

YuMD

A Unified System of Notetaking and Flashcard Creation

James Yu

15th December 2021

Contents

| | | |
|---|---|----|
| 1 | TL;DR | 2 |
| 2 | Introduction | 3 |
| | 2.1 Why? | 3 |
| | 2.2 What? | 4 |
| 3 | Syncing Notes with Anki | 4 |
| | 3.1 Preliminary Knowledge and Preparation | 4 |
| | 3.2 Sending Notes | 4 |
| | 3.3 Note IDs | 5 |
| 4 | VSCoDe Extension | 6 |
| | 4.1 Preview | 6 |
| | 4.2 VSCoDe Extension Settings | 7 |
| 5 | Compiling LaTeX | 7 |
| 6 | Extended Syntax | 8 |
| | 6.1 Theorems | 8 |
| | 6.2 Definition Lists | 10 |
| | 6.3 Tables | 10 |
| | 6.4 Footnotes | 13 |
| | 6.5 References | 14 |
| | 6.6 A Section | 15 |
| | 6.7 Citations | 16 |
| | 6.8 Smartypants | 16 |
| | 6.9 Including other .yumd files | 16 |
| | 6.10 Introduction | 17 |
| | 6.11 Conclusion | 17 |
| | 6.12 LaTeX Math | 17 |
| | 6.13 Figures | 18 |
| | 6.14 Unsupported: Native Code | 19 |
| 7 | Defining Anki Flashcards | 19 |
| | 7.1 Explicit Flashcard Generation | 19 |
| | 7.2 Default Deck | 20 |
| | 7.3 Inferred Fields | 20 |
| | 7.4 Inferred Note Type | 21 |
| | 7.5 Occlusion (Cloze Flashcards) | 23 |
| | 7.6 Implicit Flashcards | 25 |
| 8 | YuMD File Configuration | 28 |
| | 8.1 A Complete Configuration | 28 |
| | 8.2 General Options | 30 |
| | 8.3 Document-Level Options | 31 |
| | 8.4 HTML Output-Specific Options | 31 |
| | 8.5 LaTeX Output-Specific Options | 31 |
| | 8.6 Anki-Specific Options | 31 |
| 9 | Appendix | 32 |
| | 9.1 Quality of Life Improvements | 32 |
| | 9.2 Smartypants Replacements | 34 |
| | 9.3 Known Issues and Annoyances | 34 |
| | 9.4 Hardcoded Preamble | 35 |

1 TL;DR

YuMD is an extension of Markdown allowing for the typesetting of rich and complex notes, with first-class support for flashcard creation from directly within your notes. YuMD lets you

1. create and directly sync flashcards between .yumd notes and Anki, avoiding the need to write out notes multiple times,
2. export notes into LaTeX to create well-typeset PDFs, and
3. (for advanced users) export notes into HTML files for further processing.

A full reading of this documentation is recommended if you want to leverage the full potential of YuMD, but if you are short on time, it is essential to read Section 3 and Section 7. A skim through Section 6 and Section 8 is also highly recommended.

Note 1.1. Like Python, YuMD relies critically on correct indentation to determine the structure of your notes. Make sure things are aligned correctly. Importantly, it is **strongly recommended using spaces rather than tabs to indent**. YuMD treats all tab characters as 4 spaces, which can easily lead to misunderstanding between you and the parser. **It may help to enable Indent Using Spaces, specifically 2 spaces in VSCode** (don't worry: this setting will be specific to .yumd files and you won't need to constantly switch indentation settings whenever you work with other files).

Example 1.1 (the importance of correct indentation). Correct indentation allows you to create complex and nested structures in your document. How much indentation is required is quite intuitive within YuMD. For example, to create multiple paragraphs within a list item, each paragraph must begin at the same level of indentation as the first paragraph. The following example combined with some experimenting by yourself should be enough for a solid understanding:

```
1. a list
1. with a complex list
   item,

   containing multiple
   paragraphs, and a
   theorem:

Theorem.
: The statement
  "indentation does
  not matter" is false.

Proof.
: Attempting the
  following will not
  produce a theorem
  within a list, since
  the `:` is not
  indented far enough:

1. Hi

   Proposition.
   : This will not work

   Proposition
   : while this will.
```

```
1. a list
2. with a complex list item,

   containing multiple paragraphs,
   and a theorem:

Theorem 1.1. The statement
'indentation does not matter' is
false.

Proof. Attempting the following
will not produce a theorem within
a list, since the : is not indented
far enough:

(a) Hi

   Proposition. : This will not
   work

Proposition while this will.
```



Example 1.2. This documentation is itself created purely using YuMD. A skim through the [.yumd source](#) can prove very helpful in quickly familiarising yourself with the versatility of YuMD despite its simplicity.¹

Tip. If line wrapping is enabled in the text editor, levels of displayed indentation could become misleading. It is recommended setting the VSCode setting `Editor: Render Whitespaces` to `selection` to help identify the actual number of spaces in the file.

Note 1.2. For those new to Markdown, [this page](#) provides a quick reference to basic syntax. YuMD extends this to allow for much more complex note creation. This is detailed in Section 6.

2 Introduction

2.1 Why?

YuMD was created as a solution to the following trilemma.

Proposition 2.1 (Trilemma of notetaking). It is impossible to have the following conditions at the same time from the same base of notes:

1. ability to produce a good-looking document, preferably in PDF form (such as using LaTeX)
2. creation of flashcards
3. fast notetaking with support for complex elements including tables, theorems, figures, etc.

Partial disproof. While highly imperfect, and undoubtedly full of small bugs, YuMD attempts to solve this. ■

Existing solutions only allow for up to two conditions in Proposition 2.1 at one time:

1. Notion’s intuitive UI satisfies conditions 1 and 3 and allows for quick notetaking with mildly good-looking formatting but offers no way of producing flashcards,
2. Remnote satisfies conditions 2 and 3 allows for fast notetaking with first-class support for flashcard creation, but has an ugly UI, ugly formatting, and offers no way of collating all notes in one document.
3. Obsidian has a half-baked extension to create flashcards, so satisfies conditions 1 and 2 to some extent, but this, along with any non-standard element necessitates extremely verbose syntax. Complex, nested structures of theorems, tables, etc. are out of the question.

Some solutions are even worse and only satisfy one condition:

1. ‘vanilla’ Markdown ([CommonMark](#), [Github Flavoured Markdown](#), etc.) is hugely limited and has substandard or non-existent support for tables, theorems, and many other things necessary for rich notes, while
2. using pure LaTeX creates good-looking, highly-complex documents, but is a pain to type, and some complex formatting requires complex workarounds and hours of hair pulling (cough cough nested tables cough cough).

Until now, the most realistic solution has been to create multiple copies of notes: once in a notetaking program such as Notion or directly in LaTeX, and again by *rewriting all the notes* in Anki.

YuMD aims to solve this by

1. extending Markdown to include theorems, tables, footnotes, citations, references, and other important features (see Section 6), and
2. offering functionality to:
 - (a) computationally generate LaTeX (and HTML) code, saving huge amounts of time typing, debugging, and searching Stack Overflow, and
 - (b) identify, send and sync flashcards to Anki from within your notes.

¹unfortunately, copying and pasting code from this PDF won’t preserve spacing, but feel free to use the [.yumd source](#) as a reference.

In sum, YuMD allows for complex and rich note content, while providing first class support for flashcard creation. At the same time, YuMD's LaTeX output allows you to create a good-looking final document while skipping the hair-pulling involved in writing actual LaTeX. YuMD is thus also useful for those who do not need the flashcard feature: typesetting reports and supervision work is a breeze with YuMD.

Note 2.1. Everybody studies in a different way. YuMD was created with my own way of studying and revising in mind, which could be very different from your way of studying. Nevertheless, it the hope that YuMD can help as many people as possible. Particularly, YuMD may be of help for those who like to type up notes and utilise electronic flashcards.

2.2 What?

The term YuMD refers broadly to the notetaking system outlined above. Physically, it refers to the file type `.yumd` (although all text files are accepted by YuMD regardless of the extension), and its physical manifestation as a computer program.

As a computer program, YuMD is a C++ library compiled for Windows (`yumd.dll`) and macOS (`libyumd_osx_intel.dylib`). YuMD ships with:

1. an executable (`yumd.exe` and `yumd_osx_intel`),² which provides a command-line interface³ to this library, and
2. a VSCode extension (`yumd.vsix`).

The binaries and VSCode extension can be obtained freely [here](#), although source code is not public for now.

The VSCode extension uses both the library and the executable to provide:

1. real-time scroll-syncing preview with source mapping,
2. document outlines,
3. folding ranges,
4. commands to generate `.tex` and `.pdf` output, and to send notes to Anki, and
5. (primitive and at times inaccurate) syntax highlighting, (see [known issues](#))

Note 2.2. For now, YuMD will stay closed-source, but this decision is open for review in future.

Note 2.3. There is no current plan to compile native binaries for Apple Silicon. Macs with Apple Silicon should nonetheless be able to run the Intel binaries through the Rosetta 2 compatibility layer.

3 Syncing Notes with Anki

This section details the procedure for sending cards to Anki and syncing between YuMD notes and Anki.

3.1 Preliminary Knowledge and Preparation

For a rundown on the syntax for defining flashcards, see Section 7.

