  $ah \in H$  and
 $a^{-1}h \in H$ , since  $H$  is a subgroup of  $G$ .
But  $zh = x(ah)$  and  $xh = z(a^{-1}h)$  for all
 $h \in H$ .

```

Therefore $zH \subset xH$ and $xH \subset zH$, and thus $xH = zH$. Similarly $yH = zH$, and thus $xH = yH$, as required. This concludes the proof of Lemma [LeftCosetsDisjoint]. ~

Lemma 1. Let H be a subgroup of a group G , and let x and y be elements of G . Suppose that $xH \cap yH$ is non-empty. Then $xH = yH$.
Proof. Let z be some element of $xH \cap yH$. Then $z = xa$ for some $a \in H$, and $z = yb$ for some $b \in H$. If h is any element of H then $ah \in H$ and $a^{-1}h \in H$, since H is a subgroup of G . But $zh = x(ah)$ and $xh = z(a^{-1}h)$ for all $h \in H$. Therefore $zH \subset xH$ and $xH \subset zH$, and thus $xH = zH$. Similarly $yH = zH$, and thus $xH = yH$, as required. This concludes the proof of Lemma 1. □

Of course, each of the predefined blocks can be customized further using attributes and rules. For example, by including the following metadata rule, we can typeset proofs with *Proof* in an italic style instead of bold:

```
~Proof: before="[_Proof_. ]{.proof-before}"
```

4.13. Special block elements

Madoko supports some special block elements that get expanded automatically.

- [TITLE]: Expands to a title element generated from the [metadata](#) keys *title*, *subtitle*, *title date* and author info (*author*, *affiliation*, and *email*). In the LaTeX backend this will invoke the `\maketitle` command.
- [TOC], [TOC=*name*]: Expands to a table of contents. See also Section 4.10.
- [FOOTNOTES]: Expands to the defined footnotes. If there are footnotes and this element is not present, the footnotes are automatically included at the end of the document. In the LaTeX backend, footnotes always appear at page footers.
- [INCLUDE=*file*]: Expands to the contents of *file*. Include elements are processed before any other processing happens and can for example be used to include meta data elements.

```
[INCLUDE="presentation.mdk"]
```

Include files are first searched relative to the current directory, then the directory of the input file, followed by the output directory, and finally the `styles` directory in the installation directory of Madoko.

The INCLUDE element can also be used to include parts of a file, see Section 4.13.1 for more information.

- [BIB]: Includes the bibliography entry file (.bbl) generated by BibTeX. Internally expands to:

```
~ Begin TeX
[INCLUDE="myfile.bbl"]
~ End TeX
```

See also Section 4.11.

4.13.1. Advanced: including file fragments

The full syntax of the INCLUDE element is:

```
[INCLUDE(=file)?(:range)?]
```

where the *range* is either a *fragment name* or line range. For example we can include part of file by using:

```
[INCLUDE=foo.hs:10-18]
```

or

```
[INCLUDE=foo.hs:20]
```

which would include lines 10 to 18 and just line 20 from `foo.hs`. Of course remembering line ranges can be error prone, so we can also give names to fragments of files and use those to refer to them. For example, we may have code samples in `foo.hs` as:

```
-- BEGIN:Var
-- BEGIN:Syntax
data Exp = Number      Int
         | Add          Exp Exp
         | Subtract     Exp Exp
         | Multiply     Exp Exp
         | Divide       Exp Exp
-- END:Syntax
         | Variable     String;
-- END:Var

--BEGIN:Eval
evaluate :: Exp -> Int
evaluate (Number i)      = i
evaluate (Add a b)       = evaluate a + evaluate b
evaluate (Subtract a b)  = evaluate a - evaluate b
evaluate (Multiply a b)  = evaluate a * evaluate b
evaluate (Divide a b)    = evaluate a `div` evaluate b
--END:Eval
```

Where the `BEGIN name` and `END name` delimit file fragments. Note that the fragments can be nested. We can now include particular fragments as:

```
[INCLUDE=foo.hs:Var]
[INCLUDE=foo.hs:Eval]
```

for example. Sometimes, it can be a hassle to always mention the full file name. In that case, we can first do a few empty includes of all the needed files (so Madoko knows the fragment names), and then just use the fragment names directly, e.g. first an empty include at the start of the document by using line number 0:

```
[INCLUDE=foo.hs:0]
```

and later in the document, we can refer to fragments in that file using:

```
[INCLUDE:Syntax]
```

The delimiters for fragments are always ‘*start* BEGIN|END *name*’ where *start* includes most of the usual comment characters, namely, //, --, #, and %. If you need some other sequence, you can customize this using the **Fragment Start** and **Fragment End** metadata keys. For example, the default is set as:

```
Fragment Start: ^(?:\\\/|--|[#%]) *BEGIN *: *(\w+) *$
Fragment End   : ^(?:\\\/|--|[#%]) *END  *(?:[:] *(\w+) *)? $
```

5. Syntax: Metadata and Styling

5.1. Metadata

Similar to [multimarkdown](#), a document can begin with a special metadata section that contains meta information like the document title, the author, etc. Moreover, this section can contain attribute rules to globally apply attributes to certain elements, much like CSS rules.

Metadata must come immediately as the first thing in a document, and consists of keys followed by a colon and then the key value. A key value can span multiple lines by indenting, and you can leave blank lines between different keys.

```
Title       : An overview of Madoko
Author      : Daan Leijen
Affiliation  : Microsoft Research
Email       : daan@microsoft.com
Heading Base: 2
MathJax     : True
```

5.1.1. Special metadata keys

Any metadata key and value can be given (and referred to using [entity names](#)), but certain keys have special meaning to Madoko.

- **Title:** The title of the document. For example, in HTML output it determines the <title> element. It is also used by the special [TITLE]

element (Section 4.13).

- **Subtitle:** An optional subtitle.
- **Title Note:** A general note about the document. Used in the [TITLE] element where it is put below the title and authors.

Title Note: Draft, &date; (version 1.0)

- **Author, Address, Affiliation, Email, Author Note:** the author name, address, affiliation, email, and general note. There can be multiple authors. The author info is also used by the [TITLE] element to generate a proper document title header. The authors are also written to the author <meta> tag. In the case of multiple authors, the address, affiliation, author note, and email always belong to the last defined author. If you define multiple **Address** or **Affiliation** entries for one author, they are placed above each other.
- **Toc depth (=3):** The maximum depth of headings that are included in the table of contents (Section 4.10).
- **Heading depth (=3):** The maximum depth of headings that are numbered (Section 5.5). Set it to zero to suppress numbering of headings completely.
- **Heading base (=1):** Usually, a top heading (# heading) maps to a <h1> or \chapter element in HTML and LaTeX respectively. By increasing the **Heading Base** you can change this mapping. For example, by using:

Heading Base: 2

the top headings will map to <h2> or \section elements instead.

- **Section Depth (=0):** Maximal heading level where <section> elements are wrapped around the content. The default value ensures no section wrapping takes place.
- **Section Base (=1):** All headings at **Section Base** and lower are treated as equal for the purpose of wrapping <section> elements around the content. This is especially useful for presentations where we would like to start slides equally for headings at level 1 and 2 (see Appendix A.2).
- **Bib or Bibliography:** Specify a bibliography (.bib) file to be used by the BibTeX tool to generate a list of references (see Section 4.11).

Bib: ../mybibliography.bib

- **Bib Style or Biblio Style (=plain):** Specify a BibTeX style that is used to format the list of references. See Section 4.11.1 for more information.

Bib Style: plainnat

- **Cite Style (=numeric):** Specify the citation style used for citations, *natural*, *_textual*, *super*, or *numeric*. See Section 4.11.2 for more information.

- **Cite All** (=false): Set to **True** to cite everything in the included bibliography files.
- **BibTex** (=bibtex): The command to run the BibTeX tool. Set it to **false** to suppress running BibTeX (for example, where writing bibliography entries by hand (see Section A.4))
- **Pdf Latex** (=xelatex): The command used to run the LaTeX program when the `--pdf` command line flag is present.
- **Fragment Start**, **Fragment End**: Used to specify file fragment delimiters, see Section 4.13.1.

The following keys are not used as this moment but have a standard meaning:

- **Copyright**: Copyright information. Written to the copyright `<meta>` tag.
- **License**: License for this document. Written to the license `<meta>` tag.
- **Keywords**: A list of document keywords. Written to the keywords `<meta>` tag.
- **Description**: A short description of the document. Written to the description `<meta>` tag.
- **Comment**: General comments.
- **Revision**: Revision of the document.
- **Phone**: Phone number of an author.

5.1.2. HTML keys

Some keys are only interpreted for HTML output:

- **Css**: Specify a CSS file that needs to be included in HTML output. There can be many CSS keys present.

