

Markdown, Wikitext, SGML & XML

Charlotte Tupman

Markdown & Wikitext

What is Markdown?

- Contrary to its name, markdown is not the ‘opposite of markup’ (whatever that would be!) but is, rather, a **specific markup language that has a simple syntax**
- Its aim is to allow you to create a structured document with a small number of characters
- Designed to be unobtrusive
- It focuses on the formatting of the document
- Created in 2004 and used today by Reddit, GitHub, SourceForge, OpenStreetMap and many other sites

Written	Rendered
<code>_italic_</code> or <code>*italic*</code>	<i>italic</i>
<code>__bold__</code> or <code>**bold**</code>	bold
<code>___bold-italic___</code> or <code>***bold-italic***</code>	<i>bold-italic</i>
<code>~~strikethrough~~</code>	strikethrough
<code>>!spoilers!<</code>	<div></div>
<code>^superscript</code> or <code>^(superscript)</code>	^{superscript}
<code>`code`</code>	code

A ‘cheatsheet’ of some Reddit markdown
https://www.reddit.com/wiki/markdown#wiki_new_reddit-flavored_markdown

Flavours of markdown

- Markdown is not completely standardised
- It has an informal specification and there are a number of different implementations, often created because users have wanted to add new features particular to their needs
- These can be thought of as variants or 'flavours' of markdown, e.g. Python markdown, Stack Overflow markdown, GitHub flavoured markdown. You can compare these using a tool like Babelmark

<https://babelmark.github.io>



Compare Markdown Implementations (FAQ)

```
~~~strikethrough~~~
```

CONVERT CTRL ENTER

☒ Normalize

Results 22 implementations

#0

[Blogdown](#) 0.2.4 Haskell
[cebe/edra](#) 1.2.0 Php
[cebe/markdown](#) 1.2.0 Php
[cheapskate](#) 0.1.0.5 Haskell
[commonmark.js](#) 0.28.1 Javascript
[kramdown](#) 1.2.0 Ruby
[league/commonmark](#) 1.3.0 Php
[lunarmark](#) 0.4.0 Lua

```
1 <p>
2 ~~~strikethrough~~~
3 </p>
```

```
~~~strikethrough~~~
```

Results 12 implementations

#1

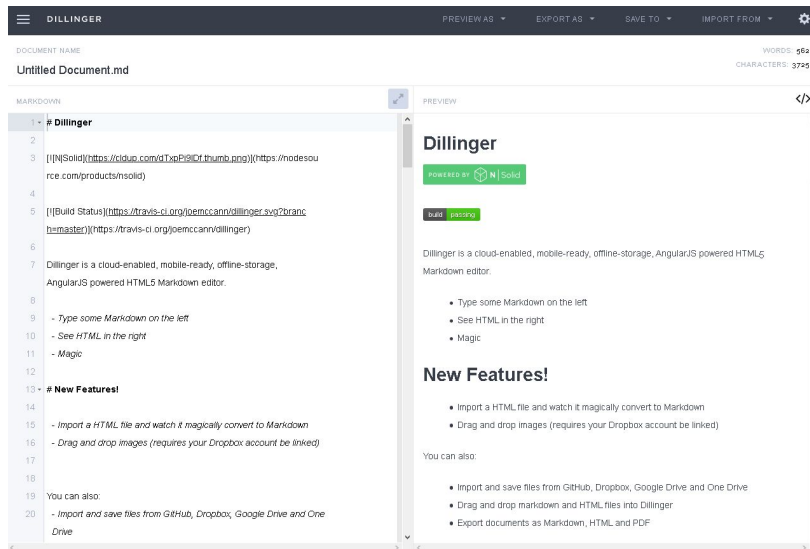
[cebe/gfm](#) 1.2.0 Php
[DFM-Latest](#) 2.46.0.0 C#
[DFM](#) 2.46.0.0 C#
[GitHub Flavored Markdown](#) 0.21.0 C
[league/commonmark GFM](#) 1.3.0 Php
[markdig \(advanced\)](#) 0.22.0.0 C#
[Markdig \(DocFX\)](#) 2.46.0.0 C#
[Marked](#) 1.2.0 Javascript
[pandoc](#) 2.11.0.1 Haskell
[parsedown](#) 1.8.0-beta-7 Php
[rdiscount](#) 2.1.8 Ruby
[s9e/TextFormatter \(Fatdown/PHP\)](#) Php

```
1 <p>
2 <del>strikethrough</del>
3 </p>
```

```
strikethrough
```