The **AnkiConnect** addon is required to allow YuMD to communicate with Anki, which can be obtained [here](#). Please leave the port `localhost:8765` reserved for **AnkiConnect**.

3.2 Sending Notes

Before sending notes to Anki, *ensure that Anki is open and the AnkiConnect addon is installed*. It is also recommended not having the **Browse** window open in Anki, since viewing flashcards while changes are being made can cause glitches within Anki.

²I am lying. The extension also ships with a version compiled for Ubuntu on WSL (the library `libyumd.so` and executable `yumd`). It is highly recommended ignoring this.

³see Section 9

To send your flashcards to Anki, use the command **YuMD: Send Notes to Anki**, which can be found in the toolbar menu, shown in Fig. 3.1, or typing **Shift+Ctrl+P** (Windows) or **Shift+Command+P** and typing ‘YuMD: Send Notes to Anki’. It might be helpful to assign a keybinding to this.

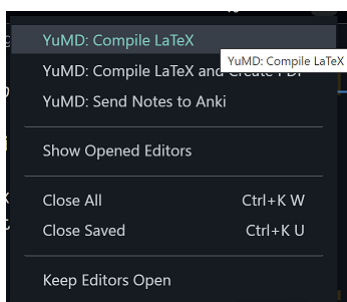


Figure 3.1: Sending notes to Anki through VSCode.

Alternatively, notes can be sent to Anki via the command line using:
`/path/to/yumd_executable path/to/document.yumd --anki`

Note 3.1. The **YuMD: Send Notes to Anki** command works by sending this command to the integrated terminal. It is critical that there **isn’t already text on the current line of the terminal** since this will prevent the command from working correctly.

Note 3.2. You may at times be confronted with the error message **Open a YuMD file to send to Anki**. This is because the corresponding `.yumd` file was not in focus when sending the command. This is a drawback of VSCode’s API: it is difficult to determine visible text editors. Be sure to *first select the text editor corresponding to the desired YuMD file before using the command*.

3.3 Note IDs

After sending your notes to Anki for the first time, YuMD will tag flashcards with a note ID within your YuMD file. This allows YuMD to keep track of created flashcards.

```

69 Definition (complementary goods). ##1639238088374
70 : Good \(\mathbf{i}\) is a complement for good \(\mathbf{j}\) at price \(\mathbf{p_j}\) if
71   \[
72     \mathbf{p_{dv}\{x_i^*\}\{p_j\} < 0.}
73   \]
74

```

Figure 3.2: Note ID added by YuMD after sending flashcards to Anki. Avoid touching.

Note 3.3. Avoid tampering with the note ID as this is the only thing pointing YuMD towards the flashcard in Anki.

Editing flashcards To change the content of a flashcard, simply make the change within your YuMD document and rerun **YuMD: Send Notes to Anki**. Using the note ID, YuMD will update the corresponding flashcard in Anki rather than creating a new one.

Note 3.4. Never make changes to flashcards within Anki itself. Direct changes within Anki will be overwritten the next time you run **YuMD: Send Notes to Anki**.

Deleting flashcards To delete a flashcard, append the string `delete` *after* the note ID, as shown in Fig. 3.3, and rerun YuMD: Send Notes to Anki. Upon deletion, YuMD will replace the note ID with the tag `ignore`, as in Fig. 3.4, which prevents it from re-creating the flashcard the next time you run YuMD: Send Notes to Anki.

Append ‘delete’ after the note ID to mark it for deletion.

```
69 Definition (complementary goods). ##1639238088374delete
70 : Good \(\) is a complement for good \(\) at price \(\) if
71 \{
72   \pdv{x_i^*}{p_j} < 0.
73 \}
74
```

Figure 3.3

After deletion, YuMD automatically replaces the tag with ‘ignore’ to avoid recreation of the flashcard upon next run.

```
69 Definition (complementary goods). ##ignore
70 : Good \(\) is a complement for good \(\) at price \(\) if
71 \{
72   \pdv{x_i^*}{p_j} < 0.
73 \}
74
```

Figure 3.4

Note 3.5. Avoid deleting notes directly within Anki without manually changing the flashcard tag to `ignore`.

4 VSCode Extension

4.1 Preview

The VSCode extension provides real-time preview by running the command YuMD: Show YuMD Preview, or by clicking the preview icon shown in Fig. 4.1

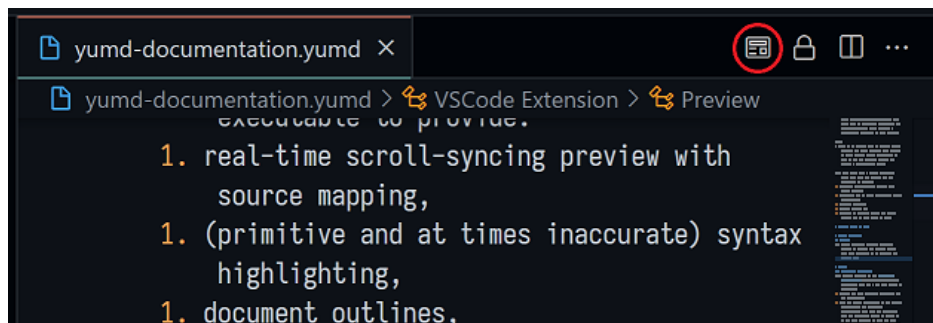


Figure 4.1: Click to show YuMD preview.

This preview leverages YuMD’s HTML output and updates as you type.

Critically, source mapping means that the preview is aware of exactly which line of code each element came from. This allows for scroll syncing between the `.yumd` text editor and the preview webview.

Tip. **Ctrl-** or **Command-**clicking an element in the preview jumps to the corresponding line of code in the corresponding `.yumd` file. This can prove hugely useful for quickly navigating through your notes.

Locked Preview Locked mode can be toggled using the command YuMD: Toggle Lock YuMD Preview, or toggling the lock button next to the preview button.

When using multiple `.yumd` files, the preview will by default change to render each file whenever you switch files in the text editor. Locked mode prevents this, keeping the preview fixed on the `.yumd` file to which it was locked.

Additionally in locked mode, the preview reads only what is saved to disk, updating on save, rather than on type.

Tip. For large documents with lots of math, or for slow computers, YuMD preview will begin lagging behind due to the slow MathJaX render. It is recommended enabling locked mode to produce previews on save, which will substantially reduce the lag. [Splitting the file](#) and only previewing subfiles could also help.

Note 4.1. The YuMD preview is far from an exact representation of the final LaTeX output. Things like section and equation numbering are non-existent within the HTML output. Other things like table of contents can only be enabled within LaTeX output (see [Section 8.5](#)). The preview is intended to function as a preview offering assurance that YuMD is interpreting your notes correctly and as a quick way of navigating through your notes.

Example 4.1 (single and double quotes). One difference between the two types of output is how they handle single and double quotes.

Rendered LaTeX ignores whether you have used single or double quotes. By default, outer quotations will be typeset using single quotation symbols, even if they are written using double quotes, and inner quotation marks will produce double quotation symbols. This can be switched by setting `language` to `american` (see [Section 8.1](#) and [Section 8.5](#)), with the side effect of changing hyphenation rules and everything `babel`-related.

In HTML on the other hand, single and double quotes are respected.

| | |
|---|---|
| "this is surrounded by 'single' quotes" | 'this is surrounded by "single" quotes' |
|---|---|

4.2 VSCode Extension Settings

The YuMD VSCode extension provides a small range of settings, mostly related to the preview.

It is highly recommended changing the default font, by searching for `YuMD: Preview Font Family` within VSCode settings. The font `"Stix Two Text"` is highly recommended.

A custom CSS stylesheet is also supported if you are (undoubtedly) unhappy with the default styling offered. For purposes of brevity, a documentation of styling is not included here. Instead, we would recommend running

```
/path/to/yumd/executable path/to/document.yumd --html path/to/output.html
```

and inspecting `path/to/output.html` to investigate how to style the rendered elements.

Note 4.2. Unfortunately, you must close and reopen the preview whenever you change a preview-related setting for it to take effect.

5 Compiling LaTeX

To compile YuMD notes into LaTeX, the VSCode extension provides the command `YuMD: Compile LaTeX`. Using this, you are free to post-process the outputted `.tex` files however you want. Alternatively, the command line can be used:

```
/path/to/yumd/executable path/to/document.yumd --latex path/to/desired/output/directory
```

If you do not plan to do your own post-processing, and simply want a PDF file, you can use the command `YuMD: Compile LaTeX and Create PDF`, which compiles YuMD into LaTeX and then runs `pdflatex` \rightarrow `biber` \rightarrow `pdflatex` to generate the PDF file.

Note 5.1. When compiling into LaTeX, YuMD copies dependencies (images and bibliographies) into the target directory, preserving the directory structure.

Note 5.2. See [Note 3.1](#).

6 Extended Syntax

This section details the additional components introduced by YuMD which make it amicable for rich note-taking and even report writing.

CommonMark Conformity YuMD largely conforms to the [CommonMark specification](#), save for a few edge cases. In particular:

1. unicode and HTML character references are not implemented,
2. tabs are always interpreted as 4 spaces, contradicting the CommonMark specification that tabs are translated into the equivalent number of spaces up to the next multiple of 4. Note [1.1](#) cannot be stressed enough: **all indentation in YuMD should be done using spaces to avoid unexpected results.**

Proposition 6.1. This behaviour will be reviewed in a future update, since rounding to the next multiple of four can be implemented trivially.

