

Documentation for YuMD

Created using YuMD

12th December 2021

Contents

1	Introduction	2
1.1	Why?	2
1.2	What?	2
2	Sending Notes to Anki	3
2.1	Setup	3
2.2	Sending Notes	3
2.3	Note IDs	3
3	VSCoDe Extension	5
3.1	Preview	5
3.2	VSCoDe Extension Preferences	6
4	Compiling LaTeX	6
5	Syntax	6
6	Extended Syntax	7
6.1	Theorems	7
6.2	Definition Lists	8
6.3	Tables	9
6.4	Footnotes	13
6.5	References	13
6.6	Section Heading	14
6.7	Citations	14
6.8	Smartypants	15
6.9	Including other .yumd files	15
6.10	Introduction	17
6.11	Conclusion	17
6.12	LaTeX Math	18
6.13	Figures	18
6.14	HTML Code	19
6.15	LaTeX Code	19
7	Quality of Life Improvements	19
7.1	Physics Package support in Anki	19
8	YuMD File Configuration	20
8.1	Configuration Options	20
9	Defining Anki Flashcards	21
9.1	Explicit Flashcard Generation	21
9.2	Default Deck	23
9.3	Inferred Fields	23
9.4	Inferred note type	24
9.5	Implicit Flashcards	25
9.6	Inline Occlusion (Cloze Flashcards)	27
10	Command Line	29
11	Known Issues	30

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 to

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.

The documentation for YuMD is itself written in YuMD, and can be viewed [here](#).

1 Introduction

1.1 Why?

YuMD was created as a solution to the following trilemma.

Proposition 1.1 (Trilemma of notetaking). It is impossible to have the following three at the same time from the same base of notes:

1. good formatting, preferably in pdf form (such as using LaTeX)
2. flashcard form
3. fast notetaking with support for complex elements including tables, theorems, figures, etc.

Existing solutions only allow for up to two of these at one time:

1. Notion's intuitive UI allows for quick notetaking with mildly good-looking formatting but offers no way of producing flashcards
2. Remnote 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. pandoc, like Notion, has no flashcard support

Some solutions are even worse and only satisfy one of the three conditions:

1. vanilla markdown is hugely limited and has substandard or non-existent support for tables, theorems, and many other things necessary for rich notes
2. LaTeX creates good-looking documents but is a pain to type and complex formatting requires complex workarounds (such as nested tables)

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

YuMD aims to solve this, first by extending markdown to include theorems, tables, footnotes, citations, references, and other important features. YuMD then provides options to:

1. computationally generate LaTeX code, savings huge amounts of typing and debugging time
2. identify, send and sync flashcards to Anki from within your notes

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. For this reason, YuMD is a suitable program for writing reports and thus also useful for those who aren't interested in flashcards.

1.2 What?

The term YuMD refers to the broad notetaking system outlined above. Physically, it also 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`), which provides a command-line interface to this library, and
2. a VSCode extension (`yumd.vsix`).

Both can be obtained freely [here](#), although source code is not public for now.

Note 1.1. For now, YuMD will stay closed-source, but this decision is open for review in future.

Note 1.2. There is no current plan to compile native binaries for Apple silicon. Macs running on Apple silicon should nonetheless be able to run the binaries through the Rosetta 2 compatibility layer.

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

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

2 Sending Notes to Anki

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

2.1 Setup

For a rundown on the syntax for defining flashcards, see Section 9.

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

2.2 Sending Notes

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

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. 1.

Note 2.1. You may be confronted with the error message `Open a YuMD file to send to Anki`. This is a drawback of VSCode in that 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*.

Alternatively, notes can be sent to Anki via the command line using:

```
/path/to/yumd_executable path/to/document.yumd --anki
```

Note 2.2. 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.

2.3 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.

Note 2.3. Avoid tampering with the note ID as this is the only thing pointing YuMD towards the flashcard in Anki.

