Open Manual of Markdown Style

Julien Dutant and the Dialoa team

Table of contents

Preface

This book is an open manual of style for scholarly publishing using markdown. Open because it focuses on open-source software tools and is targeted at open-access publishers. Of markdown style because it combines style and technical advice for producing professional-quality publications using markdown. The manual itself is open-source, open-access and written in markdown.

Presentation

The OMMS aims to make markdown-based professional-quality publishing available to academic teams with no advanced technical knowledge. It explains how to convert manuscripts into, or author them as, markdown documents, and turn them into beautifully typeset outputs in multiple print and digital formats.

No prior knowledge of markdown is required; authoring and copyediting can largely be done using visual tools. Production, however, realistically requires some technical knowledge (CSS for web output, LaTeX for print output).

Relation with Quarto

The OMMS overlaps with, and complements, the excellent Quarto documentation.

- The OMMS focuses on copyediting and production. It is aimed at small publishing outfits rather than authors - though you can use it to write a thesis.

- The OMMS gives prescriptive style and typesetting advice. It's a manual of style rather than technical documentation.

Status

The manual is a work in progress. It is used to produce the open-access philosophy journal Dialectica and compiles the solutions we've adopted to deal with a range of typesetting problems. It is fairly extensive but far from complete.

Edit the guide

You're welcome to edit the guide. To do so:

- Install Quarto. Some package managers have it (`brew install quarto` on MacOs, `choco install quarto` on Windows, but not on Linux.) See Chapter **??** on package managers.

- Open the `guide` folder in RStudio (or Visual Code, if you use Quarto with that).
- Chapters are listed in `_quarto.yaml` . If you create a new chapter you need to add it here.
- Chapter files are in the `chapters` folder - obviously. Numbers in their names are for convenience: they ensure that chapters in the folder appear in the same order as they appear in the book.
- Chapters can be edited in visual or source mode. They're "Quarto markdown" (`.qmd`) files but that's just like Pandoc markdown.
- To render a chapter open and it and use Render > Render HTML. You'll see the output in RStudio's preview pane.
- To render the whole book Build > Render Book or open the `index.qmd` file and do Render > Render HTML.

Credits and license

Developed by the Dialoa team and philosophie.ch.

Part I

Authoring

1 Introduction

This part gives instructions for authors.

2 Instructions for authors

Authors can prepare manuscripts in markdown, docx, LaTeX, html formats. Below are some recommendations for authoring manuscripts in formats other than markdown. For markdown format, they can use the copyeditor recommendations of this guide.

2.1 Generic

Provide an informative abstract, 50 to 200 words.

Formulate it carefully: it is indexed in databases and search engines, and plays in crucial role in the long-term visibility of your article.

2.2 MS Word

Authors should use Word's styles:

- Heading 1, Heading 2 etc for headings
- Title for the title
- Block citations
- (optional) Author for author (requires creating a style called "Author)

Formulas should be entered in Word's equation mode. This helps our document converter to turn them into reusable LaTeX formulas.

References should be provided in a separate BibTeX (`.bib`) or CSL-Json file (e.g. using Zotero).

[*Requires extension:* `sections-to-meta`] Abstract and acknowledgements should be provided below "Abstract" and "Acknowledgements" headings at the beginning of the paper. If the paper itself doesn't start with a heading, separate those from the main text by a horizontal line (three dashes and return).

If the document contains complex mathematical symbols or formulas, make sure that these are put in "equation mode". This ensures that pandoc converts them to LaTeX code and spares you the need to re-encode them.

There are various ways in which authors can make the editing process easier and quicker:

- Checking their bibliography for mistakes
- Sending their bibliography as a .bib file, which can easily be created if they use software such as Zotero.

- (If they use Word) Preparing their Word document: writing their formulas in “equation mode”, using the definite styles for article title, abstract, author name, section titles, block citations.
- Remind them of the importance of a good title and abstract.

2.3 LaTeX

Authors should provide a LaTeX source file, with references in a a separate BibTeX file and any included file (images, preamble) or package not available in standard distributions.

TikZ figures and other advanced packages (ie those that aren’t handled by MathJax) should be compilable as standalone documents.

3 Grammar and usage

3.1 Possessives

Possessives and plural. Distinguish shared vs separate ownership:

- Ali and Ben's book(s). The book(s) jointly authored by Ali and Ben.
- Ali's and Ben's book(s). The book(s) authored by Ali and the book(s) authored by Ben.

Part II

Copyediting

4 Preliminaries

This chapter explains what you need to set up a markdown production chain.

4.1 What you need

4.1.1 Skills

- Use a text editor (RStudio will do).
- Optionally, use RStudio's visual editor. This provides a MS Word-like way of editing a markdown document: no need to view the source code. An easy entry into markdown and a nicer way to edit. See Section **??**.
- Optionally, how to use the command line interface. The very basics are enough: navigate to a directory, run a command. See Appendix **??**.

4.1.2 Software

All the software needed is free: Pandoc, LaTeX and optionally RStudio.

See the Dialectica chapter on generating outputs to set yourself up Chapter **??**.

4.2 Working folder

Create a working folder for your journal. This may be shared between copyeditors if you're a team. We suggest a structure such as this:

```
tree/copyediting
  2023-v51
      01-i01
          01-ehrenfest-afanassjewa
              history
          02-estrin
      03-i02
          01-ehrenfest-afanassjewa
              history
          02-estrin
              history
tree/published
```

```
   2023-v50
       01-i01
           01-ayrton
               history
           02-berdichevsky
               history
       03-i02
           01-borg
               history
           02-daubechies
               history
tree/guide
tree/resources
   fonts
tree/template
   1.0
       defaults
       filters
       metadata
       scripts
   1.1
       defaults
       filters
       metadata
       scripts
```

We're dividing the articles in two stages, *copyediting* and *published*. We could further divide copyediting into copyediting proper (language, style) and production (typesetting) but these tend to be intertwined in markdown production.

'Copyediting' and 'published' folders are organized by year-volume and month-issue.

Each article has its own folder, with a `history` subfolder to keep track of the author's original submissions and revisions.

To these we add a few other folders for resources shared among copyeditors:

- `template`: template engine to produce the journal. This might change over the years, so it is organized by versions. Each issue includes metadata that indicate which version of the template was used. It includes `scripts` that copyeditors may need to copy or use.
- `guide`: this guide
- `resources`: resources such as fonts, logos, documentation.