6.1 Theorems

YuMD makes it easy to create theorems, definitions, proofs, corollaries, etc., in the style of mathematical textbooks. Blasphemous as it is, all of these environments are referred to internally as theorems.

The syntax for a theorem is:

```
<theorem identifier> (<theorem name>).  
:  
  <block content>  
  <block content>  
  
  <block content>
```

or

```
<theorem identifier>.  
:  
  <block content>  
  <block content>  
  
  <block content>
```

Note 6.1. To correctly set out a theorem,

1. the period . after the closing bracket or theorem identifier (if no theorem name) is critical,
2. the colon : must be on a newline and at the same indentation level as the theorem identifier, and must immediately proceed a space,
3. content within the theorem must be aligned with the first block element after the colon, and
4. unless it is the first line of the file, there must be a blank line before the theorem to separate it from the previous paragraph.

<theorem identifier> is case-insensitive and can be any of

1. `assumption` or `ass`,
2. `axiom` or `axm`,
3. `conjecture` or `cnj`,
4. `corollary` or `cor`,
5. `definition` or `def`,
6. `derivation` or `drv`,
7. `example` or `exm`,
8. `exercise` or `exr`,
9. `hypothesis` or `hyp`,
10. `lemma` or `lem`,

11. `note` or `not`,
12. `problem` or `prb`,
13. `proof` or `prf`,
14. `proposition` or `prp`,
15. `question` or `que`,
16. `remark` or `rmk`,
17. `solution` or `sln`,
18. `theorem` or `thm`, and
19. `tip`.

`<theorem name>` is optional must be placed in brackets after a space after the theorem type and before the period. This is useful for naming theorems and specifying terms to define, and allows for [quick flashcard creation](#). The exception here is with the proof environment, where, by convention, the entire ‘*Proof*’ text is replaced with the text given in brackets. This is handy in defining your own ‘proof’ environments.

Theorem (Fermat's little).
: If (p) is a prime
number, then for every
integer (a) ,

$$\begin{aligned} & \left[\right. \\ & \quad a^p \equiv a \pmod{p}. \\ & \left. \right] \end{aligned}$$

Prf.
: Left as an exercise to
the reader.

Theorem 6.1 (Fermat's little). *If p is a prime number, then for every integer a ,*

$$a^p \equiv a \pmod{p}. \quad (6.1)$$

Proof. Left as an exercise to the reader. ■

Definition (economics).
: Too many to list.

Proof (Why too many).
: Ask Cambridge lecturers
and you will either
1. receive a different
answer to the
previous professor or
1. receive no answer.

Definition 6.1 (economics). Too many
to list.

Why too many. Ask Cambridge
lecturers and you will either

1. receive a different answer to the
previous professor or
2. receive no answer.

■

Since theorems are implemented internally as a special case of [definition lists](#), it is possible to have multiple entries. This simply creates a new paragraph and has no special effect.

Remark.
: You can indeed define
multiple entries,
: but this has no effect,
and you are better off

simply using a blank
line to make paragraphs.

Remark. You can indeed define multiple
entries,

but this has no effect, and you are
better off

simply using a blank line to make
paragraphs.

6.2 Definition Lists

Although the `definition` theorem is available, you may prefer to use definition lists, the syntax for which is:

```
<term>
: <definition>

<term>
: <definition>

<term 1>
<term 2>
<term 3>
: <definition A>
: <definition B>
```

Critically,

1. a colon is required on a newline to mark each new definition,
2. each term-definition group must be separated by a blank line as above, and
3. definitions can contain block content, but each newline must be indented to align with the first.

Multiple terms can be assigned to the same definition, multiple definitions can be assigned to the same term, and multiple terms can be assigned to multiple definitions, although the use case for this is probably very limited.

| | |
|---|---|
| Term : Definition goes here, with > blockquotes. | Term Definition goes here, with blockquotes. |
| Multiple Terms Here : are assigned : to multiple : definitions here. | Multiple Terms Here are assigned to multiple definitions here. |

6.3 Tables

The area where YuMD deviates the most from other Markdown implementations is with regard to tables. The [most popular method](#), as implemented by GitHub Flavoured Markdown, uses ugly gutter characters to mark out tables, limiting yourself to plaintext in each cell. Instead, tables are defined in YuMD using two-dimensional lists. This gives the advantage of both

1. readability, and critically
2. the flexibility to place block elements (blockquotes, lists, theorems, etc.) within tables, rather than just plaintext.

The basic syntax for a table is by nesting unordered lists using `+` or `-`. For a valid table, you must use only `+` or only `-`.

Tables Defined Row-by-Row Tables using `+` are defined row-by-row.

| | | |
|------------------------|--------------------------------|--------------|
| + + Table | Table | Data |
| + Data | Mathematics: | Blockquotes: |
| + + Mathematics: | | |
| \[| $F(x) \equiv \int_0^x f(t) dt$ | Insert |
| F(x) \equiv | (6.2) | in- |
| \int_0^x f(t) dd{t} | | spir- |
| \] | | a- |
| | | tional |
| + Blockquotes: | | quo- |
| | | ta- |
| | | tion |
| > Insert inspirational | | here. |
| quotation here. | | |

Captions A final row with a single cell can be used to define a caption for a table. Critically, this cannot contain block content, only inline elements (math, emphasis, etc.).

| | | | |
|-------------------------|---|----------------|-----------------|
| + + Person \ (A\) | Person A | Econ | <i>Pembroke</i> |
| + **Econ** | Person B | Eng | <i>Trinity</i> |
| + *Pembroke* | Person C | Compsci | <i>Girton</i> |
| + + Person \ (B\) | Table 6.1: Subject choices for person $i \in \{A, B, C\}$. | | |
| + **Eng** | | | |
| + *Trinity* | | | |
| + + Person \ (C\) | | | |
| + **Compsci** | | | |
| + *Girton* | | | |
| + + Subject choices for | | | |
| person \ (i \in \{A, | | | |
| B, C\} \). | | | |

Note 6.2. The blank line between each table entry is entirely optional. The following examples omit these blank lines.

Headings If all cells of the first row contain only emphasis items (italic or strong), then the first row is interpreted as a heading row and formatted centred and bold.

| | | |
|--------------|-------------|------------|
| + + **Name** | Name | Age |
| + **Age** | Benson | 19 |
| + + Benson | Maxwell | 20 |
| + 19 | | |
| + + Maxwell | | |
| + 20 | | |

Alignment Row Cell alignments can be defined if the very first row contains cells with text `l` or `c` or `r` or `j` depending on whether you want left-aligned, right-aligned, centred or justified columns respectively. This will align the rest of each column accordingly. By default, left alignment is used if no alignment row is given.

| | | |
|---------------------|----------|----------------------|
| + + l | Benson | 19 |
| + + r | Maxwell | 20 |
| + + Benson | Universe | <i>13.7 trillion</i> |
| + + 19 | | |
| + + Maxwell | | |
| + + 20 | | |
| + + Universe | | |
| + + *13.7 trillion* | | |

If a heading row is desired, it must immediately *follow* the alignment row. Heading cells are centred regardless of the specified column alignment.

| | | |
|--------------------|-------------|---------------------|
| + + l | Name | Age |
| + + r | Benson | 19 |
| + + **Name** | Maxwell | 20 |
| + + **Age** | Universe | <i>13.8 billion</i> |
| + + Benson | | |
| + + 19 | | |
| + + Maxwell | | |
| + + 20 | | |
| + + Universe | | |
| + + *13.8 billion* | | |

Transposed Tables In many a case, it is more convenient to define tables column-by-column rather than row-by-row. For this, `-` is used instead of `+`.

| | | |
|------------------|--------------|-----|
| - - Oil | Oil | £1 |
| - - Soy sauce | Soy sauce | £3 |
| - - Hoisin sauce | Hoisin sauce | \$2 |
| - - Beer | Beer | \$2 |
| - - £1 | | |
| - - £3 | | |
| - - \$2 | | |
| - - \$2 | | |

Although the table body is transposed, **heading and alignment rows remain interpreted by row**.

```

- - l
- - c
- - r
- - **Name**
- - **Subject**
- - **College**

- - Lorem
- - Ipsum
- - Dolor
- - Sit
- - Engineering
- - Computer Science
- - Economics
- - MML
- - Pembroke
- - Kings
- - Trinity
- - Girton

- - Subjects and colleges.

```

| Name | Subject | College |
|-------|------------------|----------|
| Lorem | Engineering | Pembroke |
| Ipsum | Computer Science | Kings |
| Dolor | Economics | Trinity |
| Sit | MML | Girton |

Table 6.2: Subjects and colleges.

6.4 Footnotes

Inline Footnotes Footnotes can be defined inline by using the syntax `[^<footnote text>]`. This is possibly the most concise and useful way of defining footnotes, but has the limitation that only inline formatting (emphasis, links, etc.) can be included within the footnote. No block elements can be included if defined this way.

Footnote References Rather than writing out the footnote within the text, you can define the footnote elsewhere and refer its label within the text. Footnotes are defined using the following syntax:

```
[^<footnote-label>]:
  <block content>
```

Critically,

1. `<footnote-label>` must not contain spaces, and
2. the content of the footnote must be indented by at least 3 spaces.

Footnotes are then referred to using the syntax `[^<footnote-label>]`.

```

This is my footnote.[^a footnote
goes *here*.]