```
Css: lib/main.css
```

```
Css: http://foo.com/bar.css
```

```
Css: http://fonts.googleapis.com/css?family=Open+Sans
```

Use **Css: clear** to clear the list of CSS files. This can be used for example to not include the default `madoko.css` style file.

madoko.net: When not giving an extension or full path, Madoko will try to load the css file from the Madoko server's `/style` directory. This is used for the standard `madoko.css` file.

- **Script**: Specify a Javascript file that needs to be included in the HTML output via a `<script>` tag. There can be many script keys present.

```
Script: lib/main.js
```

madoko.net: Usually, scripts are *not* loaded in the preview as this may lead to performance problems. You can use the `preview` class to signify that the script should be loaded in the preview too:

```
Script: reveal.js {.preview}
```

- **HTML Meta:** Specify content included in a `<meta>` tag. There can be many meta tags specified.

HTML Meta: `http-equiv="refresh" content="30"`

Use the value `clear` to clear the list of meta tags. This can be used for example to exclude the default `viewport` meta tag that is emitted for proper viewing on mobile devices.

- **HTML Header:** The value is included literally in the `<head>` section of the HTML document.
- **Math Mode (`=static`):** The mathematics rendering mode: `static` or `dynamic`. In static mode, you can use the following keys to fine-tune the rendering:

- **Math Dir:** *dir* (`=math`): The relative directory where Madoko stores the images for the math formulas. All images can also be embedded in the web page if you set **Math Embed** high enough.
- **Math Dpi:** *dpi* (`=300`): The resolution at which the images are rendered. The default is a fairly high resolution. Decrease it to make the generated images smaller. Use the `-vv` flag to see the sizes of all the images generated for formulas.
- **Math Scale:** *scale* (`=108`): Scaling in percentage used for all math. The default looks quite good for most fonts, but you may want to change it depending on the main font used.
- **Math Embed:** *size* (`=1`): Limit in kilobytes at which images are embedded in the HTML page instead of separate `.png` images. Set it to 0 to never embed an image, and high enough to always embed.
- **Latex:** *cmd* (`=latex`): The command used to invoke LaTeX when rendering math to a `.dvi` file.
- **Dvipng:** *cmd* (`=dvipng`): The command used to generate `.png` images from the `.dvi` file generated by LaTeX.
- **Math Pdf Latex:** *cmd* (`=xelatex`): The command used to invoke LaTeX when rendering math that requires a `.pdf`.
- **Convert:** *cmd* (`=convert`): Specify the `convert` command that is executed for images that are generated from `.pdf` output (like PSTricks, or TikZ).

The following keys are used when math mode is dynamic:

- **MathJax** (`=false`): Set this key to the path of your MathJax script. Set this key to `True` to include the standard secure script to the latest MathJax installation on the [CDN network](#).

MathJax: `True`

- **MathJax Ext**: requires that the **MathJax** key is set. Adds a MathJax extension to the loaded scripts. See Section A.5 for more information.

5.1.3. LaTeX keys

For LaTeX output, the following keys are relevant:

- **Document Class** or **Doc Class** (=book): Specify the LaTeX document class. Can be prefixed with its options using square brackets:

Document Class: `[9pt]article`

madoko.net: When leaving out the extension, Madoko assumes this is a standard LaTeX document class (available from CTAN). If an extension is given, the file is loaded (or created) from the current directory.

- **Package**: Specify a LaTeX package that is included via `\usepackage`. The package name can be prefixed with its options using square brackets.

Package: `fancyhdr`

Package: `[colorlinks=true]hyperref`

If the package name ends with `.tex` and has no options, it will be included using the `\input` command instead. You can clear the package list using **Package:** `clear`. This can be used to not include the default `madoko.sty` package.

Sometimes, a package does not work when typesetting mathematics in plain LaTeX. This can happen for example if the rest of the document needs a package that only loads in some other latex engine, like XeLaTeX. In that case, you can exclude the package when typesetting basic math by using the star option:

Package*: `pgfplots`

madoko.net: When leaving out the extension, Madoko assumes this is a standard LaTeX package (available from CTAN). If an extension is given, the file is loaded (or created) from the current directory.

- **Tex Header**: The value is included as is before the `\begin{document}` command of the LaTeX output.
- **Tex Header***: The value is included as is before the `\begin{document}` command of the LaTeX output but excluded when rendering basic mathematics.

5.2. Entities

Madoko uses entities extensively. If Madoko finds an entity name (*&name;*) it looks up the name in various places: if there is a block element with that *id*, the entity name is replaced by the *label* value of that block (see Section 4.2). If there is no block with that identity, Madoko looks if there is a metadata value with that name, and replaces the entity name with its value:

```
The title of this document is "&title;".\
And this section has label &sec-entity;.\
Standard entities are looked up last &Delta;&hArr;&delta;.
```

```
The title of this document is "Madoko Reference".
And this section has label 5.2.
Standard entities are looked up last  $\Delta \Leftrightarrow \delta$ .
```

Entity names for labels and metadata values, can consist of letters, underscores, minus signs, and digits. In contrast to regular HTML entity names, Madoko compares the entity name for labels and metadata in a case-insensitive way.

Note that if label or metadata values contain themselves entities, these are expanded recursively (up to a certain limit) which can provide a powerful abstraction mechanism. For example, if we define a metadata value `Title2`:

```
Title2: (&Title;,&Title;)
```

then the entity `&Title2;` expands to “(Madoko Reference,Madoko Reference)”.

If entity names are used inside `attributes`, Madoko first looks for the name as one of the attribute key values. For example, you can use names like `&id;`, `&class;`, and `&label;`.

Some predefined metadata keys are quite useful as entities:

- `&date;`. The current (compilation) date in [ISO 8601](#) international format.
- `&time;`. The current (compilation) time in [ISO 8601](#) international format.
- `&madoko-version`. The version of the Madoko compiler.
- `&docname;`. The file name of the document without extension and directory.
- `&filename;`. The full file name of the document.

There are also a few special entity names:

- `&nl;`. Expands to the newline character.
- `&br;`. Expands to a forced line break (i.e. `
`).
- `&>null;`. Expands to an empty string.
- `&source;`. Inside attributes, expands to the literal content of the element.
- `&&`. Expands to the literal `&` character. This is necessary sometimes since entity expansion is done recursively without regard for other formatting attributes. For example, if you have a metadata value `Foo` that needs to expand to ``&``, you would need to write it as:

Foo: `&&`

or otherwise the `&` part would get expanded even within the back ticks. The reason why Madoko expands regardless of other formatting is to allow powerful abstraction where we can for example build up regular expressions from smaller parts.

For some examples of the usage of these elements, see Sections 5.6 and 5.5.2.

5.3. Attributes and Styling

Writing Madoko documents is clearly about content: the main advantage of using a markdown format is that it allows the writer to concentrate on prose and content instead of formatting. However, in the final stages it is desirable to *style* a document to make it look good. This section describes some tricks and tips that can help doing this well.

5.3.1. Attributes

An essential addition of Madoko is good support for attributes. The syntax is similar to attribute definitions in [Kramdown](#), [Pandoc](#), and [MultiMarkdown](#), and denoted between curly braces:

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

For most block elements, like lists, paragraphs, block quotes, etc, you can write attributes on the line directly following the block. For list items, the attributes directly follow the item:

```
This is a paragraph in small-caps.
{ font-variant=small-caps}
```

```
* {color=navy} This is a 'mylist'
* in italics
{ .mylist font-style="italic" }
```

```
THIS IS A PARAGRAPH IN SMALL-CAPS.
```


- *This is a 'mylist'*
- *in italics*

For fenced code blocks, custom blocks, and headers, the attributes are specified on the first line.

For inline elements, you can follow links, images, code, and bracketed text (between [and]) with attributes. Bracketed text with attributes become a `` in the HTML backend. It is of course recommended to put attributes when possible in link or image definitions themselves:

```
Here is an ![butterfly] image.
With [bold]{font-weight=bold} text, and even the
T[E]{vertical-align=-0.5ex margin-left=-0.25ex \
      margin-right=-0.25ex}X
logo.
```

```
[butterfly]: butterfly.png { width=100px vertical-align=top }
```

Here is an  image. With **bold** text, and even the $\mathrm{T_E}X$ logo.

Note that inside attributes, and also metadata values, a backslash followed by a newline acts as a line joiner and removes the newline and the following whitespace before processing the attributes. This is convenient when defining long strings inside attributes. For example, if a figure has a long caption, we could write it as:

```
~ Figure { #myfigure \
           caption="Here is a really \
                   long caption." }
...
~
```

To get a literal line break inside an attribute string, use the entity `&nl`; instead.

5.3.2. CSS formatting attributes

Attributes that are not special to Madoko (as described in Section A.8), are passed as CSS style attributes in the HTML backend. For the LaTeX backend, Madoko includes special style files (`css.sty` and `madoko.sty`) that process many CSS formatting commands (and ignores all others). Currently formatting commands that are recognized are:

- `display=(block|inline|block-inline|hidden)`
- `margin=(auto|length)`
- `margin-left,margin-right,margin-bottom,margin-top`
- `padding=(auto|length)`
- `padding-left,padding-right,padding-bottom,padding-top`
- `width=length`
- `height=length`
- `vertical-align=(top|middle|bottom|baseline|length)`
- `border-style=(none|solid|dotted)`
- `border-left-style, border-right-style.`
- `border-top-style, border-bottom-style.`
- `border-width=length`

- `border-color=`*color*
- `text-align=(center|right|left|justify)`
- `text-indent=length`
- `line-height=length`
- `font-style=(italic|oblique|normal)`
- `font-variant=(small-caps|normal)`
- `font-weight=(bold|normal)`
- `font-size=(xx-small|x-small|small|medium|large|x-large|xx-large)`
- `font-family=(monospace|serif|sans-serif|normal|family)`
- `color=`*color*
- `background-color=`*color*.
- `penalty=`*number*
- `page-align=(top|bottom|here|forcehere)`. Used for Figure custom blocks (see Section 4.12.1).
- `float=(left|right)`. Limited support at this time in LaTeX.

Note that the syntax at this point, the syntax is quite a bit more restrictive than plain CSS. For example, we cannot just say:

```
{ border="solid 1px" }
```

as in CSS, but need to state each property separately as:

```
{ border-style=solid border-width=1px }
```

Also, *lengths* must use common units, such as 10ex, 5em, 4px, 2cm, or -6pt for example. Percentages only work for width's.

A font family can be a comma separated list of fonts. However, in LaTeX only the last font in the list is used. If it is not one of the standard font families (`serif`, `sans-serif`, `monospace`), it is selected using the `\fontspec` command.

```
~ { font-family=Georgia }
This is in the Georgia font.
~
```

This is in the Georgia font.

A *color* must be either a named color or use the HTML `#rrggbb` format. The recognized named colors are the basic CSS colors shown in Figure 5.

If you would like to use another named color that is not one of the above, you can often just define an entity in the metadata:

Indigo: `#4B0082`

and use Madoko's expansion to use it inside an attribute:

```
This is "[Indigo]{color=&Indigo;}".
```


| Name | Hex value | Color |
|------------------|-----------|---|
| Red | #FF0000 |  |
| Lime | #00FF00 |  |
| Blue | #0000FF |  |
| Yellow | #FFFF00 |  |
| Cyan, Aqua | #00FFFF |  |
| Magenta, Fuchsia | #FF00FF |  |
| Maroon | #800000 |  |
| Green | #008000 |  |
| Navy | #000080 |  |
| Olive | #808000 |  |
| Teal | #008080 |  |
| Purple | #800080 |  |
| Orange | #FFA500 |  |
| Black | #000000 |  |
| DimGray | #696969 |  |
| Gray | #808080 |  |
| DarkGray | #A9A9A9 |  |
| Silver | #C0C0C0 |  |
| LightGray | #D3D3D3 |  |
| White | #FFFFFF |  |
| Gainsboro | #DCDCDC |  |
| FloralWhite | #FFFAF0 |  |
| Ivory | #FFFFF0 |  |

Figure 5. The basic CSS colors

This is “Indigo”.

In general, it is hard to emulate a complex CSS layout in LaTeX so some particular combinations may not work as expected in LaTeX. We strive to make it work seamlessly though so please [report any issues](#). Nevertheless, the above formatting commands should suffice in most situations and we can program already many fancy examples:

```
[**Aaaa[a]{vertical-align=-0.3ex}
[aa]{vertical-align=-0.7ex}
[r]{vertical-align=-1.4ex}
[g]{vertical-align=-2.5ex}
[h]{vertical-align=-5ex}**]{font-size=large height=0pt }
he shouted but not even the next
one in line noticed that something
terrible had happened to him.
{width=18em border-style=solid border-width=1px \
padding=1ex background-color=FloralWhite}
```

Aaaaa aa r he shouted but not
 even the next g one in line noticed that
 something terrible h had happened to him.

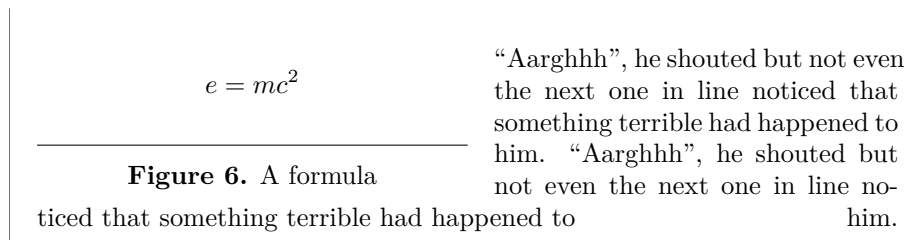
5.3.3. Float

There is limited support for the `float` attribute but it is generally quite hard to emulate this well in LaTeX. However, one can use the `float` attribute to put figures or general custom blocks on the left- or right-side of a page and have text flow around it. For example:

```

~ Figure { caption="A formula" width=50% float=left margin-right=1em}
~~ Math
e = mc^2
~~
~
"Aarghhh", he shouted but not even the next
one in line noticed that something
terrible had happened to him.
"Aarghhh", he shouted but not even the next
one in line noticed that something
terrible had happened to [him.]{float=right}

```



Note that it is important to give a specific width when specifying `float`. Also, in LaTeX, the height of the text flowing around the float is not always correct (especially if the text contains mathematics). To fine-tune the appearance in LaTeX, one can use the `lines` key to give the height of the float figure in text lines.

Still the LaTeX command is fragile and generally works best when immediately followed by a paragraph. Also, when using `float` on inline elements, it is implemented in LaTeX simply using `\hfill` which is hopefully enough for most common situations.

5.4. Metadata rules

Often, we need to apply specific attributes and formatting to certain elements or custom blocks. This can be done using *metadata rules*. These rules function much like regular CSS rules except that they are quite a bit more limited. A rule starts with a tilde (~), dot (.), or hash character (#) for matching a custom block, class name, or id respectively. The value of a rule are attributes that get applied to any matching block. Here are some examples:

```
~Blockquote : font-style=oblique
.bold       : font-weight=bold
#myblock    : border-width=1px border-style=solid
```

This would typeset a block quotes in an oblique font, any block with a `bold` class in bold, and the block with the `myblock` id with a solid border. You can also use a comma separated list of matches that apply to all. For example, we can give all code, inline or as a block, the class `prettyprint` as:

```
~Pre,~Code: .prettyprint
```

5.4.1. Names of predefined elements

Most standard Madoko block elements, like lists or block quotes, can also be targeted by rules. Their element names are the standard HTML names, namely:

| Element | Name |
|----------------------|---------------------------|
| Paragraph | <code>p</code> |
| Code block | <code>pre</code> |
| Code inline | <code>code</code> |
| Code escaped text | <code>code-escaped</code> |
| Block quote | <code>blockquote</code> |
| List | <code>ul</code> |
| Numbered list | <code>ol</code> |
| List item | <code>li</code> |
| Definition list | <code>dl</code> |
| Definition term | <code>dt</code> |
| Definition | <code>dd</code> |
| Horizontal rule | <code>hr</code> |
| Table | <code>table</code> |
| Heading | <code>h1,...,h6</code> |
| Unnamed custom block | <code>div</code> |
| Named custom block | <code>name</code> |

Note that every custom block *blockname* automatically also gets the class *blockname* and is thus matched by both *~blockname* and *.blockname* rules. This can be also be useful when applying styles in CSS rules.

5.4.2. Advanced: Styling in CSS

Sometimes we only want to apply certain styling to just HTML. For HTML, styling through classes works best: we can simply write some CSS (and include it through the `Css` metadata or directly within `<style>` tags.). For example,

```
<style>
.slanted { font-style: oblique; font-family: Cambria }
</style>
```

For styling in LaTeX, or when certain CSS keys are not yet processed in LaTeX, see Appendix A.7.

5.5. Numbering

For larger documents, numbering sections, figures, tables, images, etc. is quite important and Madoko supports this well. By default, Madoko will number headers, figures, and equations. Numbering for headers is by default up to 3 levels, but it can be set for the document using the `Heading depth` meta variable. If it is set to zero, it suppresses numbering for headings completely:

```
Heading Depth: 0
```

5.5.1. Advanced: Custom numbering

The numbering can be completely customized though. Counters are introduced using `@name` syntax in attributes. When such counter name occurs, it is au-

tomatically incremented. If the value of a *label* contains a counter name, it is replaced by its current value. For example, the default attributes for equations contain:

```
{ @equation label="(@equation)" }
```

This automatically will increment the `@equation` counter at each occurrence of an equation block element, and set its label to the current value surrounded by parenthesis. Similarly, the default label for main sections is defined as:

```
{ @h1 label="@h1" }
```

Actually, Madoko automatically increments counters with the same name as the block, so in the previous two cases, we should leave out `@equation` and `@h1` or we do double increments.

Of course, for headers we like to display the label by default in front of the header text. This can be done using the `before` attribute where we use the `&label;` key to insert the value of the label. So, the full definition for level 1 headings is:

```
{ label="@h1" before="&label;.&ensp;" }
```

5.5.2. Advanced: Reset counters

If a counter has a dash in its name, of the form `@prefix-name`, then the counter will be reset on every increment of the counter *prefix*. For example, the counter for subsections is reset on every new section (using [metadata rules](#)):

```
~h2: @h1-h2 label="@h1.@h1-h2" before="&label;.&ensp;"
```

Similarly, a counter like `@h1-h2-h3` would reset on every increment of counter `@h1` or `@h1-h2`.

5.5.3. Advanced: Display format

The counters above are displayed as arabic numbers but sometimes we would like to display a number using other formats. When you assign a single letter to a counter, it will continue counting using letters. This is nice for example when starting the appendix where each section is generally numbered using letters:

```
# Appendix (not numbered at all) {-}
```

```
# Section one of the appendix { @h1="A" }
```

This section will be numbered as 'A'

```
# Another appendix section
```

This section will be numbered as 'B'

The display format of a counter can also be set independently of setting its value. Currently, Madoko supports `arabic`, `arabic0`, `upper-case`, and `lower-case` display, where `arabic0` starts counting at 0. Assigning a single letter, like `@h1="A"` in the previous example is just a shorthand for:

```
{ @h1=upper-case @h1=1 }
```

As a final example, suppose we would like to count figures using lower-case letters and per section. We can do this by defining the default attributes for figures as a metadata rule:

```
~Figure: @h1-figure=lower-case label="@h1\/-@h1-figure"
```

The label definition here displays figure labels of the form 2-b for example (where we used the empty [escape sequence](#) `\/` to prevent the counter name `@h1` being read as `@h1-`). Note that because this is a metadata rule, we could not use the assignment `@h1-figure="a"` here, or otherwise every figure would get numbered as `a`. In this case we just want to set the display mode here and not a specific value.

5.6. Advanced: Replacement

Madoko has three attributes that can transform the content of block or inline element, namely `before`, `after`, and `replace`. The `before` and `after` elements just add content before and after:

```
~ Myblock { before="*Myblock*: " after="." }
the content
~
```

Myblock: the content.

The `replace` attribute replaces the entire content with its value. You can have multiple replacers and they are all applied in order. The replacers can be cleared using the special `clear` value. Finally, all replacers can contain entity names (Section 5.2) which are expanded. Useful entities are `&source`; which expands to the current content of the block (possibly having already some replacers applied), and `&nl`; which expands to a newline character. In particular, `before` and `after` are defined in terms of `replace`, where `before=value` is just syntactic sugar for `replace=value\/&source`; and similarly for `after`.

As an advanced example, for this document I defined a metadata rule for the `Sample` custom block that replaces its content by both a code block and a regular markdown block. In a simplified form, it is defined as:

```
~Sample:
  replace="~ Begin SampleBlock&nl;\
    ~~~~&nl;&source;&nl;~~~~&nl;\
    ---- &nl;&source;&nl;\
    ~ End SampleBlock"
```

5.6.1. Advanced: Regular expression replacement

A `replace` attribute can also define a general [regular expression](#) replacement

of the form `/regex/replacer/(g|i|m|c)?`. The regular expression *regex* is matched against the content, and a match is replaced by *replacer*. The options are:

- **g**: instead of just the first match, it will globally replace all matches found in the content.
- **i**: use case-insensitive matching.
- **m**: do a multi-line match where `^` and `$` match the beginning and end of each line respectively instead of just the start and end of the content.
- **c**: do case conversion; enlarges valid escape characters with one of `luLUE` which can be used to do case conversion, see the next section for more information.

Inside the *replacer* we can use the following escape sequences:

- `\digit`: gets replaced by the *digit* capture group in *regex*.
- `\/`: becomes a forward slash.
- `\\`: becomes a single backward slash.

For example, here is an example where we replace `<quoted>` text by single guillemet quotes:

```
~ { replace="/<(.*?)>/&lsaquo;\1&rsaquo;/g" }
Here is < quoted > text.
~
_____
Here is ‹ quoted › text.
```

Case conversion When we pass the `c` flag, we can also use special case conversion escape sequences in the *replacer* expression:

- `\U` or `\L` transform the following text up to the next `\E` (or the end of the replacement) to upper- or lower-case respectively.
- `\u` and `\l` replace the following character to upper- or lower-case.

```
~ { replace="/(\w+)/\u\1/gc" }
all words to title-case.
~
_____
All Words To Title-Case.
```

Mapping strings The `replace` attribute can also perform a *mapping* when it has the form:

```
// regex1 / repl1 // ... // regexn / repln //(g|i|m|c).
```

A mapping matches against the regular expression $(regex_1) | \dots | (regex_n)$, and when it finds a match on some group i , it applies the replacer $repl_i$. The replacement expression can use capture groups as usual.

For example, we could device a *greek* or *hiragana* mode replacement. Here is a simplified example:

```
~ Greek { replace="//a/&alpha;//d/&delta;//n/&nu;//g" }
greek daan.
~
~ Hiragana {replace="//ko/&#12371;//ni/&#12395;//\
\\(.)/\1//chi/&#12385;//ha/&#12399;//\
n/&#12435;//g" \
font-family="MS Gothic" }
konnichiha is Japa\nese.
~
_____
greek δααν.
こんにちは is Japanese.
```

Note that the previous example generally needs a `font-family` specification to display correctly in LaTeX since Japanese characters are not in the standard font. See Appendix A.10 for more information.

5.6.2. Advanced recursion and replacement

As more advanced example of recursion and replacement, we can actually calculate [fibonacci numbers](#). In the following sample, n number of `x` characters are replaced by $fib(n)$ `y` characters:

```
How many `y`s is the Fibonacci of 5 `x`s?
~ Fib
xxxxx
~
_____
How many y's is the Fibonacci of 5 x's?
yyyyyyyy
```

This is done through the following metadata rule:

```
~Fib: replace='^x?$/y/'
      replace='/xx(x*)/~Fib&nl;x\1&nl;~&nl;~Fib&nl;\1&nl;~/'
      notag tight
```

The first replacer will replace a single optional `x` with a `y`. The second one matches 2 or more `x`'s, and replaces these recursively by two new `Fib` blocks: one with $n-1$ and one with $n-2$ `x` characters. These blocks will get processed

now recursively. Finally, by using the `notag` attribute we suppress the inclusion of many `div` elements in the HTML, while the `tight` attribute suppresses the addition of paragraph elements.

This example shows the replacement facility of Madoko is quite powerful. It is even possible to define a generic [SKI combinator expander](#) which makes Madoko's replacement mechanism (almost) Turing complete⁵.

6. References

- [1] J. Fagerberg, D.C. Mowery, and R.R. Nelson, editors. *Oxford Handbook of Innovation*, volume 1. Oxford University Press, Oxford, 2004.
- [2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.
- [3] O. Grandstrand. *Oxford Handbook of Innovation*, chapter Innovation and Intellectual Property Rights. Volume 1 of Fagerberg et al. [1], 2004.
- [4] Donald E. Knuth. *The TeX book*. Addison-Wesley, 1984.
- [5] Leslie Lamport. *LaTeX: A Document Preparation System (2nd Edition)*. Addison-Wesley, 1994. See also [4].

A. Appendix

A.1. Command line options

Generally Madoko is invoked on the command line as:

- `madoko [options] files`

where the options can consist of:

⁵Almost, since Madoko expands only up to a certain limit and since regular expressions cannot express arbitrary nesting levels.

| Short | Long | Description |
|------------------------|-------------------------------|--|
| | <code>--version</code> | Display version information. |
| <code>-v</code> | <code>--verbose</code> | Be more verbose. |
| | <code>--odir=dir</code> | Write output files to the specified directory. |
| | <code>--xmp</code> | Only process markdown between <code><xmp></code> tags. |
| | <code>--tex</code> | Output a LaTeX file too. |
| | <code>--pdf</code> | Generate a PDF file (implies <code>--tex</code>). |
| <code>-f</code> | <code>--full</code> | Generate a full html/latex document. |
| | <code>--sandbox</code> | Run in a sandbox for secure server execution. |
| | <code>--sanitize</code> | Always escape or suppress user defined html. |
| | <code>--pedantic</code> | Pedantic mode. |
| | <code>--bench</code> | For benchmarking: turn off numbering, etc. |
| | <code>--installdir=dir</code> | Set installation directory explicitly. |
| <code>-r</code> | <code>--rebuild</code> | Force rebuild bibliography, math, etc. |
| <code>-mkey:val</code> | <code>--meta=key:val</code> | Semi-colon separated list of metadata values |

All the boolean flags can be negated by prefixing them with `no`, for example to suppress a full document use `--nofull`.

The verbose flag (`-v`) can be repeated to become more verbose. For example, by using `-vv` LaTeX warnings and the size of math images are displayed.

Normally, Madoko writes the output files to the same directory as where the input file resides but you can override this with the `--odir` option. The HTML output for a file `file.mdk` consists of `file.html` and `madoko.css`. The latter contains standard styling for Madoko and is necessary for example to display lines in tables correctly. For LaTeX output (`--tex`), Madoko also emits `file.tex` and two style files, `css.sty` and `madoko.sty`, which contain necessary commands to process the Madoko output in LaTeX.

The `--xmp` option instructs Madoko to only process content between `<xmp>` and `</xmp>` tags. This can be useful if the main body of your document is in HTML and only some parts are written in Madoko.

The `--full` flag forces Madoko to generate a *full* document. A full document is self sufficient and contains for example for HTML proper `<head>` and `<body>` elements instead of just the markup for the content. Similarly, for LaTeX, a full document contains an `\documentclass`, `\begin{document}` and necessary `\usepackage` commands. The `--full` flag is usually not necessary since Madoko will generate a full document by default, but it can be suppressed using `--nofull`.

The `-m` flag can be used to set metadata values. For example, when specifying what command to use to invoke LaTeX, we can say `-mpdflatex:/foo/bar/xelatex` for example.

The `--sandbox` flags runs Madoko in a sandbox: it can only read files underneath the current directory, the output directory, and include paths, and it can only write files underneath the output directory. Moreover, documents cannot change the following metadata keys: `latex`, `pdflatex`, `dvipng`, `math-pdflatex`, `bibtex`, `math-convert`, `convert`, `ps2pdf`, and `dvips`. Note that this flag only

influences the Madoko program, any run of LaTeX or other external programs need to be secured separately.

A.2. Slide shows and presentations

Madoko can generate excellent slide shows using either the [reveal.js](#) library in HTML or the [Beamer](#) package in LaTeX.

Start a slide show with including the `Presentation` style at the start of the document:

```
[INCLUDE=presentation.mdk]
```

Slides are now automatically started on a level 1 (#) or 2 (##) header, or by using a `~Slide` custom block explicitly. Use the `.fragment` class on any element to create overlays that appear one by one. The `.fragmented` class can be used on a list to have each item appear in order. For example:

```
# my first slide
```

And some content

```
# and my second slide
```

```
* with
* a list
* in it
{.fragmented}
```

The HTML slide shows support the `~Notes` custom block for speaker notes, and the `~Vertical` custom block to create a set of vertical slides.

Here is a small [example slide show](#) written in Madoko ([html](#), [pdf](#), [source](#)).

A.2.1. Using Reveal.js

The HTML backend works with the [reveal.js](#) library:

- It expects the reveal.js [library](#) to be in a subdirectory “reveal.js”. You can use another url by setting the `Reveal Url` metadata key, for example:
`Reveal Url: http://lab.hakim.se/reveal-js`
- Select another default theme by setting the `Reveal Theme`, like `sky` for example.
- Use the `data-background` attribute on headings to set the background color or background image for a slide.
- The Javascript `revealConfig` variable contains the default configuration options and can be extended inside `<script>` tags.

A.2.2. Using Beamer

The LaTeX backend uses the [Beamer](#) package:

- Set the `Beamer Theme` metadata key to select another theme, like `singapore` for example.
- Set the `Beamer Theme Options` for different options for the theme.
- Use the `.pause` class to insert pauses anywhere in a slide.

A.3. Advanced: Customizing citations

For each citation, you can actually change how it is displayed by setting the `cite-style` attribute on the citations:

First a natural citation `[@Goo93]{cite-style=natural}`,
then a super one `[@Goo93]{cite-style=super}`.

First a natural citation (Goossens et al., 1993), then a super one².

Moreover, we can customize the formatting of citations by specifying braces etc. The general format of a citation style is:

base `[: (sort|nosort)] [: open,close,sep [, aysep] [, yysep]]`

where *base* is one of `natural`, `numeric`, `textual`, or `super`, and the values *open* to *yysep* are double quoted strings:

- *open*: the opening brace.
- *close*: the closing brace.
- *sep*: separator between citations.
- *aysep*: Optional separator between authors and years. Equal to *sep* by default.
- *yysep*: Optional separator between years with a common author. Equal to *aysep* by default.

For example:

Switch to a compact bold numeric style
`[@Goo93;@FBerg04]{cite-style='numeric:["**","**"],"**;**"}'`
or an unsorted super style
`[@Lamport:Latex;@Knuth:Tex]{cite-style=super:nosort}`

Switch to a compact bold numeric style `[1;2]` or an unsorted super style^{5,4}

Clearly, one should usually only do this for the `Cite Style` metadata key, and not sprinkle this kind of formatting through your prose.

A.4. Advanced: Not using BibTeX

If necessary, it is possible to completely circumvent using BibTeX and write your bibliography entries by hand. First, we need to ensure Madoko will not run

the BibTeX tool for us:

BibTeX: False

Next, we write our bibliography entries inside a Bibliography block:

```
~ Bibliography { caption="00" }
...
~
```

where the `caption` is only used in LaTeX output and should be a string that is the widest label necessary for the numeric style. Inside the bibliography block, you can put `Bibitem` blocks for each bibliography entry. For example:

```
~~ Bibitem { #WadlerThiemann03 }
Philip Wadler and Peter Thiemann.
_The marriage of effects and monads._
ACM Trans. Comput. Logic, 4(1):1--32, 2003.
~~
```

Moreover, you can add a `caption` that is displayed when hovering over a citation, and a `searchterm` that is used when the user clicks on the magnifying glass icon:

```
~~ Bibitem { #WadlerThiemann03 \
  caption="Wadler and Thiemann: \
    The marriage of effects and monads" \
  searchterm="Wadler+Thiemann+Marriage+Effects+Monads" }
Philip Wadler and Peter Thiemann.
_The marriage of effects and monads._
ACM Trans. Comput. Logic, 4(1):1--32, 2003.
~~
```

The above example is for numeric style citations. For author-year citations, the entry should have an explicit `cite-label`:

```
~~ Bibitem { #remy cite-label='R  my(1993)' }
Didier R  my.
_Type inference for records in a natural extension of ML._
In Carl\ A. Gunter and John\ C. Mitchell, editors,
Theoretical Aspects Of Object-Oriented Programming.
Types, Semantics and Language Design. MIT Press, 1993.
~~

~~ Bibitem { id='nielson:polyeffect' \
  cite-label='Nielson et\ al.(1997)Nielson, Nielson, and Amtoft'}
Hanne\ Riis Nielson, Flemming Nielson, and Torben Amtoft.
_Polymorphic subtyping for effect analysis:
The static semantics._
In Selected papers from the 5th LOMAPS Workshop on
Analysis and Verification of Multiple-Agent Languages,
```

pages 141--171, 1997. ISBN 3-540-62503-8.

~~

A `cite-label` should have the form *authors(year)longauthors*, where the *longauthors* are optional. Madoko parses these labels to correctly format citations.

A.5. Advanced: packages in dynamic math mode

In dynamic math mode packages are not loaded since MathJax cannot handle arbitrary LaTeX packages. We need a special MathJax ‘extension’. For example, the `amscd` package is available as `AMScd` and we can use it in MathJax as:

MathJax Ext: `AMScd`

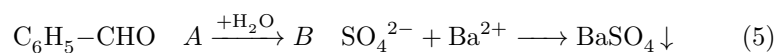
Another package that is available in both Latex and MathJax is the `mhchem` package:

MathJax Ext: `mhchem`

Package : `mhchem`

which can be used to easily draw chemical formulas:

```
~ Equation
\ce{C6H5-CHO}\quad
\ce{A$ ->[\ce{+H2O}] B$}\quad
\ce{SO4^2- + Ba^2+ -> BaSO4 v}
~
```



In the previous examples, it may seem superfluous to specify both a MathJax extension and LaTeX package but not in all cases the names or package functionality happens to be exactly the same.

A.6. Advanced: Using Prettify to highlight code

If you are not fond of static highlighting, you can also highlight syntax dynamically using [Google Prettify](#). Of course, this only works HTML output, PDF output is still highlighted using Madoko. To enable prettify, first include the Google script in the metadata (see Section 5.1):

Script: `https://google-code-prettify.googlecode.com/svn/loader/run_prettify.js`

and add the `.prettyprint` class using attributes:

```
``` javascript { .prettyprint .linenums }
```

```
function hi() {
 return "hi";
}
...

function hi() {
 return "hi";
}
```

Note that Madoko automatically disables static syntax highlighting in the HTML output if the `.prettyprint` class is present.