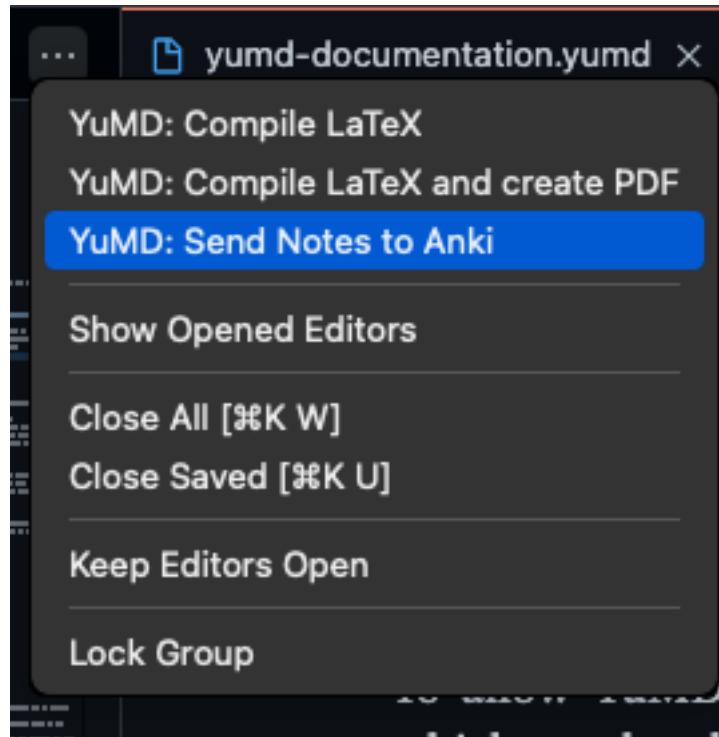


Figure 1: Sending notes to Anki through VSCode.

```

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

```

Figure 2: Note ID added by YuMD after sending flashcards to Anki. Avoid touching.

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 2.4. Never make changes to flashcards within Anki itself. All your changes will be overwritten by the contents of the YuMD file 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, and rerun YuMD: **Send Notes to Anki**. Upon deletion, YuMD will replace the note ID with the tag ‘ignore’, as in Fig. 4, which prevents it from recreating the flashcard the next time you run YuMD: **Send Notes to Anki**.

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

Figure 3: Append ‘delete’ after the note ID to mark it for deletion.

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

Figure 4: After deletion, YuMD automatically replaces the tag with ‘ignore’ to avoid recreation of the flashcard upon next run.

Note 2.5. Avoid deleting notes directly within Anki without manually changing the flashcard tag to ‘ignore’.

3 VSCode Extension

3.1 Preview

The VSCode extension provides real-time preview by running the command YuMD: **Show YuMD Preview**, or by clicking the preview icon, as shown in Fig. 5

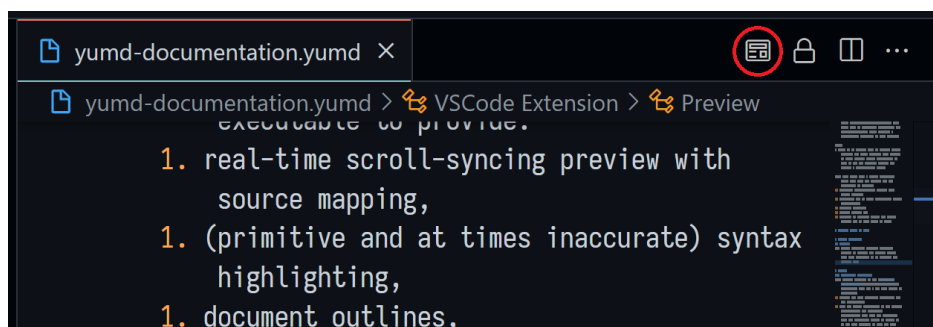


Figure 5: Click to show YuMD preview.

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.

Locked Preview Locked mode can be activated using the command `YuMD: (Un)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.

In locked mode, the preview reads only what is saved to disk. This means that the preview updates 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 to enable locked mode to produce previews on save, which will substantially reduce the lag.

3.2 VSCode Extension Preferences

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

It is highly recommended to change the default font, by searching for `YuMD: preview font family`. The font "Stix Two Text", which can be downloaded [here](#) is highly recommended.