```

```

This is my footnote.a
aa footnote goes here.

```

```

See footnote.[^my-fntnote]

```

```

[^my-fntnote]:
  A big footnote with

  1. > blockquotes
  1. and lists

```

```

See footnote.a
aA big footnote with
1. blockquotes
2. and lists

```

Note 6.3. The syntax for an inline footnote is identical to the syntax for a footnote reference. If `<footnote-label>` is not defined or has spaces, then it will be interpreted as an inline footnote.

Note 6.4. The ordering of footnote definition and reference does not matter. Footnotes can be referred to before they are defined as in the example above.

6.5 References

YuMD allows you to refer to parts of the document in a similar manner to LaTeX. All blocks can be referenced and linked to in HTML, but only section headings, figures, tables and specific theorems in LaTeX.

To label an element, use the syntax `#<label-name>` at the appropriate position as shown in Table 6.5. See Section 6.12 for how to label equations.

Critically, `<label-name>` cannot contain spaces.

To refer back to the element, you can either use syntax

1. `[<link text>](#<label-name>)` similarly to how you would use a link, or simply
2. `[#<label-name>]` to allow LaTeX to automatically create link text by cleverly inferring the type of element. Note that the HTML output will simply be a link titled `#<label name>`, without clever inference.

References can appear before the label. References to undefined elements will issue a warning.

Table 6.5 shows how to refer to labelled elements, and where to place labels for sections, theorems, tables and figures.

Note 6.5. Only specific theorems can be linked to in LaTeX. Specifically, unnumbered theorems cannot be linked to. These are:

1. `assumption`,
2. `proof`,
3. `remark`,
4. `solution`, and
5. `tip`.

Note 6.6. In order to label a figure, you **must have a blank line before the figure** (demonstrated in Table 6.5). Otherwise, there is ambiguity with regard to labelling the figure or labelling the paragraph before it.

Note 6.7. If you would like to typeset a paragraph with the text `#<hashtag>`, you must escape the `#`. **Not escaping will produce an empty paragraph**, since the hashtag is interpreted as a label.

`#borisout!!!`

`\#borisout!!!`

`#borisout!!!`

See [#my-section] for art.

A Section #my-section

Theorem. #big-thm
: An important theorem.

- - Micro
- - Macro
- - Block
- - Tambakis
- - Hi. #big-tbl

![Art.](./pem.png) #art

[The theorem] (#big-thm)
is important. Please also
see [#big-tbl] and
[this art] (#art).

To label a figure without
a caption, use a space as
a "caption".

! [] (pem.png) #true-art

See [#true-art] for wow.

To label a table without a
caption, place it where
the caption would go.

[#ptr] shows possible
ways of declaring a
pointer.

- - `int *a; int* b;`
- - `int * c;`
- - Understandable
- - Whyyyyyyyyyyyyyyyyyyyyyyy
- - #ptr

See Section 6.6 for art.

6.6 A Section

Theorem 6.2. *An important theorem.*

| | |
|-------|----------|
| Micro | Block |
| Macro | Tambakis |

Table 6.3: Hi.



Figure 6.1: Art.

The theorem is important. Please also
see Table 6.3 and this art.

To label a figure without a caption, use
a space as a 'caption'.



Figure 6.2

See Fig. 6.2 for wow.

To label a table without a caption, place
it where the caption would go.

Table 6.4 shows possible ways of
declaring a pointer.

| | |
|----------|---------------------------|
| int *a; | Understandable |
| int* b; | |
| int * c; | Whyyyyyyyyyyyyyyyyyyyyyyy |

Table 6.4

Table 6.5: Referencing across the document.

6.7 Citations

YuMD allows you to include BibTeX `.bib` bibliography files and use citations in-text or as footnotes. Bibliographies are included by specifying a path to the bibliography in the file configuration, such as using a YAML header at the top of the file (see Section 8), and optionally specifying the `citation-style` as `in-text` or `footnote`:

```
---
bibliography: "./path/to/bibliography.bib"
cite-style: in-text
...
```

Multiple bibliographies can also be included:

```
---
bibliography: ["./path/to/bibliography-1.bib", "./path/to/bibliography-2.bib"]
cite-style: footnote
...
```

To cite an entry in an included BibTeX file, use the syntax `[@<entry-name>]`. Optionally, you can use the syntax `[@<entry-name>:<text before>]` and `[@<entry-name>:<text before>:<text after>]` to include text before and/or after the citation, such as page numbers.

If no `<text before>` is desired but `<text after>` is, place a **space between the two colons** to avoid YuMD mistaking the line as the beginning of a flashcard.

Critically, `<entry-name>` cannot contain a space (although `<text before>` and `<text after>` can).

| | |
|--|--|
| <code>[@sargent2009ends]</code> posits that persistent fiscal deficits are the driver behind persistently high levels of inflation. | Sargent (2009) posits that persistent fiscal deficits are the driver behind persistently high levels of inflation. |
| <code>[@de2010girl: :p. 18]</code> argue this could have possibly led to higher rates of fertility during marriage, through a greater propensity to procreate. | De Moor and Van Zanden (2010, p. 18) argue this could have possibly led to higher rates of fertility during marriage, through a greater propensity to procreate. |

Note 6.8. Paths can be specified relative to the `.yumd` file from which the file is included, or absolutely. Note that environment variables and placeholders like `~` are not respected by YuMD.

Note 6.9. YuMD does not allow you to mix citation styles within the same document. You must either use `in-text` or `footnote` for all citations.

6.8 Smarty pants

By default, YuMD implements a plethora of ‘typography’ substitutions such as arrows and dashes, which are detailed in Table 9.1. This can be turned off by specifying `typography: false` in the file configuration.

6.9 Including other `.yumd` files

A powerful feature of YuMD is the ability to import other `.yumd` files within a `.yumd` file. This is done by using the syntax `[[<./path/to/file>]]`, where the path is given relative to the directory containing the `.yumd` file from which the file is included, or absolutely. The leading `‘./’` is unnecessary if the included file exists in the same folder or a subdirectory.

Files can be included anywhere within a document, including within other blocks.

Assuming the following file structure:

```
report.yumd
sections/
  intro.yumd
  cclsns.yumd
qts/
  friedman.yumd
```

In `report.yumd` (the root file):

```
## Introduction
[[sections/intro.yumd]]
## Conclusion
[[sections/cclsns.yumd]]
```

In `sections/intro.yumd`:

```
Introduction content goes
here, with quotation from
Friedman:
> [[../qts/friedman.yumd]]
```

In `sections/cclsns.yumd`:

Conclusion goes here.

In `qts/friedman.yumd`:

```
A society that puts
equality before freedom
will get neither.
```

6.10 Introduction

Introduction content goes here, with quotation from Friedman:

A society that puts equality
before freedom will get
neither.

6.11 Conclusion

Conclusion goes here.

Note 6.10. The YuMD executable must be fed the root `.yumd` file in order to include all files correctly.

Note 6.11. LaTeX and HTML outputs treat included files differently. When outputting LaTeX, YuMD will create a `.tex` file for all `.yumd` documents, and preserve the given directory structure. When outputting HTML, YuMD generates a single large `.html` file.

Note 6.12. Included `.yumd` files inherit the configuration of parent `.yumd` files. Configuration is merged with (in the case of non-scalars, such as bibliography lists) or overwrites (in the case of scalars) the configuration of parent files. For example, this means that if `title-type: document` is defined in a parent `.yumd` file, you must manually specify `title-type: none` or `title-type: chapter` in all child `.yumd` files in order to avoid creating multiple undesired title pages throughout your document, since each child file inherits the `title-type` of `document` by default.

Note 6.13. Including files which do not exist will issue a warning and have no effect on the final LaTeX or HTML output.

6.12 LaTeX Math

Inline math can be typeset by surrounding it either with escaped brackets `\(` and `\)` or with dollar signs `$`. Display math can be typeset with either escaped square brackets `\[` and `\]` or double dollar signs `$$`. By convention, brackets are the preferred notation.

Display math is automatically typeset as an `align` environment, so multiple lines of equations can be created with no hassle using standard `\\` notation, aligned as usual with `&`.

Example 6.1. The following code:

Definition (variance-covariance matrix).

: A variance-covariance matrix $\mathrm{Var}(\hat{\beta})$ is defined

by:

```
\[
  \mathrm{Var}(\hat{\beta})_{ij} \equiv
    \mathrm{Cov}(\hat{\beta}_i, \hat{\beta}_j) \equiv
    \mathrm{E}[(\hat{\beta}_i - \mathrm{E}(\hat{\beta}_i))
      (\hat{\beta}_j - \mathrm{E}(\hat{\beta}_j))].
\]
```

All variance-covariance matrices are positive semi-definite.

Results in:

Definition 6.2 (variance-covariance matrix). A variance-covariance matrix $\mathrm{Var}(\hat{\beta})$ is defined by:

$$\mathrm{Var}(\hat{\beta})_{ij} = \mathrm{Cov}(\hat{\beta}_i, \hat{\beta}_j) \quad (6.3)$$

$$\equiv \mathrm{E}[(\hat{\beta}_i - \mathrm{E}(\hat{\beta}_i))(\hat{\beta}_j - \mathrm{E}(\hat{\beta}_j))]. \quad (6.4)$$

All variance-covariance matrices are positive semi-definite.