## A.7. Advanced styling in LaTeX

Sometimes we need to style LaTeX code specially, or handle cases where the CSS elements are not yet supported by Madoko (as described in Section 5.3.2).

For the LaTeX backend, all attributes are passed to the LaTeX commands, and there are generic hooks to apply LaTeX formatting. For example, we can include the following LaTeX commands in a `TexRaw` section to make all `Slanted` custom blocks use a slanted font:

```
~ TexRaw
\cssClassRule{slanted}{font-style=oblique}
~
```

The following commands are quite useful for this:

- `\cssClassRule{class}{attributes}`: adds *attributes* to any command that contains *class*.
- `\cssClassRuleEnv{class}{env}`: encloses any block with class *class* in the LaTeX environment *env* (after margins and padding).
- `\cssClassRuleCmd{class}{\cmd}`: applies any content with class *class* to the single argument LaTeX command `\cmd` (after margins and padding).
- `\cssClassRuleDoBefore{class}{cmds}`: execute commands *cmds* before formatting the content for any block that contains the class *class* (but after margins and padding).
- `\cssClassRuleDoAfter{class}{cmds}`: execute commands *cmds* after formatting the content for any block that contains the class *class*.
- `\cssClassRuleDo{class}{cmds}`: execute commands *cmds* when processing the attributes. These *cmds* can contain:
  - `\cssDoBefore{cmds}`: executes *cmds* before formatting the contents.
  - `\cssDoAfter{cmds}`: execute *cmds* after formatting the contents.
  - `\cssDoEnv{env}`: enclose the formatting in *env*.
- `\cssIfHasClass{class}{then}{else}`: This command is can be used inside *cmds* of the `\cssClassDo(Before|After|)` commands. If this element has *class* then execute the *then* part, else the *else* part.

- `\cssNewKey{css}{attrname}{\csscmdname}{default}`: Define a new attribute *attrname*. When present its value gets bound to `\csscmdname`. The *default* value is optional.
- `\cssNewLengthKey{css}{attrname}{\csscmdname}`: Define a new attribute *attrname* that contains a length. When present its value gets bound to `\csscmdname`. The *default* value is `0pt`.
- `\cssElemRule{elem}{attributes}`: adds *attributes* to any custom block *elem*.
- `\cssElemRule(Env|Cmd|Do|DoBefore|DoAfter)`. As the `Class` variants but for elements. In contrast to the `Class` rules, these variants will apply commands before margins and padding is applied. If this is not wanted, use the `Class` variant instead since those also match on element names.

Inside rules, you can also access most CSS attributes that defined through `\cssNewKey` or `\cssNewLengthKey`. The attributes are named using camel casing with a `css` prefix, i.e. `\cssCamelCasing`, where `text-indent` is stored in `\cssTextIndent`, or `border-left-style` as `\cssBorderLeftStyle`.

As an advanced example, here is how `madoko.sty` defines the rules for the `Figure` custom block:

```
\cssNewKey{css}{page-align}{\cssPageAlign}{}
\cssElemRuleDo{figure}{%
 \eifstrequal{\cssPageAlign}{top}%
 {\def\@pagealign{t}}%
 {\eifstrequal{\cssPageAlign}{bottom}%
 {\def\@pagealign{b}}%
 {\eifstrequal{\cssPageAlign}{here}%
 {\def\@pagealign{h}}%
 {\def\@pagealign{tbp}}}% default
\cssIfHasClass{wide}%
 {\cssDoBefore{\@expandafter{\begin{figure*}}[\@pagealign}}%
 \cssDoAfter{\end{figure*}}}%
 {\cssDoBefore{\@expandafter{\begin{figure}}[\@pagealign}}%
 \cssDoAfter{\end{figure}}}%
}

\newlength{\mdCaptionlen}
\newcommand{\mdCaption}[1]{%
 % center a caption if it is smaller than the line width
 \settowidth{\mdCaptionlen}{#1}%
 \ifnum\mdCaptionlen<\linewidth%
 \parbox{\linewidth}{\noindent\centering #1}%
 \else%
 \parbox{\linewidth}{\noindent #1}%
 \fi%
}
\cssClassRuleCmd{figure-caption}{\mdCaption}
```

```
\cssClassRule{figure-caption}{text-align=justify}
```

Where we recall that the standard metadata rules for a `Figure` are:

```
~Figure: label='[@figure]{.figure-label}' .align-center
 after='&nl;***** {.figureline}&nl;\
 [[**Figure\ &label;.*]\
 {.caption-before}&caption;]\
 {.figure-caption}&nl;{notag}&nl;'
 toc=tof toc-line='&label;. &caption;'
```

as shown in Appendix A.12. The sample uses two extra functions from `css.sty`:

- `\eifstrequal{str1}{str2}{then}{else}`: expands the content of *str1* and then compares for equality with *str2*, and executes either the *then* or *else* branch. Use `\eeifstrequal` if both string arguments must be expanded.
- `\@expandafter{arg1}{arg2}`: first expands *arg2* and then continues executing *arg1* (followed by the expansion of *arg2*). Used in the previous example to expand the optional argument to the `figure` environment to one of `b`, `h`, or `t`.

## A.8. Special attribute keys

Some attributes have a special meaning to Madoko. In particular:

- `tight`: Suppresses enclosing the first text content in this block as a paragraph. This is used for example for a `BibItem` block.
- `notag`: Suppresses the output of a `div` tag (or LaTeX environment) for a custom block.
- `toc`, `toc-line`, `toc-depth`: Used for generating custom table of contents. Used for example by headings. See Section 4.10 for more information.
- `before`, `replace`, and `after`: Used to transform the content of the block. See Section 5.6.
- `start=num`: Gives the start number for an ordered list (Section 4.4).
- `target=frame`: Gives the HTML frame target for a link.
- `math-needspdf`: If `true`, forces the `pdf` conversion method for an equation (instead of using `dvi`). This is needed when using fancy LaTeX math using TikZ for example.
- `snippet-needspdf`: If `false`, uses the `dvi` conversion method for a snippet (instead of using `pdf`). This can improve performance of image extraction. (Section 4.9.5).
- `texelem=elem`: Uses *elem* as the TeX environment instead of the name of the custom block.
- `id=id`: Sets the *id* of an element. Usually this is done using a hash name directly as `#id`.
- `class=classes`: Adds to the classes of this element. Usually set using a dot name as in `.class`. If you use `class=clear`, the classes are cleared. Similarly, you can use `.class=clear` to remove one particular class.