4.3 New Workflow Start [Draft]

Note: below, when saying "in a terminal", we mean either PowerShell on Windows, or your default terminal in Mac or Linux. All of the commands provided should work in all PowerShell (Windows), zsh (Mac) and bash (Linux).

The association is setting up a standardized compilation environment for Dialectica. The environment is encapsulated in a *docker container* and includes all of the tools needed for the copyediting work, including pandoc, LaTeX, Lua, Quarto, and the fonts. The idea is that you will only need to interact with the container in your local machine, as if it was a "box" containing the Dialectica copyediting tools, without needing to install each tool separately. In this way, all of the copyeditors can share a single complete environment (i.e., the versions of all of the tools used by the team will be the same), minimizing compiling problems and compatibility issues. This also allows for issues to be fixed in a centralized manner, as new versions of the environment (with fixed issues) will affect everyone.

Follow the steps here everytime you want to start working using the Dialectica compilation environment. This section assumes you have already set up your machine as explained in Section **??**.

1. First make sure that docker is running. In a terminal, do:

```
docker ps
```

If there are no errors, then docker is running. If not, try the following to start docker:

- On Windows, open Docker Desktop
- On Mac, in a terminal, do `colima start`
- On Linux, in a terminal, do `sudo service docker start`
- Now wait until docker starts, and do `docker ps` again

2. Now check if you have the latest version of the image of the compilation environment. Every new release will be announced on the Team News, and they shouldn't happen very often. If a new version has been released, or if you're unsure if you have the last version, in a terminal do the following. For this, you will need the `access token` which you can find in the Institutional Setup page in our Google Drive:

```
docker login -u philosophiech
# Paste the access token as the password
docker pull philosophiech/dltc-env:latest
# Wait until the pull is complete, and check with
docker image ls
```

NOTE: terminals often hide the characters when you type in passwords. Just copy and paste the access token when asked for the password and it will work. - If you downloaded a new image, you need to stop any container created with the old one. For this, go to the root of the git repository (where the docker compose file is) and do

```
# On Windows and Linux
docker compose down
# On Mac
docker-compose down
```

3. From step 1, if you do have the latest image, and if in the output of `docker ps` you see the name "philosophiech/dltc-env:latest", then the compilation environment is ready to be used

- If not, in a terminal, navigate to the root of the git repository you cloned (where the docker compose file is), and do:

```
git pull

# On Windows and Linux
docker compose up -d
# On Mac
docker-compose up -d
```

Now if you do `docker ps` you should see the name of the container ("philosophiech/dltc-env:latest").

4. Start VSCode, select "Remote Explorer" on the vertical toolbar to the left, select "Dev Containers" on the dropdown menu above, then hover above "compilation-env" and click on the arrow next to it. You can now "Open Folder" and choose "dltc-workhouse", which is the same folder you have in Dropbox.

All set, you're ready to work. Remember that you can open a terminal inside VSCode (in the menu above: Terminal » New Terminal), which will give you a terminal *inside* the container. Here you'll find all of the tools you need (pandoc, LaTeX, Lua, quarto, dltc-make, and the fonts).

5 Preparing a manuscript

Steps to prepare a manuscript for copyediting.

5.1 Create a working folder for your article

If you don't have a dedicated working folder for your article, create one.

If you're going to use RStudio, create a project for the article with File > New Project…. Use 'Existing directory' if your project already has a folder, otherwise "Create project in a new directory" to create one.

5.2 Place original materials in a preserved folder

A submission comes with a manuscript file and possibly others: bibliography, PDF version, figures, etc. It's handy to have those in your working folder for the article, but you want to keep them safe. Place them either in a subfolder called `original` or `history`. (The `history` folder will then be used to keep track of your exchanges with the author.)

5.3 Convert to markdown

Two options: conversion box (RStudio) or with Pandoc. The first is easy if you're not familiar with the terminal and allows you to do a bunch of manuscripts at once. The second is faster for a single manuscript once you've learnt how to do it.

5.3.1 With the conversion box and RStudio

Requires RStudio and a `conversion-box` folder at the root of journal's working folder.

1. Delete any manuscript already present in the `conversion-box` folder: Word files (`.docx`, `.doc`), LaTeX files (`.tex`), markdown or quarto files (`.md`, `.qmd`). This is a temporary folder, whomever left it there has made copies.
2. Copy your original manuscript (MS Word, LaTeX) in the `conversion-box` folder. You can copy several if you want to convert multiple manuscripts in one go.
3. Rename the manuscript file(s) if necessary: manuscript filenames *must not contain spaces, colons, question or exclamation marks.*
4. Open `conversion-box.Rproj` in the `conversion-box` folder. This opens RStudio in the 'conversion box' project.

5. In RStudio, do Build > Build All, or equivalently hit Shift-Ctrl-B (Win. Linux) or Shift-Cmd-B (Mac). This converts any MS Word or LaTeX file in the folder to markdown.
6. Copy the resulting markdown file from the `conversion-box` to your article's working folder.

5.3.2 With Pandoc only

Open a terminal. If using RStudio or VSCode, there is a Terminal tab. This opens a terminal already located in your article's working folder. If you're opening a terminal from your system, you need to navigate (using `cd`, change directory commands) to your article working folder.

If you need help with using the terminal see section Appendix **??**.

Run the following command:

```
pandoc -s original/manuscript.docx -o manuscript.md
```

- `original/manuscript.docx` is the path and filename of the original manuscript. Here I'm assuming that the submission is called `manuscript.docx` and placed in a folder `original`. On Windows we use backlash to mark folders so that would be `original\mansucript.docx` instead.
- `-o manuscript.md` tells pandoc to convert to markdown (`.md`) and save the result as `manuscript.md` in your cuurent folder.
- `-s`, short for `--standalone`, tells Pandoc to produce a "standalone" document, i.e. include a header with any metadata it is able to extract from the source.

For instance, if your terminal was not located in your article working folder but in its `original` subfolder, and you wanted to write the result in the main sufolder (i.e. one folder up), you'd use instead:

```
pandoc -s mansucript.docx -o ../manuscript.md
```

Where `../` (`..\` on Windows) means "one folder up".

5.3.2.1 LaTeX manuscripts: check the bibliography field

Converting from LaTeX is the same, but **pay attention to the bibliography file location**. The resulting markdown file may have a `bibliography` key:

```
bibliography: original/references.bib

bibliography: C:/Windows/Users/Zotero/references.bib
```

Update these if needed. Here `original` is the subfolder for preserving the original: I should instead copy the `.bib` file to the main article folder and replace this key with `bibliography: references.bib`. The second one, `C:/Windows...` is a location in the author's computer as it was cited in their LaTeX file, it should be removed.