A custom CSS stylesheet is also supported. 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.

4 Compiling LaTeX

When compiling `.yumd` files into LaTeX, YuMD copies dependencies (images and bibliographies) into the target directory, preserving the directory structure.

To compile YuMD 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 → biber → pdflatex` to generate the pdf file automatically.

Note 4.1. See [Note 2.2](#).

5 Syntax

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. For this reason, **all indentation in YuMD should be done using spaces to avoid unexpected results.**

6 Extended Syntax

CommonMark by itself is largely enough for writing prose, but since we are not English students, CommonMark alone can be quite limiting. This section details the additional components introduced by YuMD which make it amicable for actual notetaking and even report writing.

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 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
3. blank lines can exist between block elements belonging to the theorem, but must be at the same indentation as the first block element after the colon.
4. unless it is the first line of the file, there must be a blank line before the theorem syntax 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. question or que,
15. remark or rmk,
16. theorem or thm, and

17. tip.

`<theorem name>` is optional and is typeset in brackets beside the theorem type. This is useful for naming theorems and specifying terms to define, allowing for quick flashcard creation. The exception here is with the proof environment, where by convention, entire ‘*Proof*’ text is replaced with the text given in brackets. This is handy in defining your own ‘theorem’ environments.¹

<pre>Theorem (Fermat's little). : If \((p)\) is a prime \leftrightarrow number, then for every integer \leftrightarrow \((a)\), \[a^p \equiv a \pmod{p}. \]</pre>	<p>Theorem 6.1 (Fermat’s little). <i>If p is a prime number, then for every integer a,</i></p> $a^p \equiv a \pmod{p}. \quad (6.1)$ <p><i>Proof.</i> Left as an exercise to the reader. ■</p>
<pre>Prf. : Left as an exercise to \leftrightarrow the reader.</pre>	

<pre>Definition (economics). : There does not exist one\leftrightarrow .</pre>	<p>Definition 6.1 (economics). There does not exist one.</p>
<pre>Proof (PProof of no single\leftrightarrow definition). : Ask Cambridge \leftrightarrow professors and you will either: 1. receive an unsatisfactory answer\leftrightarrow , or 1. receive no answer.</pre>	<p><i>Proof of no single definition.</i> Ask Cambridge professors and you will either:</p> <ol style="list-style-type: none">1. receive an unsatisfactory answer,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.

6.2 Definition Lists

Although it is recommended to use the definition theorem to indicate terms and definitions, YuMD implements definition lists, the syntax for which is:

```
<term>
: <definition>

<term>
: <definition>

<term>
<term>
: <definition>
```

¹a footnote which should work

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

simply using a newline \leftarrow
to define your \leftarrow
paragraph.

: No penalties for using \leftarrow
extra colons though.

Remark. You can indeed define multiple entries,
but this has no effect and you are better off
simply using a newline to define your paragraph.
There are no penalties for using extra colons though.

```
<term>
: <definition>
: <definition>
```

```
<term>
<term>
<term>
: <definition>
: <definition>
```

Critically,

1. a colon is required on a newline to mark each new definition
2. definitions can contain block content, but each newline must be indented to the same level as the first
3. 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.

6.3 Tables

The area where YuMD deviates from other markdown implementations the most is with regard to tables. The most popular method, as implemented by GitHub 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 `+`s. For a valid table, you must use only `+`s. Block elements can be used within cells if indented correctly.

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

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.).

Note 6.2. The newline between each table entry is entirely optional. The following examples omit use of newlines.

Term
: Your definition goes \leftarrow
here,
and can contain block \leftarrow
content,
such as
> blockquotes,
but must be indented \leftarrow
correctly.

New Term
: A new term/definition \leftarrow
group
must be separated by a \leftarrow
newline.

Multiple
Terms
Here
: are assigned
: to multiple
: definitions here. But \leftarrow
this
is probably a useless \leftarrow
feature,
implemented only for
completeness.

Term Your definition goes here, and
can contain block content, such as
blockquotes, but must
be indented correctly.