Editing markdown

- Unlike a WYSIWYG editor, you don't see the changes taking place in front of you, but markdown editors give a preview, e.g. Dillinger <https://dillinger.io>, StackEdit <https://stackedit.io>
- Markdown is non-proprietary, can be opened by a large number of applications, and can be used with any operating system: widely supported
- There are tools for converting between formats, e.g. Pandoc <https://pandoc.org>



Dillinger lets you type markdown in the left-hand pane and view its output in the right-hand pane

What is **Wikitext**?

- **Wikitext** is the markup language used to format pages in wikis (& it can also be used to refer to a document written in wiki markup language)
- It's sometimes known as wiki markup or wikicode
- Wikitext, like markdown more generally, aims to achieve a simplified way of creating structure in your document
- There's no commonly accepted standard wikitext language (although some use markdown) - this depends on what wiki software is used on a site.

Some examples of **wikitext** from Wikimedia

What it looks like	What you type
<p><i>Headings</i> organize your writing into sections. The <i>Wiki</i> software can automatically generate a <i>table of contents</i> from them.</p> <p>Subsection</p> <p>Using more "equals" (=) signs creates a subsection.</p> <p>A smaller subsection</p> <p>Don't skip levels, like from two to four equals signs.</p> <p>Start with 2 equals signs, not 1. If you use only 1 on each side, it will be the equivalent of h1 tags, which should be reserved for page titles.</p>	<p>== Section headings ==</p> <p>"Headings" organize your writing into sections. The "Wiki" software can automatically generate a [[help:Section table of contents]] from them.</p> <p>=== Subsection ===</p> <p>Using more "equals" (=) signs creates a subsection.</p> <p>==== A smaller subsection ====</p> <p>Don't skip levels, like from two to four equals signs.</p> <p>Start with 2 equals signs, not 1. If you use only 1 on each side, it will be the equivalent of h1 tags, which should be reserved for page titles.</p>
<ul style="list-style-type: none"> • <i>Unordered lists</i> are easy to do: <ul style="list-style-type: none"> • Start every line with an asterisk. • More asterisks indicate a deeper level. <p>Previous item continues.</p> • A newline • in a list <p>marks the end of the list.</p> <ul style="list-style-type: none"> • Of course you can start again. 	<p>* "Unordered lists" are easy to do:</p> <p>** Start every line with an asterisk.</p> <p>*** More asterisks indicate a deeper level.</p> <p>*. Previous item continues.</p> <p>** A newline</p> <p>* in a list</p> <p>marks the end of the list.</p> <p>*Of course you can start again.</p>
<p>1. <i>Numbered lists</i> are:</p> <ol style="list-style-type: none"> 1. Very organized 2. Easy to follow <p>A newline marks the end of the list.</p> <ol style="list-style-type: none"> 1. New numbering starts with 1. 	<p># "Numbered lists" are:</p> <p>## Very organized</p> <p>## Easy to follow</p> <p>A newline marks the end of the list.</p> <p># New numbering starts with 1.</p>

https://meta.wikimedia.org/wiki/Help:Wikitext_examples

SGML

What is **SGML**?

- SGML = Standard Generalised Markup Language
- It is a **standard for defining markup languages** for documents (so, not a markup language in itself, but a method of describing them)
- Originally designed for technical documentation
- It is **declarative** rather than presentational: it describes the document, rather than specifies how it should be processed. In other words, it carries no implications about how a document should be formatted: declaration and formatting instructions are two separate tasks
- Designed to be both computer- and human-readable

What is SGML?