5.3.2.2 More options for Pandoc conversion

See Pandoc's Manual: reader options for more options.

6 About Markdown copyediting

NOTE TO SELF: this chapter should be reorganized. and probably divided. It has three components: - the aims of markup editing, what markdown is, and the like. Not hands-on, best moved to the publishing part. - basics of markdown syntax, 'advance warnings', using RStudio. This is hands-on, should be here. *There should also a roadmap to the copyediting chapters ahead.* - detailed prescriptions about what to encode or not. Not sure where to fit that.

6.1 Aims

In an automated copyediting workflow, the job of the copyeditor isn't to fine-tune the appearance of the output (fonts, spacing, etc.) Rather, the copyeditor's job is to **encode all, and only, the manuscript's features that the automatic converter needs to know about**.

Typical manuscripts (MS Word documents, LaTeX file) contain both too much and too little information for a publisher's purposes. A MS Word file contains information about font use and margin size, which the publisher must get rid of since they're applying their own font and margins. It also typically fails to encode crucial information in in a machine-readable way. For instance, an author may:

- use mere boldface to indicate their section titles: **1. Methods**. A human reader understands that they mean a section heading. But an automatic converter needs to know that this is section title, and not a simple bold text, to typeset it correctly.
- manually write '(Dupuis 2020, pp. 29-42)' for their citations. If an automatic converter knows that this is a citation of a given paper, at a given page range, it can turn it into a link, format it consistently throughout the journal (instead of '2020: 29–42' in some papers and '2019, pp.12-15' in others), and check that all and only references cited appear in the bibliography.
- manually write internal cross-references ("see fn. 6", "see page 8"). If an automatic converter knows that these are meant to refer to other parts of the text, they can turn them into links and adjust them if the numbering changes.

Our copyediting task is to remove unnecessary formatting, and turn important features into codes that the automatic processor can pick up and deal with.

Markdown is a light-weight, intuitive way of encoding meaningful features. For instance:

- `a *simple* solution` encodes a word that needs to be emphasized (typically, italics, but it's for the automatic converter to decide).

- `# Methodology` encodes a heading titled "Methodology".
- `[@Doe2019, 22-29]` encodes a citation of the pages 22 to 29 of an article that the converter will find under the name `Doe2019` in the article's associated bibliography file.

6.2 Style

We will rely on the Chicago Manual of Style in most cases, except where we state otherwise.

6.3 What to encode

When copyediting we need to decide which features of the original manuscript we want to keep, or *encode*, in our document. For instance, we don't want to keep the author's specific choice of fonts, or whether they italicize headings. But we do want to keep italics used for emphasis, or hyphens used to cut a word at the end of a line.

The goal is to encode *non-stylistic* features of the article, and those features only. The stylistic or merely presentational ones are handled automatically by your journal style.

The features we keep are *encoded* in markdown syntax. For instance, emphasis (italics) is encoded by enclosing words within asterisks:

```
In this sentence the last world is *emphasized*.
```

When a feature is encoded, the journal's template 'understands' it. This means that it'll be able to typeset it correctly, according to the journal style, in all the outputs formats we need—PDF, webpage, ebook.

Here's a list of features we want to encode. The list is in progress: not everything we want to encode is handled by our template yet. An X under "syntax" means that there is markdown code (aka syntax) to encode the feature in question. An X under "implemented" means that our template is able to typeset the feature in question.

Feature	Syntax?	Imple mented?	Note
Headings	Yes	Yes	
Paragraphs	Yes	Yes	
Footnotes	Yes	Yes	
Emphasis (italics, bold)	Yes	Yes	
Superscripts and subscripts	Yes	Yes	
Citations	Yes	Yes	
Cross-reference: heading	Yes	Yes	
Cross-reference: footnote	Yes	No	
Cross-reference: specific location	Yes	No	Must be handled separately in PDF vs HTML.
Cross-reference: image, figure, table	Yes	Yes	

Feature	Syntax?	Imple mented?	Note
Cross-reference: theorem, statement	Yes	Yes.	
Quotations: block quotations	Yes	Yes	Encoding the source too?
Quotations: inline	Yes	Yes	Unnecessary. Ordinary ' or " is enough.
Lists: numbered, unnumbered	Yes	Yes	
Lists: continuously numbered throughout the text	Yes	Yes	
Lists: ad-hoc numbers ((9'), (a*)…)	Yes	Yes	
Statements: simple indented blocks	Yes	Yes	
Tables	Yes	Yes	
Tables: column alignment	Yes	Yes	
Tables: cells spanning several rows or cols	No	Yes	handled by `pandoc` but not documented yet
Tables: custom borders	No	No	
Formulas	Yes	Yes	
Images	Yes	Yes	
Links	Yes	Yes	as bib entries or footnotes
Columns (incl. formulas side by side)	Yes	Yes	

Here are some features **not** to be preserved:

Feature	Note
Links in the text	Preferably as bibliography entries, otherwise in footnotes
Curly quotes	handled by `pandoc`
Special positioning of formulas, tables, …	To be avoided as much as possible

Note on **special symbols**:

- Special symbols must typically be preserved. For instance, the double square brackets should not be entered as `[[` and `]]` but with their LaTeX codes `\\llbracket` and `\rrbracket`.

6.4 About Markdown

Markdown is a syntax - a way of writing or encoding document. A markdown document is a plain text file that can be viewed in any text editor REWRITE THIS is a syntax enhanced plain text with machine-readable information, a.k.a. 'markup'. More specifically, we'll use `pandoc`'s Markdown - the variant of that syntax that is most suited to academic texts and that is fully understood by our automatic converter, `pandoc`.

The copyediting files we work on are thus in `pandoc`'s Markdown format (extension `.md`). These are just plain text documents, which can be edited with any text editor (Notepad, TextEdit, …), but where we use a certain codes to encode information needed by our automatic document converter (`pandoc`). For instance, a document may contain the line:

> In this article, I will prove that the *metaphysical* question is not nonsense.

where the * encode the fact that the word "metaphysical" is emphasised. The automatic converter can pick up on that information and typeset the word accordingly in the PDF and HTML outputs. Typically, emphasized words are typeset in italics, but we could decide to typeset them with small caps or letter spacings as well. So we only encode the fact that the word is to be emphasized, using the * notation, and leave the rest to the converter.