Note 6.14. YuMD disallows blank lines in math elements. This is because blank lines result in ambiguity with regard to whether the Markdown is contiguous LaTeX math code or a new paragraph.

Note 6.15. LaTeX disallows using the & alignment character in alignment environments within a table. There is a workaround for this, but this is not implemented by YuMD. If necessary, this can be considered in a future update.

Note 6.16. YuMD automatically loads in the [physics package](#), an extremely useful package that allows you to quickly typeset derivatives, vectors, auto-sizing brackets, and common matrix types. Although this works automatically in LaTeX output and for the HTML preview output in VSCode, Anki (which uses MathJax to render LaTeX math) does not load the `physics` package by default. Section 9.1 details how to enable the `physics` package in Anki on desktop. The `physics` package is by no means necessary and many people choose not to use it (either [by ignorance](#) or [purposeful avoidance](#)), so this is completely optional.

Referencing Equations YuMD supports the standard LaTeX `\label{<equation-label>}` syntax, allowing you to refer to equations across the document.

| Utility is given by | Utility is given by |
|--|---|
| <pre>\[U(x, y) = \min\{ax, by\} \label{pc-util} \]</pre> | $U(x, y) = \min\{ax, by\} \quad (6.5)$ |
| <pre>Expenditure is minimised to achieve a given utility if \{ax\} and \{by\} in [#pc-util] are equal.</pre> | <p>Expenditure is minimised to achieve a given utility if ax and by in Eq. (6.5) are equal.</p> |

6.13 Figures

Images YuMD supports including images using the syntax

`! [<figure caption>](./path/to/image.png)`

Table 6.5 shows this in action. If you would like no figure caption but still want figure numbering within LaTeX or need to reference the figure, leave a space between the square brackets.

Advanced: SVGs SVGs allow you to create figures with LaTeX-rendered text and math.

If you have Inkscape and have exported an SVG image using the [PDF+LaTeX export format](#), the SVG can be included using the syntax

`+ [<figure caption>](./path/to/image.svg)`

YuMD searches for the corresponding `.pdf_tex` and `.pdf` files in the same directory as the `.svg` file.

If the Inkscape binary is in the `PATH` environment variable, or alternatively if a path to an Inkscape binary is explicitly set within the VSCode extension settings, a file watcher detects and automatically generates and updates corresponding `.pdf_tex` and `.pdf` files whenever `.svg` files are created or updated, saving the hassle of going to the command line every time a change is made within an SVG. The VSCode extension also makes it easy to create and edit SVGs linked to a `.yumd` document: `Ctrl`- or `Command`-clicking the path opens the SVG in Inkscape or creates a blank SVG if one does not exist.

6.14 Unsupported: Native Code

HTML Code Unlike standard Markdown, where all HTML is treated as valid Markdown, YuMD only supports partial treatment of HTML. Although printed verbatim when outputting HTML, all HTML code is typeset as a code block when outputting LaTeX. We **strongly recommend avoiding the use of raw HTML** in your YuMD files.

LaTeX Code Other than in [math mode](#) and in the [custom LaTeX appended to the preamble](#), YuMD does not support direct LaTeX code. Backslashes are escaped when outputting LaTeX.

7 Defining Anki Flashcards

A core function of YuMD is to allow seamless creation of flashcards from within your notes, syncing them to Anki. YuMD offers a plethora of ways to set out flashcards, so it is well worth reading this entire section. Once you have written your flashcards, Section 3 details how to sync them with Anki.

The easiest way to introduce flashcard syntax is to begin with the most verbose, before covering the shorter and more succinct syntax which allow for much more rapid and intuitive flashcard creation.

7.1 Explicit Flashcard Generation

Flashcards can be generated explicitly by using the syntax

```
<deck name>::<note type>
  --- <field name> ---
  <block content>

  --- <field name> ---
  <block content>

  --- <field name> ---
  <block content>
```

This creates a flashcard of `<note type>` with the specified content, and sends it to deck `<deck name>`.

The content of the flashcard must be indented by at least 2 spaces.

Crucially, `<deck name>` and `<note type>` must exist. By design, YuMD does not create decks/note types if they do not exist, as this could lead to accidental creation in the case of typos. Create your decks and note types before sending your flashcards to Anki (or stick with the defaults — read below).

The field separators `---` can be as long as you want, and can contain spaces, but crucially must have at least 1 hyphen on both sides. The following code is therefore valid:

```
Deck::Note Type
-----Field 1-----
Field 1 content
-- - --   Field 2   -- --- -
Field 2 content
- Field 3-
Field 3 content,
```

although whether anybody would ever want to write this in practice is the question.

Note 7.1. Flashcards can be sent to subdecks as you would expect. For example,

```
Deck::Subdeck::Subsubdeck::Note Type
  <content>
```

is completely valid. Ensure that the subdeck exists before sending your flashcards.

We now detail the many levels of omission and inference offered by YuMD which allow for much more rapid flashcard creation, since, admittedly, the full syntax for a flashcard is quite verbose.

7.2 Default Deck

Rather than explicitly specifying the deck name for every flashcard, you can leave it blank, and YuMD will send the flashcard to the file-specific default deck. This default deck can be specified in the file configuration, such as in a [YAML header](#) at the top of the file:

```
---
anki:
  deck: <name of default deck>
...
```

By default, `deck` is set to `Default`, which should exist in every Anki installation unless you have renamed it. It may be of use to set `default-deck` to something more helpful, such as `Macroeconomics`.

Example 7.1. The following code:

```
---
anki:
  deck: Macroeconomics
...

::Custom Note
  Flashcard content...
```

will send a flashcard of type `Custom Note` to deck `Macroeconomics`.

7.3 Inferred Fields

Rather than explicitly specifying the name of your fields, YuMD can infer them by matching them up with the order of the fields as defined in Anki. Thus, given a note type `Custom Note` with fields `A`, `B` and `C` in that order, the code

```
::Custom Note
---
  <content within A>
```

```

---
<content within B>
---
<content within C>

```

creates an appropriately-filled flashcard.

If the first field is inferred, you can omit the first ---:

```

::Custom Note
  <content within A>
  ---
  <content within B>
  ---
  <content within C>

```

Note 7.2. YuMD supports mixing explicitly-named fields and inferred fields. Named fields are populated first, and unnamed fields will fill the remaining fields in the flashcard in order.

Note 7.3. The first field must have content, but content in remaining fields can be left blank.

Note 7.4. Field separators must have two or more hyphens.

7.4 Inferred Note Type

<note type> can be left blank and YuMD will scan the content of the flashcard and automatically determine the desired type of flashcard.

Cloze If <note type> is blank and YuMD detects any [cloze element](#) within the flashcard, either inline or block level, YuMD will infer note type **Cloze** by default, which should exist in all Anki installations unless you have renamed it.

Example 7.2. The code

```

::
  To achieve the golden rule steady state, policymakers must {adjust the savings rate  $\leftarrow$ 
    \((s\)}\}.

```

generates a flashcard of type **Cloze** in the default deck, occluding the text ‘adjust the savings rate s ’.

You can specify a note type other than **Cloze** in the configuration using

```

---
anki:
  cloze-note-type: <custom cloze note type name>
...

```

Theorems If the flashcard contains a single unnamed field and the only item within that field is a *named* theorem (definition, proof, corollary, etc.), YuMD will create a flashcard with two fields, separating the theorem name, which functions as a prompt, and theorem body.

```

::
Theorem (Fermat's little).
: If  $(p)$  is a prime number,
  then for every integer
   $(a)$ ,

  
$$a^p \equiv a \pmod{p}.$$


```

| |
|---|
| Front |
| Theorem (<i>Fermat's little</i>). |
| Back |
| If (p) is a prime number, then for every integer (a) , $a^p \equiv a \pmod{p}.$ |

By default, YuMD sends a flashcard of type **Basic** to Anki, which should exist in all Anki installations unless you have renamed it, but this can be changed using configuration

```

---
anki:
  theorem-note-type: <custom theorem note type>
...

```

Definition Lists If the flashcard contains a single unnamed field and the only element within that field is a definition list, YuMD will separate the terms and definitions and create a two-field flashcard.

```

::
M1
: M0 + demand deposits,
  travelers' checkques and
  other checkable deposits

M2
: M1 + small time deposits,
  small savings deposits

M3
: M2 + large time deposits

M4
: M3 + least liquid assets,
  such as long-term bonds

```

| |
|---|
| Front |
| M1 |
| M2 |
| M3 |
| M4 |
| Back |
| M0 + demand deposits, travelers' checkques and other checkable deposits |
| M1 + small time deposits, small savings deposits |
| M2 + large time deposits |
| M3 + least liquid assets, such as long-term bonds |

Figure 7.1

By default, this flashcard is of type **Basic**, but this can be changed using configuration

```

---
anki:
  definition-note-type: <custom definition note type>
...

```

Note 7.5. Cloze note type takes precedence of theorems and definition lists: if you have clozes within theorems or definition lists within the flashcard, YuMD will infer type cloze rather than theorem, which is likely the most desired behaviour.

```

::
Theorem (Fermat's little).
: If  $(p)$  is a {prime
  number}, then for every
  integer  $(a)$ ,

  {
  \[
    a^p \equiv a \pmod{p}.
  \]
  }

```

Fallback If `<note type>` is left blank and YuMD does not recognize a cloze, theorem, or definition list, it will default to note type `Basic`. This can be changed by using configuration

```

----
anki:
  plain-note-type: <custom fallback note type>
...