New Term A new term/definition
group must be separated by a
newline.

**Multiple
Terms**
Here are assigned
to multiple
definitions here. But this is
probably a useless feature
implemented only for
completeness.

+ + Table
+ Data

+ + Mathematics:
 $\backslash[F(x) \equiv \int_0^x f(t) dt]$
 $\backslash\dd{t}$

+ Blockquotes:
> Insert inspirational
quotation here.

Table	Data
Mathematics:	Blockquotes:
$F(x) \equiv \int_0^x f(t) dt$ (6.2)	Insert in- spir- a- tional quo- ta- tion here.

```
+ + Person A
+ **Econ**
+ *Pembroke*
```

```
+ + Person B
+ **Eng**
+ *Trinity*
```

```
+ + Person C
+ **Compsci**
+ *Girton*
```

```
+ + Subject choices for
    person \(\i \in
    \qty{A, B, C}\).
```

Person <i>A</i>	Econ	<i>Pembroke</i>
Person <i>B</i>	Eng	<i>Trinity</i>
Person <i>C</i>	Compsci	<i>Girton</i>

Table 1: Subject choices for person $i \in \{A, B, C\}$.

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 centered and bold.

```
+ + **Name**
+ **Age**
+ + Benson
+ 19
+ + Maxwell
+ 20
```

Name	Age
Benson	19
Maxwell	20

Alignment Row Cell alignments can be defined if the very first row contains cells with text `l|c|r|j`. This will align the rest of each column accordingly. By default, left alignment is used if no alignment row is given.

```
+ + l
+ r
+ + Benson
+ 19
+ + Maxwell
+ 20
+ + Universe
+ *13.7 trillion*
```

Benson	19
Maxwell	20
Universe	<i>13.7 trillion</i>

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

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 `+`.

However, although the table body is transposed, heading and alignment rows remain interpreted by row.

+ + l	
+ + r	
+ + **Name**	Benson
+ + **Age**	19
+ + Benson	Maxwell
+ + 19	20
+ + Maxwell	Universe
+ + 20	<i>13.7 trillion</i>
+ + Universe	
+ + *13.7 trillion*	

- - Oil	Oil	\$3
- - Soy sauce	Soy sauce	\$1
- - Hoisin sauce	Hoisin sauce	\$2
- - Ramen	Ramen	\$2
- - \$3		
- - \$1		
- - \$2		
- - \$2		

- - l	Name	Subject	College
- - c	Lorem	Engineering	Pembroke
- - r	Ipsum	Computer science	Kings
- - **Name**	Dolor	Economics	Trinity
- - **Subject**			
- - **College**			
- - Lorem			
- - Ipsum			
- - Dolor			
- - Engineering			
- - Computer science			
- - Economics			
- - Pembroke			
- - Kings			
- - Trinity			
- - Subjects and colleges.			

Table 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.

I enjoy economics. [[^] I really <i>*do</i> not*.]	I enjoy economics. ^a
	<hr/> ^a I really <i>do not</i> .

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>  
  <block content>  
  
  <block content>
```

Critically,

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

Footnotes are then referred to using the syntax `[^footnote-label]`. Note that this creates potential conflict with inline footnotes as the syntax is almost identical: if `<footnote-label>` is not defined or has spaces, then it will be interpreted as an inline footnote.

Note 6.3. The ordering of footnote definition and reference does not matter. Footnotes can be referred to before they are defined, as in the example below.

See footnote. [[^] complex-↔ footnote]	See footnote. ^a
 [[^] complex-footnote]: This is a very ↔ complicated footnote, involving 1. > blockquotes 1. and lists	 <hr/> ^a This is a very complicated footnote, in- volving 1. blockquotes 2. and lists

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 theorems in LaTeX.

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

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
2. simply `[#<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.

```

Please see [#section-↵
    heading] for
important facts and ↵
    figures.

# Section Heading #section↵
    -heading

Theorem. #a-theorem
: An important theorem.