The official documentation of `pandoc`'s Markdown can be used to supplement the instructions below.

6.5 RStudio visual markdown editor

For background see also Quarto's page on RStudio visual mode.

When you open a markdown file in RStudio, you'll see **Source** and **Visual** tabs in the upper left corner. These are two different ways of visualizing the same document. The **Source** mode shows the document as it is, namely a plain text markdown file. The **Visual** mode shows the document as a webpage would display HTML code, or as a word processor like MS Word displays documents: text encoded with emphasis is shown in italics, text encoded as headings is shown in a larger font, etc.

Here's the **Source** mode:

Here's the **Visual** mode:

You can switch between modes by clicking the 'Source' and 'Visual' tab. Exception: RStudio may refuse to switch to **Visual** if doing so would break down some elements in your document. (This is notably the case with numbered examples.)

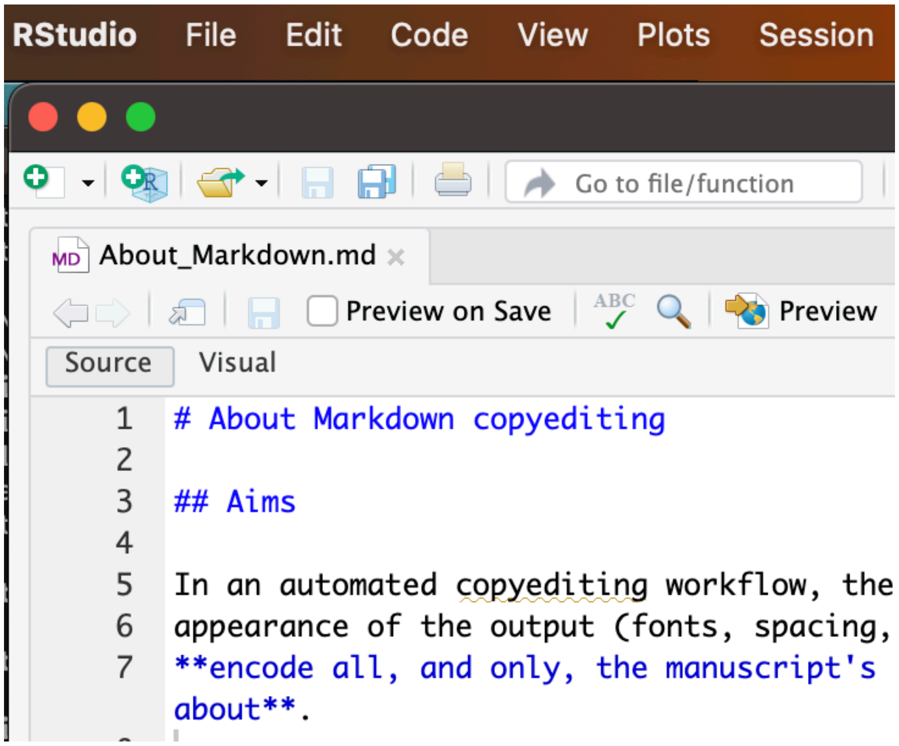
The **Visual** mode allows you to edit without using markdown code. You can insert headings, italics, formulas, tables and so on by clicking in the menus. You can also insert more complicated structural elements, e.g. a Div with its id and attributes.

The **Source** editor shows you the markdown file as it is. There's some syntax highlighting to help you visualize markdown codes.

There's a useful "outline" button at the top left to see the headings of the document.

You can still type markdown in the visual editor—it's usually faster to do so. Try it out: type `## Test` at the beginning of a line and the line will be converted to a level 2 heading. You can type formulas, enter Divs and so on.[1]

[1]Limitations: if many LaTeX formulas are inline in one paragraph the visual editor struggles to parse them

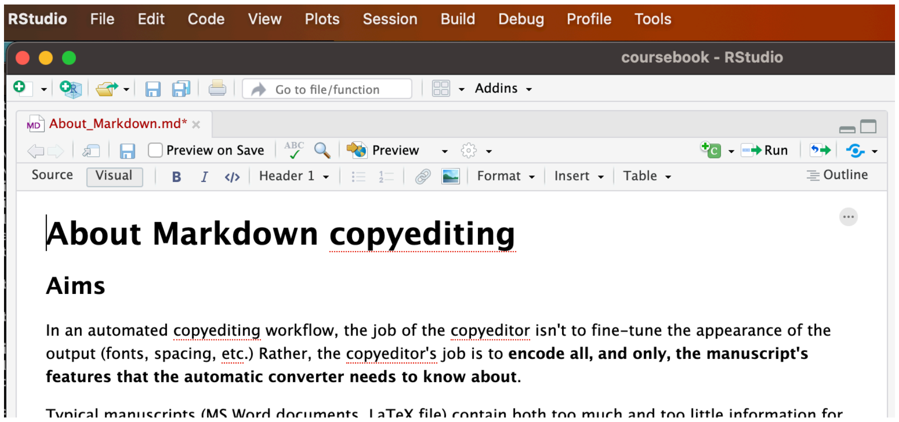


```
# About Markdown copyediting

## Aims

In an automated copyediting workflow, the
appearance of the output (fonts, spacing,
**encode all, and only, the manuscript's
about**.
```

Figure 6.1: RStudio's source mode (2023)



About Markdown copyediting

Aims

In an automated copyediting workflow, the job of the copyeditor isn't to fine-tune the appearance of the output (fonts, spacing, etc.) Rather, the copyeditor's job is to **encode all, and only, the manuscript's features that the automatic converter needs to know about.**

Typical manuscripts (MS Word documents, LaTeX file) contain both too much and too little information for

Figure 6.2: RStudio's visual mode (2023)

The visual editor is good for initial copyediting (if your document allows using it): you'll see the text and typos better. **The source editor is good final copyediting and typesetting**: you see the exact code used to generate your output and you fix special characters, line breaks, explicit LaTeX code and the like.

6.6 Heads up on special characters in markdown

In Markdown the characters below are sometimes have a special meaning:
\ & * $ [ ] _ @

Normally, you shouldn't have to worry about them. But if they aren't visible or your markdown file generates error when turned into PDF, they may be the culprit. Typically the error can be avoided by 'escaping' the character, i.e. preceding it with a backslash. Hence:

\& means & (*nb, mandatory in BibTeX files!)*

* means *

\\ means \

\$ means $

Etc.