```

This fallback note type can have as many fields as you like.

Tip. Specifying a `deck` and `[cloze|theorem|definition|plain]-note-type` in the file configuration and relying on inference can hugely speed up the process of creating complex cards within your YuMD notes.

7.5 Occlusion (Cloze Flashcards)

Anki users will be familiar with text occlusion, where parts of a sentence or paragraph are hidden and must be recalled.

Inline Occlusion To specify inline text to be occluded, surround the desired text with curly brackets `{}`. Paragraphs can have multiple occlusions. The following example generates a single flashcard where the text ‘ineffective’ and ‘indifference between bonds and cash’ is to be recalled.

```

::
When short term rates are
near zero, open market
operations are near
{ineffective} as there is an
{indifference between bonds
and cash}.

```

Figure 7.2: Basic inline occlusion.

By default, YuMD sets the index of clozes to 1. To change the cloze index, simply write `<n>:` immediately after the opening `{`, where `<n>` is the desired index. The following example sends a note containing two cards to Anki, the first requiring you to recall ‘ineffective’ and ‘indifference between bonds and cash’, and the second requiring you to recall ‘near zero’.

```
::
When short term rates are
{2:near zero}, open market
operations are near
{1:ineffective} as there is
an {indifference between
bonds and cash}.
```

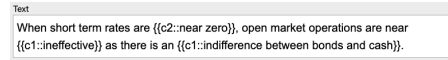


Figure 7.3: Cloze indices.

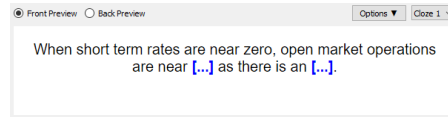


Figure 7.4: Front side.

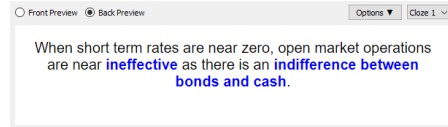


Figure 7.5: Back side.

Placeholders are also supported, which function as a prompt to help avoid ambiguity when there could be multiple things that fill a gap. This is done by appending `<placeholder text>` just before the closing `}`, where crucially, `<placeholder text>` is in plaintext. The following example results in ‘types of money’ being printed on the front side of card 1 rather than ‘[...]’:

```
When short term rates are
{2:near zero}, open market
operations are near
{1:ineffective} as there is an
{indifference between bonds
and cash:types of money}.
```

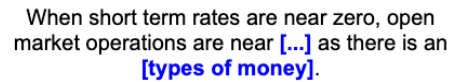


Figure 7.6: Placeholders.

Block-Level Cloze The notation so far only occludes inline text within paragraphs. If you would like to hide and recall entire blocks of content, wrap the content within curly braces like so:

```
::
<block content functioning as a prompt>
{
  <block content to be occluded>
}
```

Critically, **block clozes are only recognized within a flashcard block notated using `<optional deck name>::<optional note type>`**, unlike inline clozes which [allow the paragraph to implicitly become a flashcard](#). This is because it makes no sense to have a solitary cloze block in a flashcard: other blocks must be grouped with the cloze block in order to function as a prompt for memory recall.

Blasphemous as it is, there is no indentation for the body of the block cloze.

```

::
Independent central banks
are mandated with some, but
not necessarily all of:
{
1. price stability
1. output stability
1. low unemployment
1. liquidity and smooth
market functioning
}

```

Text

Independent central banks are mandated with some, but not necessarily all of:

{{c1::

1. price stability
2. output stability
3. low unemployment
4. liquidity and smooth market functioning

}}

Figure 7.7: Flashcard created using block cloze.

Like inline clozes, indices and placeholder text can be specified using `<d>:` and `:<placeholder text>` after and before the opening the clozing braces respectively.

```

::
A *non-independent* central
bank may be used to {serve
the interests of the current
government}, such as through
seigniorage: if a government
cannot either

{2:
1. raise taxes, or
1. issue bonds,
:ways of raising revenue}

then it may resort to
seigniorage.

```

A non-independent central bank may be used to serve the interests of the current government, such as through seigniorage: if a government cannot either **[ways of raising revenue]** then it may resort to seigniorage.

Figure 7.8

This example also demonstrates that inline and block clozes can be used in tandem.

7.6 Implicit Flashcards

Implicit flashcards further simplify flashcard notation and streamline flashcard creation by **allowing you to omit even the `::` notation**.

Inline Cloze Paragraphs with cloze text are automatically turned into flashcards. This means that throwing a few `{s` and `js` into your notes rapidly creates fill-in-the-blank style flashcards.

The following code generates two notes (three cards) within Anki, without any use of `::` notation:

The rationale behind efficiency wages is that {increased productivity per worker justifies the cost of higher wages}, but this results in {greater structural unemployment}.

Firms may offer wages above the market equilibrium to {1:attract higher-quality applicants}, and increase {2:worker effort} and reduce {2:shirking}.

| Sort Field | Card |
|---|---------|
| Firms may offer wages above the market equilibrium to (c1:attract higher-quality applicants), and increase (c2:... | Cloze 1 |
| Firms may offer wages above the market equilibrium to (c1:attract higher-quality applicants), and increase (c2:... | Cloze 2 |
| The rationale behind efficiency wages is that (c1:increased productivity per worker justifies the cost of higher wag... | Cloze 1 |

Figure 7.9: 2 notes and 3 cards generated.

| | | | |
|---|--------------|---------|---------|
| Front Preview | Back Preview | Options | Cloze 1 |
| The rationale behind efficiency wages is that [...], but this results in [...]. | | | |

Figure 7.10: Note 1.

| | | | |
|--|--------------|---------|---------|
| Front Preview | Back Preview | Options | Cloze 1 |
| Firms may offer wages above the market equilibrium to [...], and increase worker effort and reduce shirking. | | | |

Figure 7.11: Note 2, cloze 1.

| | | | |
|---|--------------|---------|---------|
| Front Preview | Back Preview | Options | Cloze 2 |
| Firms may offer wages above the market equilibrium to attract higher-quality applicants, and increase [...] and reduce [...]. | | | |

Figure 7.12: Note 2, cloze 2.

Theorems and Definition Lists Named theorems and definition lists are turned into flashcards automatically, in the same manner as described in Section 7.4, without needing to explicitly reside within a flashcard.

Thm (utility representation).
: If a preference relation \succsim over a finite set of alternatives \mathcal{A} is complete and transitive, then it has a utility representation.

| |
|---|
| Front |
| Theorem (utility representation). |
| Back |
| If a preference relation \succsim over a finite set of alternatives \mathcal{A} is complete and transitive, then it has a utility representation. |

Figure 7.13

Note 7.6. Only named theorems can be sent this way to Anki — unnamed theorems provide no prompt for recall and so are meaningless as a flashcard. If you would like to send an unnamed theorem to Anki, put clozes inside it or wrap it explicitly in a flashcard with some other elements.

Preventing implicit flashcard creation To prevent theorems and definition lists from being turned into flashcards, append `##ignore` after the period in the theorem, or after the first term in the definition list. For example, the following code results in no flashcard creation:

Theorem (first order condition of local extrema). ##ignore
: Let d be an interior point in D such that d is a local extremum of a continuous function f with domain D . If f is differentiable at d , then $f'(d) = 0$.

Local maximum ##ignore
: A point $d \in D$ is a local maximum of a function f with domain D if there exists some $\delta > 0$ such that $f(x) \leq f(d)$ for all $x \in D$ such that $|\abs{x - d}| \leq \delta$.

Local minimum

: A point $d \in D$ is a local minimum of a function f with domain D if there exists some $\delta > 0$ such that $f(x) \geq f(d)$ for all $x \in D$ such that $|\text{abs}\{x - d\}| \leq \delta$.

Grouping Clozes Implicit cloze flashcards are rendered on a paragraph-by-paragraph basis. This means for example that the following code produces three flashcards, rather than the intended single flashcard:

In the balance sheet:

1. **assets** denote {anything valuable owned by the institution} (+),
1. **liabilities** denote {anything valuable that the institution owes to others} (-), and
1. **net worth, equity or capital** is {assets minus liabilities}.

In order to group these together, you **must place them explicitly within a flashcard**:

::

In the balance sheet:

1. **assets** denote {anything valuable owned by the institution} (+),
1. **liabilities** denote {anything valuable that the institution owes to others} (-), and
1. **net worth, equity or capital** is {assets minus liabilities}.

The VSCode extension provides a handy way of doing this: select your elements to be grouped, type **Ctrl+Space** (in both Windows and Mac), and select the **wrap-in-card** snippet.

Grouping Theorems/Corollaries/Proofs Often, it is desired to group a theorem with a proof, or a theorem with a corollary (or anything else desired). YuMD's flexibility will allow you to produce many creative ways of achieving this, but the following two methods are suggested:

In the first method, you can explicitly place multiple theorems within a single flashcard and use inline or block cloze to hide their content:

::

Theorem (intermediate value).

: {Let f be continuous on $D = [a, b]$ and let $A = \min_{x \in D} f(x)$ and $B = \max_{x \in D} f(x)$. Then f takes on all the values between A and B .}

Corollary (fixed point).