- - Micro
- - Macro
- - Maths
- - Block
- - Tambakis
- - Robertson
- - An important table. #↵
    important-table

![True art.](./figures/pem↵
    .png) #wonder

In particular,
[this important theorem](#↵
    a-theorem)
is worth remembering. We ↵
    would
also recommend a reading ↵
    of
[#important-table], and
a detailed study of
[this work of art](#wonder↵
    ).

```

Please see Section 6.6 for important facts and figures.

6.6 Section Heading

Theorem 6.2. *An important theorem.*

Micro	Block
Macro	Tambakis
Maths	Robertson

Table 3: An important table.

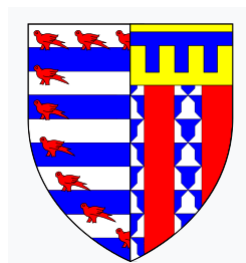


Figure 6: A true work of art.

In particular, [this important theorem](#) is worth remembering. We would also recommend a reading of [Table 3](#), and a detailed study of [this work of art](#).

6.7 Citations

YuMD allows you to include BibTeX `.bib` bibliography files and use citations either in ‘in-text’ style by default, or in ‘footnote’ style. Bibliographies are included by specifying a path to the bibliography in the YAML header or corresponding `.yucfg` file (see YuMD file configuration), and optionally specifying the citation style:

```

---
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.

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

Note 6.4. Paths can be specified relative to the current `.yumd` file, or absolutely. Note that environment variables and placeholders like `~` are not respected by YuMD.

Note 6.5. 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.

<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 6.6. 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 Smartyants

By default, YuMD implements a plethora of ‘typography’ substitutions which are detailed in Table 4. This can be turned off by specifying `typography: false` in the YAML header or corresponding `.yucfg` file.

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 current `.yumd` file, or absolutely.

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

Note 6.7. The leading `./` is unnecessary when using relative paths.

Note 6.8. Including files which do not exist will issue a warning and have no effect on the final LaTeX or HTML output.

Note 6.9. Included `.yumd` files inherit the configuration of parent `.yumd` files. Configuration is merged with (in the case of lists, such as bibliographies) or overwrites 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 6.10. 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.

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

Table 4: Smartypants replacements.

Assuming the following file structure:

```
report.yumd
sections/
  introduction.yumd
  conclusion.yumd
quotations/
  friedman.yumd
```

In report.yumd (the root file):

```
# Introduction
[[sections/introduction.↵
  yumd]]
# Conclusion
[[sections/conclusion.yumd↵
  ]]
```

In sections/introduction.yumd:

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

In sections/conclusion.yumd:

Conclusion goes here.

In quotations/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.

6.12 LaTeX Math

YuMD supports math-mode LaTeX. Inline math can be typeset by surrounding it either with dollar signs $or with escaped brackets $\(and $\). Display math can be typeset with either double dollar signs $or with escaped square brackets $\[and $\]. By convention Brackets are preferred to dollar signs.$$$$$$

Display math is automatically typeset as an `align` environment, so multiples 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:

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


$$\mathrm{All\ variance-covariance\ matrices\ are\ positive\ semi-definite.}$$

```

Results in:

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

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

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

All variance-covariance matrices are positive semi-definite.

Note 6.12. 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 6.13. 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 workaround can be considered in a future update.

Note 6.14. 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 use **physics** by default. Section 7.1 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.

6.13 Figures

Images YuMD supports including images using the syntax `![<figure caption>](./path/to/image.png)`

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

We have:	We have:
$U(x, y) = \min\{ax, by\}$	$U(x, y) = \min\{ax, by\} \quad (6.5)$
Utility is maximised where ax and by in Eq. (6.5) are equal.	Utility is maximised where ax and by in Eq. (6.5) are equal.

```
+ [<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 extension settings (see YuMD extension settings), a file watcher detects and automatically generates and updates corresponding `.pdf_tex` and `.pdf` files whenever `.svg` files are created or updated. The VSCode extension also makes it easy to create and edit SVGs linked to a `.yumd` document: ctrl-clicking the path opens the SVG in Inkscape or creates a blank SVG if it does not exist.