The one exception is the square brackets. Markdown syntax allows two ways of encoding math formulas: wrapping them in $ or wrapping them in parentheses or brackets:

- `$...$` and `\(...\)` can be used to enter an *inline* formula
- `$$...$$` and `\[...\]` can be used to enter a *display* (i.e. block) formula

So if you try to espace both an opening and a closing bracket, your markdown will be read as a math formula.

Here are problem cases you might encounter:

- A reference cited is titled *Fear & Loathing in Las Vegas*. You have the title with `&` in your BibTeX file, and it generates an error.

  The problem is that in LaTeX/BibTeX `&` is a special character (to indicate tabs).

  Solution: 'escape' the ampersand in your BibTeX file:

  ```
  title = {Fear \& Loathing in Las Vegas}
  ```

- Your text has a pair of asterisks:

  ```
  Principle (*) is strong. Principle (*) is stronger than Principle (**).
  ```

  The text between the asterisks is wrongly interpreted as emphasized:

and may mangle them, turning some text into formulas and the other way round; if so you'd better off using the source mode.

> Principle () *is strong. Principle* () is stronger than Principle (**).

Solution: escape the asterisks:

```
Principle (\*) is strong. Principle (\*) is stronger than Principle (\*\*).
```

> Principle (*) is strong. Principle (*) is stronger than Principle (**).

- Square brackets for elided texts in citations. In practically all cases they'll be fine. The following markdown code will display as intended, for instance:

  ```
  The [simple] models [...] are useful
  ```

  There's only trouble in a rare case: the elided text matches a section header. For instance, suppose you have:

  ```
  The simple [models] are useful.

  ## Models
  ```

  This is wrongly interpreted as a link:

  > The simple models are useful.

  Why? Pandoc automatically assign the header "Models" a link identifier: `models` (see the chapter on cross-referencing). And if this identifier is present, `[models]` becomes a link to it. And one of the section headings is "Models".

  Solutions:

  - If you're working in markdown code, leave it as is. We run pandoc with an option that disables these links.
  - If you're working in the RStudio visual mode, add an identifier to the header, `## Models {#sec:models}` (Pandoc) or `## Models {#sec-models}` (Quarto)

7 Copyeditor workflow

Your article folder should contain a copyediting checklist and a revision tracker. Use the checklist on your first round of copyediting the article. Use the revision tracker to keep track of unfinished tasks and communications with the author (changes that you need to notify the author of, questions you need to ask them, record of their answers).

For instance, use a subfolder `history` and put in it files `checklist.md` for the checklist and `r1.md`, `r2.md` for revision rounds.

Below are examples of checklist and revision document for you to copy/paste.

7.1 Copyediting workflow

1. **Pre-conversion changes** (optional). Fix the author's MS Word or LaTeX document for better conversion results. In MS Word: use headings styles, metadata sections, and optionally use equations for formulas and special symbols. In LaTeX: usually nothing is needed, but sometimes unnecessary LaTeX commands and packages need to be removed (e.g., framed boxes).
2. **Convert to markdown**.
3. **Metadata**. Fill in the metadata.
4. **Section headings**. Check that section headings are encoded, given them identifiers for crossreferences (optional).
5. **Format elements**. Apply markdown formatting to blockquotes, lists, statements, formulas, etc.
6. **Cross-references**. Encode cross-references within the document (to section headings, footnotes, statements etc.).
7. **Citations**. Encode citations in Markdown. If using Zotero: create a (sub)collection for the paper, look up the citations online and add them to the collection using Zotero's browser plugin. Export the collection as a BibTeX file (`.bib`) in your article folder. Insert the citations in the document, ideally using a Zotero-markdown plugin for your editor (e.g. Citation Picker for Zotero in VS code).
8. Add a `# References` heading at the end of the document.
9. Render the output and proofread.
10. Go through the checklist.

7.2 The checklist

```
## Importing and Preparing the Original Document, Setting up Workspace

- [ ] Create working folder. If using Rstudio, create a project.
- [ ] Copy original manuscript in the folder (`original` subfolder).
- [ ] Prepare manuscript for conversion (headings styles, footnotes, formulas), so that pa
- [ ] Convert to markdown with `pandoc -s <originalfile> -o <outputfile.md>`.
- [ ] Create revison tracker files: create `history` folder, create a file `r1.md` in it a

## Formatting

- [ ] Complete metadata block: title, author, affiliation, abstract, thanks, bibliography,
- [ ] Sections: are they formatted correctly at all levels?
- [ ] Footnotes: are they formatted correctly?
- [ ] Quotations: are they formatted correctly? (as blocks when needed, etc.)
- [ ] Quotation marks: check that quotation marks are coherently used: single or double; p
- [ ] Quotes vs. italics: is their usage for emphasis/highlights as per Chicago Manual (sc
- [ ] Dashes: is the usage as hyphens, en-, em-dashes correct?
- [ ] Bold: avoid bold unless absolutely needed.
- [ ] Statements: check that claims, examples, theorems etc. are formatted
  as statements. Arguments with labelled premises are custom-labelled
  lists. Block equations are display maths. Use identifiers if statements or list items
- [ ] Math content: check that math content (incl. schematic letters like $A$) is coded as
- [ ] Special symbols: are they special symbols? Encode them as such (HTMLentities, copy/p

## Proofreading

- [ ] Fix minor typos and grammatical mistakes. No need to notify the author.
- [ ] Ungrammatical, poor English, otherwise flawed sentences (e.g. we can't tell what a p
- [x] No need to advise on the content (poor structure or presentation, redudant point, ob
- [ ] One final pass with spell & grammar checker

## Crossreferences and Citations

- [ ] Crossreferences: check that you have identified the crossreferences in the text and
- [ ] Create a `.bib` file from the author's bibliography
- [ ] Identify all the references in the text and replace them with keys from your .bib fi
- [ ] Use normal vs inline citations where appropriate? Use year-only sparingly. Use multi
- [ ] Is the author's bibliography correct? (wrong title, wrong year of publication, etc.)
- [ ] Add "References" heading at the end of the document.
```

```
## Rendering in output formats

- [ ] Special symbols: do they render correctly in all outputs?
- [ ] First-line indentation: paragraphs that *continue* after a block
    quote, table or list should not have an indentation. Add `\noident`
    codes where needed.
- [ ] Math and LaTeX: check that it renders correctly in HTML output. Use *Imagify* if nee
- [ ] Tables: are the tables a faithful rendering of the original tables by the author?
- [ ] Figures: do they render correctly in all outputs?
- [ ] Bibliography: does it looks good? do the citation links work?

## Send to the author

- [ ] Compile list of questions and notifications for the author.
- [ ] Have a last look at the paper.
- [ ] Send to the author for revisions and/or approval.
```