: {Given a continuous function f defined on the interval $D = [a, b]$, with range $R = [A, B]$, if it is true that $B - b \geq 0$ and $A - a \leq 0$, then there exists a point $c \in D$ such that $f(c) = c$.}

Alternatively, you can nest the proofs/corollaries within the corresponding theorem. Take care of indentation levels when doing this: the corollary must be indented within the theorem.

Theorem (intermediate value).

: Let f be continuous on $D = [a, b]$ and let $A = \min_{x \in D} f(x)$ and $B = \max_{x \in D} f(x)$. Then f takes on all the values between A and B .

Corollary (fixed point).

: Given a continuous function f defined on the interval $D = [a, b]$, with range $R = [A, B]$, if it is true that $B \geq b$ and $A \leq a$, then there exists a point $c \in D$ such that $f(c) = c$.

Theorem *(intermediate value).*
[...]
Corollary *(fixed point).*
[...]

Figure 7.14: Method 1 (clozes).

Theorem *(intermediate value).*
 Let f be continuous on $D = [a, b]$ and let $A = \min_{x \in D} f(x)$ and $B = \max_{x \in D} f(x)$. Then f takes on all the values between A and B .
Corollary *(fixed point).*
 Given a continuous function f defined on the interval $D = [a, b]$, with range $R = [A, B]$, if it is true that $B \geq b$ and $A \leq a$, then there exists a point $c \in D$ such that $f(c) = c$.

Figure 7.15: Method 2 (nested corollary).

Note 7.7. As is clear, Anki’s default styling is acceptable but ugly. Feel free to define your own stylesheets within Anki — this is where the `[cloze|theorem|definition|plain]-note-type` options become very handy.

8 YuMD File Configuration

YuMD supports file configuration using [YAML](#) in two ways: first, through a YAML header at the **beginning** of the file, and second, through a separate `<filename>.yucfg` file beside corresponding `<filename>.yumd` file. When both are used, the configurations are merged, with the YAML header taking precedence over the `.yucfg` file.

YAML Header A YAML header is the most convenient way of setting out configuration. A YAML header is placed at the **very beginning of the file** and starts with `---` and ends with `...`. Critically, the `---` **must be the first line** of the `.yumd` file.

```
---
# YAML configuration code goes here
...
```

.yucfg Configuration File Alternatively, a configuration file with the same filename as the `.yumd` file can be created. *If you install the [YAML Language Support](#) extension by Red Hat in VSCode, this has the advantage of providing autocomplete, saving the need to refer to the documentation all the time.*

Note 8.1. The `.yucfg` file does not need to begin with `---` and end with `...` like the YAML header, although it can.

8.1 A Complete Configuration

Note 8.2. Like YuMD, YAML treats indentation very seriously: **all subkeys should be indented by two spaces**.

Below is an example of a complete configuration. All items of configuration are optional.

```
title-type: document

title: Macroeconomics
subtitle: Principles I
ccode: C0001
subject: Economics
lecturer: Milton Friedman
author: James Yu
season: Michaelmas
part: I
date: 2021-01-01
date-end: 2021-12-14
place: Cambridge
```

```

section-offset: section

bibliography: "bibliography.bib"

flashcards-as-table: auto
typography: true

cite-style: in-text
language: british

html:
  smart-quotations: true

anki:
  deck: Default
  cloze-note-type: Cloze
  theorem-note-type: Basic
  definition-note-type: Basic
  plain-note-type: Basic

latex:
  toc: false
  break-after-toc: true
  tocdepth: 3
  secnumdepth: 3
  post-preamble: |
    \numberwithin{equation}{subsection}

```

8.2 General Options

| | |
|--|--|
| title-type: <code>[document chapter none]</code> (default: none) | <ol style="list-style-type: none"> 1. If set to <code>document</code>, YuMD will print all of <code>title</code>, <code>subtitle</code>, <code>ccode</code>, <code>subject</code>, <code>lecturer</code>, <code>author</code>, <code>season</code>, <code>part</code>, <code>date</code>, <code>date-end</code>, and <code>place</code> in the form of a large title page 2. If set to <code>chapter</code>, YuMD will print <code>title</code>, <code>subtitle</code>, <code>lecturer</code>, <code>date</code> and <code>date-end</code> in the form of a smaller title section. 3. If set to <code>none</code>, YuMD will create no title. |
| title (default: YuMD Notes) | The title of the document. |
| subtitle , ccode , subject , lecturer , author , season , part , date , date-end , and place | Self-explanatory. Blasphemous as it is, season is intended to refer to the term, e.g. ‘Michaelmas 2021’. |
| date and date-end | Must be formatted in big-endian form <code>YYYY-MM-DD</code> to allow it to be printed nicely in LaTeX and HTML. |
| section-offset: <code>[part chapter section subsection paragraph subparagraph 0 1 2 3 4 5]</code> (default: section) | Specify the level of heading corresponding to a single-hash heading, e.g. <code># Heading</code> . |
| bibliography | Specify the path to included bibliographies. Can be a string or an array of strings for multiple bibliographies. |
| flashcards-as-table: <code>[auto always never]</code> (default: auto) | <ol style="list-style-type: none"> 1. If set to <code>always</code>, YuMD will format flashcards as tables with label in the first column and content in the second column. 2. If set to <code>auto</code>, YuMD will format flashcards as a table if there are two or more fields and at least one of them is named. 3. If set to <code>never</code> YuMD will output flashcards as normal document body content. |
| typography: <code>[true false]</code> (default: true) | Specify whether to use smartypants replacements . |

8.3 Document-Level Options

| | |
|---|---|
| <code>cite-style: [in-text footnote]</code> (default: <code>in-text</code>) | Specify the citation style. YuMD only considers <code>cite-style</code> specified in the root document and applies it to all included <code>.yumd</code> files. |
| <code>language</code> (default: <code>british</code>) | This is currently only used in LaTeX output to specify the language for <code>babel</code> , but may play a larger role in future. |

8.4 HTML Output-Specific Options

| | |
|--|--|
| <code>html:</code> <code>smart-quotations: [true false ↔]</code> (default: <code>true</code>) | Use typographers' quotations (fancy curly " and ') in HTML output. |
|--|--|

8.5 LaTeX Output-Specific Options

| | |
|--|--|
| <code>latex:</code> <code>toc: [true false]</code> (default: <code>false</code>) | Specify whether LaTeX should print a table of contents. <i>Note 8.3.</i> HTML output does not support table of contents output by design. |
| <code>latex:</code> <code>break-after-toc: [true false]</code> (default: <code>true</code>) | Specify whether to insert a page break after the table of contents. |
| <code>latex:</code> <code>post-preamble: <...></code> | Specify custom LaTeX preamble code to be placed after the hard-coded preamble . See this website for how to define multiline literals () in YAML. |

8.6 Anki-Specific Options

Tip. The system of configuration inheritance means that you can redefine these Anki options (or other options in general) in each included file and flashcard generation in that file will follow the desired file-specific configuration.

| | |
|---|---|
| <code>anki:</code> <code> deck: <...></code> | Specify the deck to which YuMD should send flashcards by default. |
| (default: Default) | |
| <code>anki:</code> <code> cloze-note-type: <...></code> <code> theorem-note-type: <...></code> <code> definition-note-type: <...></code> <code> plain-note-type: <...></code> | Specify the note type when YuMD infers that we have a cloze, theorem or definition list flashcard, or cannot infer. |
| (defaults: Cloze, Basic, Basic, Basic) | |

9 Appendix

For advanced users, directly using the command line may be more useful. Use option `--help` to print the available routines.

Options for Yu's Markdown:

| | |
|--|--|
| <code>-h [--help]</code> | print this message |
| <code>-w [--html] arg</code> | write HTML file. leave argument blank to write to stdout |
| <code>-m [--source-map]</code> | include source mapping into HTML output. useful for extensions |
| <code>--absolute-paths</code> | write absolute file paths for media, such as images, in HTML output; only works if the media exists on disk |
| <code>-a [--anki]</code> | send notes to anki |
| <code>-u [--anki-connect-url] arg</code> | (=127.0.0.1:8765) url used to talk to AnkiConnect |
| <code>-p [--wsl-prefix] arg</code> | specify the UNC path corresponding to the root directory of the WSL instance, eg " <code>\\wsl\$\\Ubuntu</code> ": necessary for sending media from WSL. |
| <code>-l [--latex] arg</code> | write LaTeX file(s) |
| <code>-d [--source-data] arg</code> | write JSON source data |
| <code>-i [--input] arg</code> | specify the Markdown file to parse |
| <code>-a [--ast] arg</code> | print the AST generated by the parser (for debugging purposes) |
| <code>-o [--override] arg</code> | specify the override text for the root document. parser behaves as if the file path is input, but uses this content instead |

9.1 Quality of Life Improvements

physics Package Support in Anki To enable physics package support in Anki, navigate first to Anki's installation directory. On Windows, this is located by default at `C:\Program Files\Anki`. On macOS, Anki is likely to be located under `/Applications`; right click on `Anki.app`, select `Show Package Contents` and navigate to `Resources`.