6.14 HTML Code

Unlike standard Markdown, where all HTML is treated as valid markdown, YuMD only supports partial treatment of HTML. HTML code works and is printed verbatim when outputting HTML, but is treated as a code block when outputting LaTeX. We would recommend avoiding using raw HTML in your YuMD files.

6.15 LaTeX Code

Other than [math mode](#), YuMD does not support direct LaTeX code. All backslashes in plaintext are escaped when outputting LaTeX.

7 Quality of Life Improvements

7.1 Physics Package support in Anki

Replace it with this code

```
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"],
    },
  };

```

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 starts with three hyphens and ends with three periods.

```

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

```

Critically, the three hyphens must be the first line of the `.yumd` file.

.yucfg Configuration File Alternatively, a configuration file with the same filename as the `.yumd` file can be created. *If you install the YAML language features extension by Red Hat in VSCode, this has the advantage of providing autocomplete in case you forget which configuration options are available, which I do all the time.*

The `.yucfg` file does not need to begin with `---` and end with `...`, although it can.

8.1 Configuration Options

Below is a complete configuration file. All entries are optional.

```

title-type: document

title: YuMD Notes
subtitle: ""
ccode: ""
subject: ""
lecturer: ""
author: ""
season: ""
part: ""
date: 2021-01-01
date-end: 2021-12-10
place: Cambridge

section-offset: section

```

```

bibliography: ""

cite-style: in-text
language: british

flashcards-as-table: auto # auto, always or never
typography: true # see smartypanants_element.h

html:
  smart-quotations: true

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

latex:
  toc: false
  break-after-toc: true
  tocdepth: 3
  secnumdepth: 3

```

9 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 lay out flashcards, so it is well worth reading this entire section.

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

9.1 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 3 hyphens. The following code is therefore valid:

title-type: [document chapter none] (default: none)	<ol style="list-style-type: none"> 1. If set to document, YuMD will print all of title, subtitle, ccode, subject, lecturer, author, season, part, date, date-end, and place in the form of a large title page 2. If set to chapter, YuMD will print title, subtitle, lecturer, date and date-end in the form of a smaller title section. 3. If set to none, 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, eg. 'Michealmas 2021'.
date and date-end	Must be formatted in big-endian form YYYY-MM-DD to allow it to be printed nicely in LaTeX and HTML.
section-offset: [part chapter section subsection paragraph subparagraph 0 1 2 3 4 5] (default: section)	Specify the level of heading corresponding to a single-hash heading # Heading .
bibliography	Specify the path to included bibliographies. Can be a string or an array of strings for multiple bibliographies.
cite-style: [in-text footnote] (default: in-text)	Specify the citation style. YuMD only considers cite-style specified in the root document and applies it to all included .yumd files.
language (default: british)	Specify the babel language for LaTeX.
flashcards-as-table: [auto always never] (default: auto)	<ol style="list-style-type: none"> 1. If set to always, YuMD will format flashcards as tables with label in the first column and content in the second column. 2. If set to auto, 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 never YuMD will output flashcards as normal document body content.
html: smart-quotations: [true ↵ false]	Use typographers' quotations (fancy curly " and ') in HTML output.
(default: true)	
anki: deck: <...>	Specify the deck to which YuMD should send flashcards by default.
(default: Default)	
	Specify the note type when YuMD infers

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

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

Note 9.1. 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.

9.2 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 YAML header at the top of the file:

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

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

Thus, the following code:

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

::Custom Note
  <content>
```

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

9.3 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 9.2. 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 9.3. The first field must have content, but content in remaining fields can be left blank.

Note 9.4. If you specify a greater number of fields than exists, YuMD will ignore the superfluous fields. If you specify a named field which does not exist, YuMD will issue an error and ignore that flashcard. If you specify fewer fields than exists, the remaining fields are left blank or left unchanged.

9.4 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 if you leave `<note type>` blank.

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** be default.

Thus, the code

```

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

```

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

You can specify the default cloze note type in the YAML header or configuration file using