7.3 The Revision Tracker

Here's an example of revision-tracking file. We use a new one for each back-and-forth with the author.

```
## TO DO

- things you still have to do
- item

## TO ASK

- things to ask about or notify the author

## IGNORE

- things you've considered but decided against
- including author requests

## GET HELP

- things you need to ask editors
- or the template editor about
```

8 Metadata

Your markdown document should start with a 'metadata' block. This contains information about the document that isn't part of the article's body. Some of it is information about the article itself: title, author name and affiliation, DOI, etc. The rest is information needed to turn the manuscript into published output: name of a `.bib` file containing its references, special LaTeX packages to be used for its PDF output, parameters used by your template engine (e.g. 'imagify' parameters).

In this chapter, we explain how metadata entries are formatted in general. Subsequent chapters deal with particular aspects of the metadata such as the title or author's information.

8.1 The metadata block

The metadata block is at the beginning of your markdown file, and it starts and ends with lines consisting of three hyphens `---`. Here is a template - you can add this at the beginning of the markdown file:

```
---
title: The Solution to the Problem that has Plagued Many People in the Past
shorttitle: The Solution to the Problem
author: Max Mustermann
affiliation: University of Wisemen
bibliography: references.bib
---
```

In RStudio, the metadata block is displayed both in **Source** and **Visual** modes.

The metadata block consists of a series of **fields** and their **values**. In the above example:

- the fields are `title`, `shorttitle`, `author`, `affiliation`.
- the `title` field's value is a line of text, namely "The Solution to the Problem that has Plagued Many People in the Past".

The metadata block follows what's called the YAML syntax. This dictates, for instance, that field names should be immediately followed by a colon without space (`title:` and not `title :`). All you need to know about this syntax is explained in this section.

8.2 One-line field-value

In most cases you enter a field and its value in one line:

```
title: The Solution to the Problem that has Plagued Many People in the Past
```

The field's name must be immediately followed by a colon without space:

BAD	GOOD
`title:An interesting fact`	`title: An interesting fact`

8.2.1 One-line text values

The value is usually a bit of text (including a filename like `references.bib`). In most cases, they can be entered with or without quotes. The following are equivalent:

GOOD
`title: An interesting fact`
`title: 'An interesting fact'`
`title: "An interesting fact"`

The text is assumed to be in markdown: you can surround words with `*...*` for emphasis (italics), use codes for special characters, etc.

You *must* use quotes if the text value contains a colon, an apostrophe or quotation marks:

BAD	GOOD
`title: Pleasure: friend or foe?`	`title: 'Pleasure: friend or foe?'`
`title: Lars' revenge`	`title: "Lars's revenge"`
`title: A "good" friend`	`title: 'A "good" friend'`
`title: A 'good' friend`	`title: "A 'good' friend"`
`title: My Life: a Long Story`	`title: "My Life: A Long Story"`

As you can see, if a text value contains single quotation marks, it must be put within double quotation marks and conversely. The apostrophe is entered using a single quotation marks, so a title containing one should be wrapped in double quotation marks.

If your value happens to contain both kinds of quotation marks, you can enter as a text block value or wrap it within one kind of quotation marks but 'escape' marks of that kind with a backslash whenever they appear in the title:

```
title: 'A \'quote\' and "another" in this title'
```

8.2.2 Number and boolean values

In some cases a value may be a number (`12`) or a Boolean value (`true` or `false`). Enter they without quotation marks.

```
volume: 75
```

8.3 Text block values

Another way of entering text values is to use the *text block* syntax. This is useful when the value is a longer text, e.g. an abstract, or when the value contains special characters. The basic syntax is:

```
title: |
  The Solution to the Problem that Plagued Many People in the Past
```

- One line contains the field name followed by a colon, space and the pipe symbol `|`.
- One or more subsequent lines contain the text. *Each line must be indented by 2 to 4 spaces relative to the field's name.*
- No quotation marks are used. Any added quotation marks will be treated as part of the text itself.

BAD	GOOD

`title: | A longish`
`    phrase`
	`title: |`
	`  A longish phrase`

`title: Another longish`
`    phrase`
	`title: |`
	`  Another longish phrase`

`title: |`
`    Value spread on several lines`
`but I forgot indentation`
	`title: |`
	`  Value spread on several lines`
	`  indented properly`

`title: |`
`    "This wasn't mean to have`
`    quotation marks!"`
	`title: |`
	`  With "quotation" marks and colons:`
	`  no problem!`

Even though text block values can occupy multiple lines, *they are not a way to specify line breaks.* For instance, the following:

```
title: |
  A long title that would without a doubt
  end up occupying more than one line
```

is equivalent to:

```
title: A long title that would without a doubt end up occupying more than one line
```

Using the former is not a way to force the template engine to break the title at that line.

In general, you shouldn't enter forced line breaks in titles or abstracts anyway. Your journal template's may allow you to enter manual linebreaks for a better look on the cover page or chapter page. These are not entered in the `title` field but in some separate title fields. (For instance, Dialectica uses `title-cover`.) See the instructions specific to your journal.

8.4 How to spot syntax errors in your YAML metadata block

- smart editors like VSCode or RStudio color the metadata block to distinguish keys and values, and sometimes underline in red places where the editor thinks you committed a mistake.
- you get a "YAML Parse error" when creating outputs. This states a line where an error was encountered (the mistake will be on that line or the previous ones).
- the metadata block is printed out in your output formats instead of being processed. This means it's not even recognized as a metadata block at all: check that it begins and ends with `---` lines.

8.5 Linebreaks: genuine, merely visual and explicit

Can you spot the error in this metadata block?

```
title: The Formalization of arguments with various
applications
author: Robert Michels
```

The problem is that the title is spread over two lines: it should either be all on one line or use the text block syntax (`|`).

To avoid such errors it's important to distinguish:

1. Genuine line breaks in your markdown file,
2. Merely visual line wrapping, and
3. Explicit line breaks to be printed out in your output

And, to distinguish these, you need a text editor that displays line numbers—smart editors like VSCode, RStudio, Sublime etc. all do.