Once in Anki's installation directory on either platform, open `'aqt_data/web/js/mathjax.js'` and replace its contents with the following:

```

window.MathJax = {
  tex: {
    displayMath: [["\\[", "\\]"]],
    processRefs: false,
    processEnvironments: false,
    packages: {
      "[+]" : ["noerrors", "mhchem", "physics"],
    },
  },
  startup: {
    typeset: false,
    pageReady: () => {
      return MathJax.startup.defaultPageReady();
    },
  },
  options: {
    renderActions: {
      addMenu: [],
      checkLoading: [],
    },
    ignoreHtmlClass: "tex2jax_ignore",
    processHtmlClass: "tex2jax_process",
  },
  loader: {
    load: ["[tex]/noerrors", "[tex]/mhchem", "[tex]/physics"],
  }
};

```

Note 9.1. This is only a ‘temporary’ workaround and must be applied every time Anki is updated (until the Anki developers finally enable the `physics` package by default), since `mathjax.js` will be overwritten upon each update.

Increasing Polling Speed by AnkiConnect If you have hundreds of flashcards to send, you will notice that sending flashcards will take considerable time. This is because the **AnkiConnect** addon’s default polling interval is quite long. To change this, navigate to Anki’s addon directory, which by default is located at `C:\Users\%USERNAME%\AppData\Roaming\Anki2\addons21` in Windows, and at `~/Library/Application Support/Anki2/addons21` in macOS.

Once here, open `2055492159/util.py` and locate the line containing `'apiPollInterval': 25`. Change this to `'apiPollInterval': 1`, which will *hugely* speed up the card sending/updating process.

9.2 Smarty pants Replacements

| Sequence | Replacement |
|--------------|--------------|
| (p) or (P) | § |
| (tm) or (TM) | ™ |
| (r) or (R) | ® |
| (c) or (C) | © |
| <--> | ↔ |
| <-- | ← |
| --> | → |
| <==> | ↔ |
| <== | ← |
| ==> | → |
| <-> | ↔ |
| <- | ← |
| -> | → |
| <=> | ↔ |
| <= | ← |
| => | → |
| --- | — (em-dash) |
| -- | - (en-dash) |
| ... | … (ellipses) |
| +- | ± |

Table 9.1: Smarty pants replacements.

9.3 Known Issues and Annoyances

There are a plethora of issues in YuMD and in the VSCode extension.

1. The biggest cosmetic problem is the syntax highlighting. Although it offers a rudimentary styling of YuMD code, the flexibility of Markdown means that the syntax highlighting is often inaccurate for more complex document structures. It even breaks for simple things like spaces in citation before/after text. **Whenever the syntax highlighting breaks, this is not an indication of a mistake by you. Check the preview for a more accurate confirmation that YuMD is parsing your document correctly.**
2. Big documents hugely slow down the HTML preview. This is a combination of the web renderer with VSCode being very slow and it being impossible to incrementally update the preview — the entire document must be refreshed on every change.
3. Scroll sync is sometimes derpy.
4. The HTML preview is not a complete reflection of the LaTeX output. Some things may be laid out differently, and some things that look perfectly fine in HTML may result in an error from LaTeX (like footnotes within a caption).
5. All equations in LaTeX output are numbered, while no equations are numbered in HTML.
6. You have to close and reopen the preview whenever a change in styling is made.

7. The preamble is hardcoded, although it is possible to [append custom LaTeX code](#) after the hardcoded preamble.

9.4 Hardcoded Preamble

Below is the preamble with which YuMD begins every root `.tex` document (written to be fed into `sprintf`, so `%s` is a placeholder, while `%%` denotes an actual `%` symbol for a comment). It is possible to place custom LaTeX code after this (see Section 8).

```
\documentclass{report}
\usepackage[margin=1in]{geometry}
\usepackage[T1]{fontenc}
\usepackage{lmodern} %% must be loaded with fontenc --- otherwise text will become ↵
    bitmaps
\usepackage[%s]{babel}
\usepackage{csquotes} %% will automatically use quotes of the babel language
\usepackage{amssymb}
\usepackage{amsmath}
\usepackage{amsthm}
\usepackage{mathtools}
\usepackage{etoolbox}
\usepackage{tabularx}
\usepackage{booktabs}
\usepackage{listings}
\usepackage{color}
\usepackage[normalem]{ulem}
\usepackage[style=%s]{biblatex} %% verbose or apa depending on cite style
\usepackage{xcolor}
\usepackage[
    colorlinks,
    linkcolor=blue,
    citecolor=blue,
    urlcolor=blue
]{hyperref}
\usepackage[capitalize]{cleveref} %% use capitalization so that we do not need to worry↵
    about detecting beginnings of sentences, etc
\usepackage{import}
\usepackage[useregional]{datetime2}
\usepackage{titlesec}
\usepackage{physics}
\usepackage{float} %% for H option in table
\usepackage{parskip} %% remove indentation on new paragraphs
\usepackage{caption}
\usepackage{graphbox} %% allow align=t in \includegraphics to make them work better in ↵
    minipages if first item

\MakeOuterQuote{"}

\titleformat{\chapter}[display]{}{}{0pt}{\raggedright\normalfont\bfseries\huge\↵
    thechapter\hspace*{1em}}[] %% for chapters created from metadata blocks
\titleformat{name=\chapter,numberless}[display]{}{}{0pt}{\raggedright\normalfont\↵
    bfseries\huge\phantom{\thechapter}\hspace*{1em}}[] %% for unnumbered chapters, ↵
    including the table of contents
\titlespacing{\chapter}{0pt}{2em}{1em}
```

```

\lstset{
  backgroundcolor=\color[rgb]{1,1,1},
  tabsize=4,
  rulecolor=,
  basicstyle=\ttfamily,
  upquote=true,
  aboveskip={1.5\baselineskip},
  columns=fixed,
  showstringspaces=false,
  extendedchars=true,
  breaklines=true,
  prebreak = \raisebox{0ex}[0ex][0ex]{\ensuremath{\hookleftarrow}},
  showtabs=false,
  showspace=false,
  showstringspaces=false,
  identifierstyle=\ttfamily,
  keywordstyle=\color[rgb]{0,0,1},
  commentstyle=\color[rgb]{0.133,0.545,0.133},
  stringstyle=\color[rgb]{0.627,0.126,0.941},
  aboveskip=0pt,
  literate={\&}{\textsterling}}1 %% allow & sign within listings
}

\newcommand{\noncolouredtableofcontents}{
  \begin{group}
  \hypersetup{hidelinks}
  \tableofcontents
  \end{group}
}

\ifcsundef{thematicbreak}{\newcommand{\thematicbreak}{\par\bigskip\noindent\hrulefill\leftarrow
  \par\bigskip}}{}

\theoremstyle{definition}
\ifcsundef{definition}{\newtheorem{definition}{Definition}[section]}{}

\theoremstyle{plain}
\ifcsundef{theorem}{\newtheorem{theorem}{Theorem}[section]}{}
\ifcsundef{lemma}{\newtheorem{lemma}[theorem]{Lemma}}{}
\ifcsundef{corollary}{\newtheorem{corollary}{Corollary}[theorem]}{}

\theoremstyle{definition}
\ifcsundef{definition}{\newtheorem{definition}{Definition}[section]}{}
\ifcsundef{example}{\newtheorem{example}{Example}[section]}{}

\theoremstyle{remark}
\ifcsundef{assumption}{\newtheorem*{assumption}{Assumption}}{}
\ifcsundef{proof}{\newtheorem*{proof}{Proof}}{}
\ifcsundef{exercise}{\newtheorem{exercise}{Exercise}[section]}{}
\ifcsundef{problem}{\newtheorem{problem}{Problem}[section]}{}
\ifcsundef{question}{\newtheorem{question}{Question}[section]}{}
\ifcsundef{tip}{\newtheorem*{tip}{Tip}}{}
\ifcsundef{solution}{\newtheorem*{solution}{Solution}}{}
\ifcsundef{note}{\newtheorem{note}{Note}[section]}{}

```

```

\ifcsundef{derivation}{\newtheorem{derivation}{Derivation}[section]}{}
\ifcsundef{axiom}{\newtheorem{axiom}{Axiom}[section]}{}
\ifcsundef{conjecture}{\newtheorem{conjecture}{Conjecture}[section]}{}
\ifcsundef{hypothesis}{\newtheorem{hypothesis}{Hypothesis}[section]}{}
\ifcsundef{proposition}{\newtheorem{proposition}{Proposition}[section]}{}

\ifcsundef{remark}{\newtheorem*{remark}{Remark}}{} %% notes are numbered but remarks  $\leftarrow$ 
are not

\renewcommand{\qedsymbol}{\text{\blacksquare$}} %% closed black square for proof environments

\renewcommand\thesection{\arabic{section}}

\numberwithin{equation}{section}
\numberwithin{figure}{section}
\numberwithin{table}{section}

\providecommand{\tightlist}{%%
  \setlength{\itemsep}{0pt}\setlength{\parskip}{0pt}}

%% provide maxwidth option for includegraphics
\makeatletter
\def\maxwidth#1{\ifdim\Gin@nat@width>#1 #1\else\Gin@nat@width\fi}
\makeatother

%% fix parskip within minipage
\setlength{\parskip}{\medskipamount}
\makeatletter
\newcommand{\@minipagerestore}{\setlength{\parskip}{\medskipamount}}
\makeatother

```

Bibliography

- De Moor, T. & Van Zanden, J. L. (2010). Girl power: The european marriage pattern and labour markets in the north sea region in the late medieval and early modern period. *The Economic History Review*, 63(1).
- Sargent, T. J. (2009). *The ends of four big inflations*. University of Chicago Press.