```

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

```

Theorems If the flashcard contains a single unspecified field and the only item within that field is a named theorem (theorem, definition, proof, corollary, etc.), then YuMD will create a front-back flashcard type, separating the theorem name, which functions as a prompt, and theorem body.

Thus, the code

```

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

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

```

creates the flashcard

By default, YuMD sends a flashcard of type **Basic** to Anki, but this can be changed using configuration

```

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

```

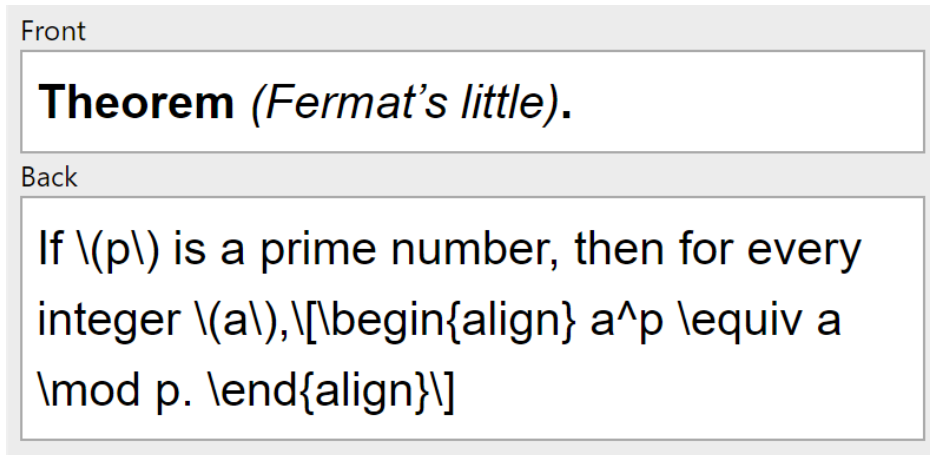


Figure 7: sdf

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 front-back flashcard.

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

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

Note 9.5. Cloze note type takes precedence of theorems and definition lists: if you have clozes within theorems within the flashcard, YuMD will infer type cloze rather than theorem, which is likely the most desired behaviour.

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 `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.

9.5 Implicit Flashcards

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

Inline Cloze Paragraphs with cloze text are automatically turned into flashcards in the same manner as described in Section 9.4. This means that throwing a few `{s` and `}s` into your notes allows you to rapidly create fill-in-the-blank style flashcards.

For example, the code

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

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

generates two notes (three cards) in Anki:

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 8: Cloze cards generated within Anki.

Theorems and Definition Lists Named theorems and definition lists are turned into flashcards in the same manner as described in Section 9.4, without needing to explicitly reside within a flashcard. Note that only named theorems are sent to Anki — unnamed theorems provide no prompt for recall and so are meaningless as a flashcard.

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

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

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

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

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  $\leftrightarrow$  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, like so:

```
::
In the balance sheet:
1. assets denote {anything valuable owned by the institution} (+),
1. liabilities denote {anything valuable that the institution owes  $\leftrightarrow$  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` 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. There are two ways of achieving this.

In the first method, you can explicitly place multiple theorems within a single flashcard and use 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.

```
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  $(f(c) = c)$ .
```

9.6 Inline 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. For example, the code

```
When short term rates are near zero, open market operations are near {c1::ineffective} as there is an {c1::indifference between bonds and cash}.
```

Generates a single flashcard where text ‘ineffective’ and ‘indifference between bonds and cash’ is to be recalled.

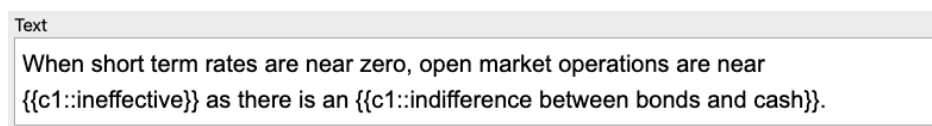


Figure 9: Basic inline occlusion.

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. For example, the code

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

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’.