A markdown file, like any text file, is simply a long string of characters. Most of them are letters, numbers and symbols. But some are *linebreak* characters: they denote the end of a line rather than a letter, number or symbol. Using ‘ ’ to represent this special character, a simple text file could be:

```
now then, let's go out to enjoy the snow... until I slip and fall!
```

This is how the computer ‘sees’ it. For us, though, it’s hard to read. So, your text editor displays these line breaks characters by starting a new line instead, which it indicates with a new line number:

```
 1 now then, let's go out
 2 to enjoy the snow... until
 3 I slip and fall!
```

These are *genuine line breaks*, in the sense that they reflect what’s actually in the file.

There is no limit to the length of (genuine) lines in a file. For instance, a two-line file may be:

```
this file starts with a very very very very very very very very very very very very long lin
```

The first line of this file is likely to be longer than the width of your text editor. If so, your editor faces a dilemma: should it show it to you as one line, in which case you’ll need to scroll to the right to read it? Or should it “wrap” the line, i.e. split it when it reaches the right end of the editor’s window, in which case what you see won’t exactly correspond to what’s in the file? Editors normally offer both options—usually in a menu called “View”: Wrapping, or No Wrapping.

But editors with line numbers give you the best comprise: they only have line numbers where there are genuine linebreaks. Without wrapping, they show you the file above like this:

```
1 this file starts with a very very very very very very very very very very very very long l
2 a short one
```

So you see that there are only two lines; the first is long and probably overflows your editor’s window to the right. In Wrapping mode, depending on window size they might show the file as follows:

```
1 this file starts with a very very very very very very very very
  very very very very long line followed by
2 a short one
```

You can see that the second line isn’t a *genuine* line break in the file because it doesn’t have a line number. It’s really the rest of the first line, merely *displayed* below it so you don’t have to scroll right to see it. By contrast, “a short one” is a genuine new line.

Back to metadata blocks. In an editor with line numbers, you can see the difference between:

```
1 ---
2 title: The Formalization of arguments with various
  applications
3 author: Robert Michels
4 ---
```

and:

```
1 ---
2 title: The Formalization of arguments with various
3 applications
4 author: Robert Michels
5 ---
```

Both try to provide the `title:` value in one line (they don't use the `|` of text blocks). The first one is correct. As line numbers show, our title is indeed given in one line (line 2). The editor merely *displays* "applications" on a subsequent line, for our convenience. What's really in the file, what the computer 'sees', is just one line. So, there's no error there. In the second, however, we have mistakenly entered a genuine linebreak between "various" and "application", as shown by the new line number 3. So, what the computer reads is:

- the title is "The Formalization of arguments with various"
- then there's a line "applications". It's not a field name, because it's not followed by a colon (it's not: `applications: ...`). So, what is it?

And you get a "YAML Parse error" message, which is to say that the computer doesn't understand the structure of your metadata block.

Genuine line breaks in your markdown files are not *explicit line breaks*, however. Recall the file containing Bashō's haiku:

```
now then, let's go out to enjoy the snow... until I slip and fall!
```

It has three genuine line breaks, which your text editor will show thus:

```
 1 now then, let's go out
 2 to enjoy the snow... until
 3 I slip and fall!
```

In markdown, however, *linebreaks are just treated as spaces.* So, if you create an output from this file, you'll get a single paragraph:

> now then, let's go out to enjoy the snow… until I slip and fall!

When turning markdown into PDF or HTML output, there's no attempt to match the lines in your markdown file. That's good because you normally don't want to manually decide where to break the lines. The only thing you want to manually enter are the *paragraph* divisions, and these are just entered by two successive linebreaks, i.e. an empty line:

```
1 This is a paragraph containing one or more
2 sentences. Line breaks in the markdown file are
3 treated as spaces, so the final output may
4 break lines elsewhere than shown here.
5
6 However, it will treat this as a second paragraph
7 because it is separated from the previous
8 by a (genuine) empty line, i.e. two successive
9 linebreaks.
```

In some cases though, like poetry verses, you want to specify *explicit* line breaks that aren't paragraphs. These are entered by 'escaping' the linebreak character, i.e. having a backslash followed by the linebreak character. In the text editor:

```
1 now then, let's go out\
2 to enjoy the snow... until\
3 I slip and fall!
```

Which corresponds to the following file:

```
now then, let's go out\ to enjoy the snow... until\ I slip and fall!
```

In markdown, escaped characters are reproduced as they are. Thus an escaped linebreak is reproduced as an *explicit linebreak* in your output. Beware that the `\` should be immediately followed by the end of the line, not a space: otherwise it's just an *explicit space* and all you'll see in your output is an extra space.

In summary:

1. *Merely apparent linebreaks* are lines starting with no numbers in your editors. They correspond to nothing real in the file; they're just a convenient way of displaying a long line.
2. *Genuine linebreaks* are linebreaks characters in the file, corresponding to new *numbered* lines in your editor. They are important in metadata block. In the main text, they are treated as spaces, so you can cut your lines whenever your lines. The exception is two linebreaks in a row, i.e. an empty line, which mean a new paragraph.
3. *Explicit linebreaks* are backslash `\` at the very end of a line. They represent forced linebreaks that must be reproduced in the output.

Terminological note: hard vs soft line wrapping. You may encounter the terms of "hard" and "soft" line wrapping. Here's what they mean.

- *Hard line wrapping* is entering *genuine linebreaks*. For instance, your editor may offer the option of reflowing your text with linebreaks at a certain fixed length. This will normally be in the "Edit" menu, as it changes the file itself.

- *Soft line wrapping* is merely *displaying* long lines on multiple lines. Text editors offer a soft wrapping option, which allows you to view long lines without having to scroll horizontally. This will normally be in the "View" menu, as it doesn't modify the file itself but only how you view it.

General comment. The beauty of working with text files such as markdown files is that there is nothing hidden: what you see in the editor is exactly what there is on file. The only thing that's not immediately obvious is genuine linebreak characters. Now that you know how to use an editor's line numbers to identify them, markdown files have no secret for you.

To see what we mean by 'hidden', compare with Word files. When you see an italicized word *fortiori* in a MS Word document, what there actually is in your `.docx` file is a complicated bit of code such as this:

```
<w:r><w:rPr><w:i/><w:iCs/><w:lang w:val="en-US"/></w:rPr><w:t>fortiori</w:t></w:r>
```

Because you can't see what's going on 'under the hood' when you modify a file in MS Word, it's virtually impossible to do proper typesetting. For that you'd need to see the file as it is; but as the example above shows, they're virtually impossible to read when seen that way. Markdown gives us the benefit of working with a humanly readable file and at the same time working with the file exactly as the computer 'sees' it.