- **label=***label*: Sets the label of an element. This is used when referencing this block as explained in Section 4.2
- **cite-label=***label*: Sets the citation label of an `BibItem` block. This is used when referencing a bibliography item as explained in Section 4.11
- **cite-style=***style*: Sets the citation style for a citation (Section 4.11.2).
- **lines=***lines*: Set the height of a float figure to *lines* height (Section 5.3.3).
- **sticky**: The attributes for this element will stick to apply to all of the same elements that follow in the document.
- **html-name**: use *name* as an HTML attribute only in the HTML backend. For example, you can use **html-target=\_top** for links inside frames.
- **css-name**: use *name* as a CSS attribute only in the HTML backend. For example, you can use **css-font-family=***Cambria* for setting a font family only in the web page.
- **tex-name**: use *name* as an attribute only in the LaTeX backend.
- **data-name**: in the HTML backend uses a HTML attribute **data-name** instead of a CSS style attribute.
- **-:** clear any attributes that have been set before (usually through meta-data rules).
- **input=***input*. Sets the input mode of this block which determines how the content of this block is processed. One of:
  - **pre**: Preformatted code (Section 4.7).
  - **raw**: Raw unprocessed input that is passed directly to the output.
  - **texraw**: Raw TeX code that is passed directly to LaTeX (See [TeXRaw](#))
  - **htmlraw**: Raw HTML code that is passed directly to the HTML output (See [HtmlRaw](#))
  - **math**: LaTeX mathematics mode input. This input is usually invoked through an `Equation` block as described in Section 4.9.
  - **mathpre**: Preformatted LaTeX mathematics. This input is usually invoked through a `MathPre` block (Section 4.9.6).
  - **mathdefs**: LaTeX math definitions. This input is usually invoked through a `MathDefs` block (Section 4.9.2).
  - **normal, markdown**: Regular Madoko markdown input (default).
  - **texonly**: Regular Madoko markdown that is only processed when generating LaTeX output. Usually used through the `TexOnly` block (Section 4.12).
  - **htmlonly**: Regular Madoko markdown that is only processed when generating HTML output. Usually used through the `HtmlOnly` block (Section 4.12).

## A.9. Special attribute classes

Some class attributes are treated specially by Madoko. In particular:



- **.para-block**. If a block element has the **.para-block** class, it will be considered part of the preceding paragraph (and not end that paragraph). This is for example done for the **Equation** block since equations are part of a paragraph. This improves typography significantly, especially for the LaTeX backend. Madoko will also assign the **para-continue** class to the paragraph preceding a **.para-block** element.
- **.para-end**: Normally, equations have a **.para-block** class and are thus considered part of the paragraph. If you would like such block to be the end of a paragraph, you should assign the **.para-end** class to end it explicitly. In the LaTeX backend this will cause the next paragraph to be indented.
- **.indent**. This class is automatically added by Madoko to any paragraph that follows other paragraphs or **.para-blocks** (and is not inside a list). This can be used to switch from *block* mode paragraphs to *indented* paragraphs using some CSS:

```
p.indent { text-indent: 1em }
```

- **.align-center**, **.align-right**, **.align-left**. These classes align the *content* of a block (instead of the block itself) to the center, right, or left.
- **.hidden**. Hides the content of the block.
- **.block**. Treat this as a block element instead of inline content. For example, in HTML output this will add a top and bottom margin to the element.
- **.wide**. Used for **Figure** blocks: in a two-column output, this figure will span both columns horizontally (see also Section 4.12.1).
- **.free**, **.textual**. These classes are used for typesetting citations (see Section 4.11.2).
- **.long**. Used to signify that a table might span multiple pages so that the LaTeX backend can use the **longtable** environment instead of the regular **tabular** environment (which cannot break across pages).
- **.plain**: used for code blocks to use a plain verbatim mode in LaTeX (see Section 4.7).
- **.list-star**, **.list-dash**, **.list-plus**. Used to customize appearance of bullets in a list.

## A.10. Unicode characters

Madoko already recognizes all named html entities and translates them to the appropriate LaTeX commands (Appendix A.11). If you often need a specific unnamed unicode character, it can be convenient to define a shorthand for it in the metadata:

```
llb: ⟦
rrb: ⟧
```

and then use those entity names instead. For example:

```
&llb;hi&rrb;.
```

---

```
[[hi]].
```

### A.10.1. Unicode in LaTeX

Unfortunately, in LaTeX these unicode characters are still unknown. You can define your own definition for it using the LaTeX `\mdDefineUnicode` command in a `TeXRaw` block at the start of the document. For example:

```
~ TeXRaw
\mdDefineUnicode{10214}{\ensuremath{\llbracket}}
\mdDefineUnicode{10215}{\ensuremath{\rrbracket}}
~
```

If no definition is given, LaTeX will work but output the entity as is, i.e. `&#10214;` for `#llbracket`. Under a unicode aware LaTeX, like [XeLaTeX](#) or [LuaLaTeX](#), it will call `\mdUnicodeChar` which will use the `\char` command to directly select the glyph from the current font.

### A.10.2. Unicode font selection in LaTeX

The previous examples works fine if only some unicode characters are used, but becomes cumbersome if you need many unicode characters, for example, when writing Japanese or Chinese characters as shown in our previous example in [Section 5.6.1.2](#). In this case, we need to select a font family in LaTeX that supports the glyphs directly. For example, MS Gothic or SimSun:

```
~ { font-family="MS Gothic"}
こんにちは is Japanese.
~
```

---

```
こんにちは is Japanese.
```

Instead of using explicit unicode characters, you can also directly use UTF8 text files with the characters shown directly. For example:

```
Directly in unicode: []
(probably not visible in LaTeX monospace fonts)
```

---

```
Directly in unicode: [] (probably not visible in LaTeX monospace fonts)
```

If most of your document is using such characters, the `\setmainfont` command of the `fontspec` package can set the default for the entire document instead of using attributes. See the package [documentation](#) for more information.

### A.11. Recognized character entities

Madoko recognizes all HTML5 named character entities and translates them correctly in LaTeX. Besides the standard named entities, Madoko also recognizes some extra named entities, like *bar*, *bslash*, *pagebreak*, *strut*, etc. These are denoted in the table with a star. The full list of predefined entities is:

number	name	glyph	remark
34	quot	"	
35	hash*	#	
36	dollar*	\$	
37	perc*	%	
38	amp	&	
39	apos	'	
40	lpar*	(	
41	rpar*	)	
42	ast*	*	
43	plus*	+	
47	fslash*	/	
60	lt	<	
62	gt	>	
92	bslash*	\	
94	caret*	^	
95	underscore*	_	
96	grave*	`	
123	lcurly*	{	
124	bar*		
125	rcurly*	}	
126	tilde*	~	
160	nbsp		~ in LaTeX
161	iexcl	!	
162	cent	¢	
163	pound	£	
164	curren	¤	
165	yen	¥	
166	brvbar		
167	sect	§	
168	uml	¨	
169	copy	©	
170	ordf	ª	
171	laquo	«	
172	not	¬	
173	shy		
174	reg	®	
175	macr	¯	

176	deg	°
177	plusmn	±
178	sup2	²
179	sup3	³
180	acute	´
181	micro	μ
182	para	¶
183	middot	·
184	cedil	¸
185	sup1	¹
186	ordm	º
187	raquo	»
188	frac14	¼
189	frac12	½
190	frac34	¾
191	quest	¿
192	Agrave	À
193	Aacute	Á
194	Acirc	Â
195	Atilde	Ã
196	Auml	Ä
197	Aring	Å
198	AElig	Æ
199	Ccedil	Ç
200	Egrave	È
201	Eacute	É
202	Ecirc	Ê
203	Euml	Ë
204	Igrave	Ì
205	Iacute	Í
206	Icirc	Î
207	Iuml	Ï
208	ETH	Ð
209	Ntilde	Ñ
210	Ograve	Ò
211	Oacute	Ó
212	Ocirc	Ô
213	Otilde	Õ
214	Ouml	Ö
215	times	×
216	Oslash	Ø
217	Ugrave	Ù
218	Uacute	Ú
219	Ucirc	Û
220	Uuml	Ü

221	Yacute	Ÿ
222	THORN	Þ
223	szlig	ß
224	agrave	à
225	aacute	á
226	acirc	â
227	atilde	ã
228	auml	ä
229	aring	å
230	aelig	æ
231	ccedil	ç
232	egrave	è
233	eacute	é
234	ecirc	ê
235	euml	ë
236	igrave	ì
237	iacute	í
238	icirc	î
239	iuml	ï
240	eth	ð
241	ntilde	ñ
242	ograve	ò
243	oacute	ó
244	ocirc	ô
245	otilde	õ
246	ouml	ö
247	divide	÷
248	oslash	ø
249	ugrave	ù
250	uacute	ú
251	ucirc	û
252	uuml	ü
253	yacute	ý
254	thorn	þ
255	yuml	ÿ
321	lstroke	Ł
322	Lstroke	ł
338	OElig	Œ
339	oelig	œ
352	Scaron	Š
353	scaron	š
376	Yuml	Ÿ
402	fnof	ƒ
710	circ	ˆ
732	tilde	˜
818	lowline	-

(in LaTeX becomes a short underscore)