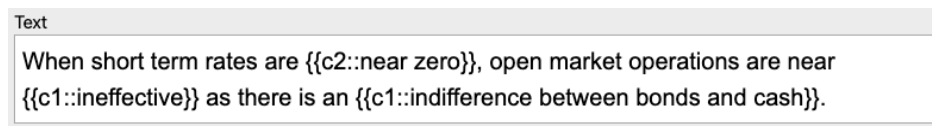


Figure 10: Cloze indices.

Placeholders are also supported, which function as a prompt. This is done by appending `<placeholder text>` just before the closing `}`, where crucially, `<placeholder text>` is in plaintext.

For example, the code

```
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}.
```

results in ‘types of money’ being printed on the front side of card 1 rather than ‘[...]’:

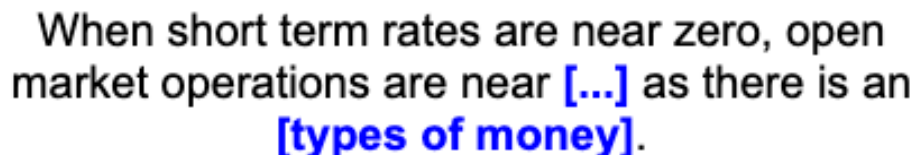


Figure 11: Plaintext placeholders.

Placeholders help to avoid ambiguity when there could be multiple things that fill a gap.

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.

For example, the code

```
::
  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
  }
```

produces the flashcard

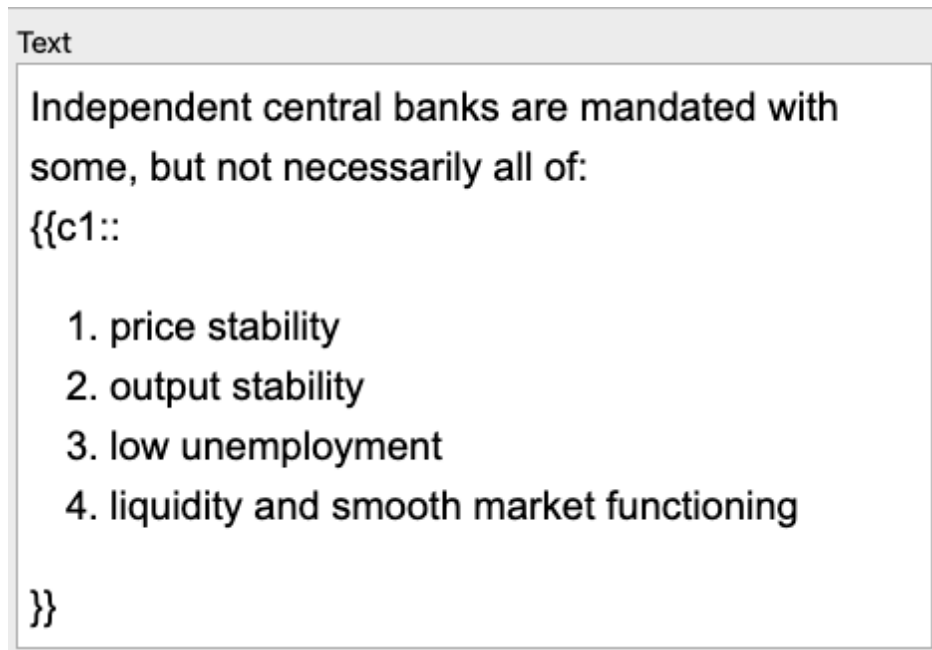


Figure 12: Flashcard created using block cloze.

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

```
::
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
:4 mandates}

A non-independent central bank may be used to {2:serve the interests ←
    of the current government}, such as through seigniorage: if a ←
    government cannot either:
{2:
1. raise taxes, or
1. issue bonds,
:2 ways of raising revenue}

then it may resort to seigniorage.
```

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

10 Command Line

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:

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

11 Known Issues

Extension runs pdflatex twice. Gap between. Forces you to use pdflatex.

‘hello’

‘how do you handle this’

“or this”

‘or this’

Proposition 11.1 (works?). this is hard. why is this not here.

In inline code, --`hello`- gets messed up

BB§

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.