8.6 List and map values

In addition to simple values like a piece of text or a number, a metadata field's value can be a *list* or a *map*.

8.6.1 Lists

A list is a sequence of values. For instance, the `keywords` field is a list of text values:

```
keywords:
- pleasure
- pain
- life
- death
```

Each item in the list is on a separate line, starting with a dash (`-`). The dashes are aligned with the field's name (`keyword` here). Quotation marks can be used, as with one-line text values (**ec-one-line-text-values?**).

An alternative, shorter way to enter a list of one-line values is to use square brackets and commas, as in:

```
keywords: [pleasure, pain, life, death]
```

That is equivalent to the above; again quotation marks can be used. It cannot be used for more complex list values such as text blocks or maps.

List items can be text blocks too, and you can mix text blocks and one-line values:

```
keywords:
- |
  A we"ird key'word with:symbols
- |
  A keyword that I would like
  to enter on multiple lines
- this item is just a line
```

A list item can also be a map. Let's explain what these are first.

8.6.2 Maps

A *map* or *dictionary* is a set of field-value pairs. The metadata block itself is a map:

```
title: The Meaning of Life
author: Jane E. Doe
bibliography: references.bib
```

But a field's value can itself be a map. We indicate this by indenting the map under the field's name. This is typically used to specify options for a template extension like Imagify (used to turned LaTeX into images):

```
title: Yet Another Mistaken Theory
author: Jane E. Doe
imagify:
  scope: all
  output_folder: _imagify_outputs
bibliography: references.bib
```

Here `imagify`'s value is a map with two fields: `scope` and `output_folder`. Note how these entries are indented and below `imagify`. By contrast, `bibliography` is aligned completely to the left, with `title`, `author` and `imagify`, so it's a main field, not a sub-field of `imagify`.

Sub maps can contains sub-sub maps and so on:

```
field1:
  subfield1:
    subsubfield1: value
    subsubfield2: value
  subfield2: value
field2: value
```

Any of the values can be a list. If it is, align the list's dashes with the field's name. For instance, we can make the value of `subsubfield1` a list of two values as follows:

```
field1:
  subfield1:
    subsubfield1:
    - first value
    - second value
    subsubfield2: value
  subfield2: value
field2: value
```

A map cannot have two fields with the same name:

```
BAD

author: Jane Smith
author: John Doe
```

However, the same field name may appear in *distinct* maps, including a submap and the main map, or distinct maps in a list of maps.

8.6.3 Maps in lists

A list item can be a map. The `author` field is a case in point:

```
author:
- name: Vincent Conitzer
  email: conitzer@cs.duke.edu
  correspondence: true
  institute: Duke University
- name: Nadine Elzein
  email: nadine.elzein@warwick.ac.uk
  institute: University of Warwick
```

Let's break this down. Our `author` field is a list of two values:

```
author:
- first value
- second value
```

Each of these values is itself a map. The second value is the following map, for instance:

```
name: Nadine Elzein
email: nadine.elzein@warwick.ac.uk
institute: University of Warwick
```

When we insert it in the list, *we make sure we indent the map* so that each of its fields is aligned to the right of the `-`:

```
author:
- first value
- name: Nadine Elzein
  email: nadine.elzein@warwick.ac.uk
  institute: University of Warwick
```

Note how `name`, `email` and `institute` are all aligned, and to the right of the second dash. Doing the same with the map corresponding to the first value we obtain the desired result:

```
author:
- name: Vincent Conitzer
  email: conitzer@cs.duke.edu
  correspondence: true
  institute: Duke University
- name: Nadine Elzein
  email: nadine.elzein@warwick.ac.uk
  institute: University of Warwick
```

Note how each of the fields `name`, `email`, `institute` appear twice but in *distinct* maps.

Values in maps can be lists too, as we said. The `institute` field is a case in point:

```
author:
- name: Susan Stebbing
  institute:
  - "King's College London"
  - Bedford College, London
- name: Someone Else
  institute: Some University
- name: Third Person
```

```
    institute: Some College
```

Here `author` is a list with three values, corresponding to the left-most dashes. Each is a map with two fields, `name` and `institute`. Note how the dashes in front of "King's College" and "Bedford College" are aligned with the `institute` field name.

8.6.4 List in lists

Though we haven't found a use for them in copyediting metadata, lists whose values are lists are also possible:

```
mylistoflists:
- - item1
  - item2
- - item3
  - item4
```

Here `mylistoflists` is a list with two values; the first value is itself a list with two values (`item1`, `item2`), and the second is another list with two values (`item3`, `item4`).

9 Title and subtitle

Title and (optionally) subtitle of an article are entered in the metadata `title` and `subtitle` field:

```
title: The Solution to the Problem that Plagued Everyone
subtitle: A Brand New Discovery
```

Titles of the form "Time Travel: a Quantum Relativity Puzzle" should be split into title and subtitle. Exceptions are cases where splitting wouldn't make sense, e.g. "Pleasure: friend or foe?".

If a title or subtitle contains colons or quotation marks, it should be either wrapped within quotation marks or entered as a text block. See Section **??**.

The `title` and `subtitle` fields should *not* contain explicit linebreaks. (See Section **??** on what explicit linebreaks are.) Not only it's best to let the reader's device set the linebreaks in 'fluid' outputs (HTML, EPUB), but these fields are also used for the table of contents and to send information to bibliographic database. They should thus be free or unnecessary formatting.

Sometimes, however, you need alternative forms of the title:

- a short title to be used in page headers and/or the table of contents.
- a title with explicit line breaks, when the title is long and default line breaks don't look good in PDF output.

For these, your journal's template should provide extra title fields. Dialectica, for instance, uses `shorttitle` if a shorter version is needed for page headers and the table of contents, `title-latex` if we need specific linebreaks on the article's first page in PDF output, and `title-cover` if we need different linebreaks on the offprint cover than on the article's first page.

Your journal's template may also have extra fields to handle how titles and subtitles appear in the table of contents. For instance, Dialectica shows both title and subtitle in the table of contents, separated by a colon. As the colon isn't appropriate when the title ends with a question mark or other punctuation, the template provides a `subtitleseparator` field that can be used to replace the colon with a space.

For details see your journal template's guidelines.