913	Alpha	$A$
914	Beta	$B$
915	Gamma	$\Gamma$
916	Delta	$\Delta$
917	Epsilon	$E$
918	Zeta	$Z$
919	Eta	$H$
920	Theta	$\Theta$
921	Iota	$I$
922	Kappa	$K$
923	Lambda	$\Lambda$
924	Mu	$M$
925	Nu	$N$
926	Xi	$\Xi$
927	Omicron	$O$
928	Pi	$\Pi$
929	Rho	$P$
931	Sigma	$\Sigma$
932	Tau	$T$
933	Upsilon	$\Upsilon$
934	Phi	$\Phi$
935	Chi	$X$
936	Psi	$\Psi$
937	Omega	$\Omega$
945	alpha	$\alpha$
946	beta	$\beta$
947	gamma	$\gamma$
948	delta	$\delta$
949	epsilon	$\epsilon$
950	zeta	$\zeta$
951	eta	$\eta$
952	theta	$\theta$
953	iota	$\iota$
954	kappa	$\kappa$
955	lambda	$\lambda$
956	mu	$\mu$
957	nu	$\nu$
958	xi	$\xi$
959	omicron	$o$
960	pi	$\pi$
961	rho	$\rho$
962	sigmaf	$\varsigma$
963	sigma	$\sigma$
964	tau	$\tau$
965	upsilon	$\upsilon$
966	phi	$\varphi$

967	chi	$\chi$	
968	psi	$\psi$	
969	omega	$\omega$	
977	thetasym	$\vartheta$	
978	upsih	$\Upsilon$	
982	piv	$\varpi$	
8194	ensp		0.5em space in LaTeX
8195	emsp		1em space in LaTeX
8195	quad*		<code>\quad</code> in LaTeX
8196	thicksp*		<code>\;</code> in LaTeX
8197	medsp*		<code>\:</code> in LaTeX
8201	thinsp		<code>\,</code> in LaTeX
8203	strut*		<code>\strut</code> in LaTeX
8203	pagebreak*		<code>\pagebreak</code> in LaTeX
8204	zwnj		
8205	zwj		
8206	lrm		
8207	rlm		
8211	ndash	—	
8212	mdash	—	
8216	lsquo	‘	
8217	rsquo	’	
8218	sbquo	‚	
8220	ldquo	“	
8221	rdquo	”	
8222	bdquo	„	
8224	dagger	†	
8225	Dagger	‡	
8226	bull	•	
8230	hellip	...	
8240	permil	‰	
8242	prime	′	
8243	Prime	″	
8249	lsaquo	‹	
8250	rsaquo	›	
8254	oline	-	
8260	frasl	$\frac{1}{2}$	
8364	euro	€	
8465	image	ℑ	
8472	weierp	℘	
8476	real	ℜ	
8482	trade	™	
8501	alefsym	ℵ	
8592	larr	←	
8593	uarr	↑	
8594	rarr	→	

8595	darr	$\downarrow$
8596	harr	$\leftrightarrow$
8629	crarr	$\curvearrowright$
8656	lArr	$\Leftarrow$
8657	uArr	$\Uparrow$
8658	rArr	$\Rightarrow$
8659	dArr	$\Downarrow$
8660	hArr	$\Leftrightarrow$
8704	forall	$\forall$
8706	part	$\partial$
8707	exist	$\exists$
8709	empty	$\emptyset$
8711	nabla	$\nabla$
8712	isin	$\in$
8713	notin	$\notin$
8715	ni	$\ni$
8719	prod	$\prod$
8721	sum	$\sum$
8722	minus	$-$
8727	lowast	$*$
8730	radic	$\sqrt{\phantom{x}}$
8733	prop	$\propto$
8734	infin	$\infty$
8736	ang	$\angle$
8743	and	$\wedge$
8744	or	$\vee$
8745	cap	$\cap$
8746	cup	$\cup$
8747	int	$\int$
8756	there4	$\therefore$
8764	sim	$\sim$
8773	cong	$\cong$
8776	asympt	$\approx$
8800	ne	$\neq$
8801	equiv	$\equiv$
8804	le	$\leq$
8805	ge	$\geq$
8834	sub	$\subset$
8835	sup	$\supset$
8836	nsup	$\subsetneq$
8838	sube	$\subseteq$
8839	supe	$\supseteq$
8853	oplus	$\oplus$
8855	otimes	$\otimes$
8869	perp	$\perp$
8901	sdot	$\cdot$



8942	vellip	$\vdots$
8968	lceil	$\lceil$
8969	rceil	$\rceil$
8970	lfloor	$\lfloor$
8971	rfloor	$\rfloor$
9001	lang	$\langle$
9002	rang	$\rangle$
9674	loz	$\diamond$
9824	spades	$\spadesuit$
9827	clubs	$\clubsuit$
9829	hearts	$\heartsuit$
9830	diams	$\diamondsuit$
8617	hooklarr*	$\hookleftarrow$
8718	bbox*	$\blacksquare$
9633	box*	$\square$
9744	ballotbox*	$\square$
9745	ballotc*	$\square$
9746	ballotx*	$\boxtimes$
10003	checkmark*	$\checkmark$
10004	bcheckmark*	$\checkmark$
10007	xmark*	$\times$
10008	bxmark*	$\times$
128270	mglass*	$\rho$

---

## A.12. Definitions of predefined custom blocks

These are the definitions of the predefined custom blocks, where we assume that `Heading Depth` and `Toc Depth` are set to 3 (and adding some spurious line breaks for formatting).

```

~Equation: label='[@equation]{.equation-label}'
 replace='&label;]{.equation-before}&nl;\
 ~ Begin Math&nl;&source;\
 &nl;~ End Math&nl;'\
 .align-center tight .para-block
~Figure : label='[@figure]{.figure-label}' .align-center
 after='&nl;***** {figureline}&nl;\
 [[**Figure\ &label;.**]\
 {caption-before}&caption;]\
 {figure-caption}&nl;{notag}&nl;'\
 toc=tof toc-line='&label;.\ &caption;'\

~Bibitem : label='[@bibitem]{.bibitem-label}'
 cite-label='&label;'\
 before='[\&label;\ \]{.bibitem-before}'

```

```

 tight texelem=mdBibitem
~Bibliography: bib-style=...
 bib-data='...'
 texelem=mdBibliography

~Note : .block before='[**Note**.]{.note-before}'
~Remark: .block before='[**Remark**.]{.remark-before}'
~Proof : .block before='[**Proof**.]{.proof-before}'

~Framed: border-style=solid border-color=black border-width=1px
~Center: .align-center

~Pre : input=pre .para-block
~Code : input=pre
~Hr : .madoko,
~Table : .madoko,

~Article : html-elem='article'
~Section : html-elem='section' tex-elem='mdSection'
~Aside : html-elem='aside'
~Nav : html-elem='nav'

~Tex : input=tex
~TexRaw : input=texraw
~HtmlRaw : input=htmlraw
~Math : input=math .para-block
~MathPre : input=mathpre .para-block
~MathDefs : input=mathdefs .hidden
~HtmlOnly : input=htmlonly
~TexOnly : input=texonly

~Date : ...,
~Time : ...,
~Madoko Version: ...,

.booktable:
 rule-top-width=2px rule-bottom-width=2px
 rule-mid-width=1.35px th-font-weight=normal
 rule-top-sep=4pt rule-bottom-sep=4pt rule-mid-sep=3pt
 th-padding-top=3pt th-padding-bottom=3pt

~Bibitem: replace='?if &bib-search-url; && &searchterm;\
 ?then &source; [&mglass;]\
 (http://&bib-search-url;/search\
 ?q=&searchterm;){.bibsearch}''
@html Bib Search Url: www.bing.com"

```

```

~Theorem : .block label='[@Theorem]{.Theorem-label}'
 before='**Theorem\ &label;.**]{.Theorem-before}'
~Lemma : .block label='[@Lemma]{.Lemma-label}'
 before='**Lemma\ &label;.**]{.Lemma-before}'
~Proposition: .block label='[@Proposition]{.Proposition-label}'
 before='**Proposition\ &label;.**]{.Proposition-before}'
~Corollary: .block label='[@Corollary]{.Corollary-label}'
 before='**Corollary\ &label;.**]{.Corollary-before}'
~Example: .block label='[@Example]{.Example-label}'
 before='**Example\ &label;.**]{.Example-before}'
~Definition: .block label='[@Definition]{.Definition-label}'
 before='**Definition\ &label;.**]{.Definition-before}'

~H1: label='[@h1]{.heading-label}' toc
 before='&label;. ]{.heading-before}'
~H2: @h1-h2 label='[@h1.@h1-h2]{.heading-label}' toc
 before='&label;. ]{.heading-before}'
~H3: @h1-h2-h3 label='[@h1.@h1-h2.@h1-h2-h3]{.heading-label}'
 before='&label;. ]{.heading-before}' toc
~H4: @h1-h2-h3-h4
 label='[@h1.@h1-h2.@h1-h2-h3.@h1-h2-h3-h4]{.heading-label}'
...

```