- SGML does not define precisely what tags mean, in the sense of how they are to be processed: it doesn't tell you anything about how a text must ultimately be rendered (i.e. it is *not* a text formatting system)
- Rather, it describes the properties and inter-relations of components within a document: for instance, whereabouts a particular component appears within a document, and in relation to which other components

```
<anthology>
  <poem><title>The SICK ROSE
  <stanza>
    <line>O Rose thou art sick.
    <line>The invisible worm,
    <line>That flies in the night
    <line>In the howling storm:
  <stanza>
    <line>Has found out thy bed
    <line>Of crimson joy:
    <line>And his dark secret love
    <line>Does thy life destroy.

  <poem>
    <!-- more poems go here  -->

</anthology>
```

Example SGML from <https://tei-c.org/Vault/GL/P3/SG.htm>

What is **SGML**?

- This is easiest to imagine by thinking about how a document can be structured in sections, with headings, subheadings, paragraphs etc. Within the content of those headings, subheadings or paragraphs you might find (e.g.) names and dates. SGML allows you to describe these with tags, e.g.

`<para><name>Henry</name> visited us in <date>2019</date></para>.`

- This markup would allow us to process all names and dates *as names and dates* regardless of how we might choose to format them (e.g. bold, italic, highlighted in green, etc.). If we wanted to, we could even process names and dates inside a paragraph differently from those that appear inside a heading.

Document Type Definition

- SGML uses a **Document Type Definition** (DTD) to specify what is ‘legal’ within any particular component of a markup language. In this respect it functions a bit like a grammar does for human languages
- A DTD helps to control how those components, known formally as **elements**, should appear within a particular class of texts
- The specification of what is legal within each particular element is called its **content model** (as it provides a model for the element’s content)
- Here’s an example of an element declaration within a DTD for `<para>`:

```
<! ELEMENT para - - (name, date?)>
```

- This means that `<para>` *must* contain `<name>` and *may* contain `<date>`

XML

What is **XML**?

- XML = eXtensible Markup Language
- It is a simplification, or derivation, of SGML
- It is a declarative or semantic markup language, allowing us to recording the **meanings** of our texts, not only the way they should look or be processed
- Can be transformed to create a variety of outputs
- It imposes a greater number of restrictions than SGML does, but it does not require a DTD: in other words, **XML can be 'well-formed' without specific elements being declared** (more on that in a moment)

Basic XML syntax: an **element**

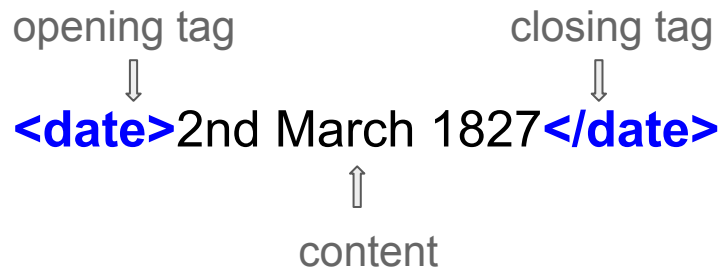
opening tag closing tag

↓ ↓

<date>2nd March 1827**</date>**

↑

content

A diagram illustrating the basic XML syntax for an element. The text "<date>2nd March 1827</date>" is shown in blue. Above the opening tag "<date>", the text "opening tag" is written in grey, with a downward-pointing arrow indicating its position. Above the closing tag "</date>", the text "closing tag" is written in grey, with a downward-pointing arrow indicating its position. Below the content "2nd March 1827", the text "content" is written in grey, with an upward-pointing arrow indicating its position.

An **empty element** contains no content

`<pb/>`

(an XML document *can* consist entirely of markup)

```
<seismographReadings>
```

```
  <event date="2009-04-11T05:43:22" id="01" richter="2.1"/>
```

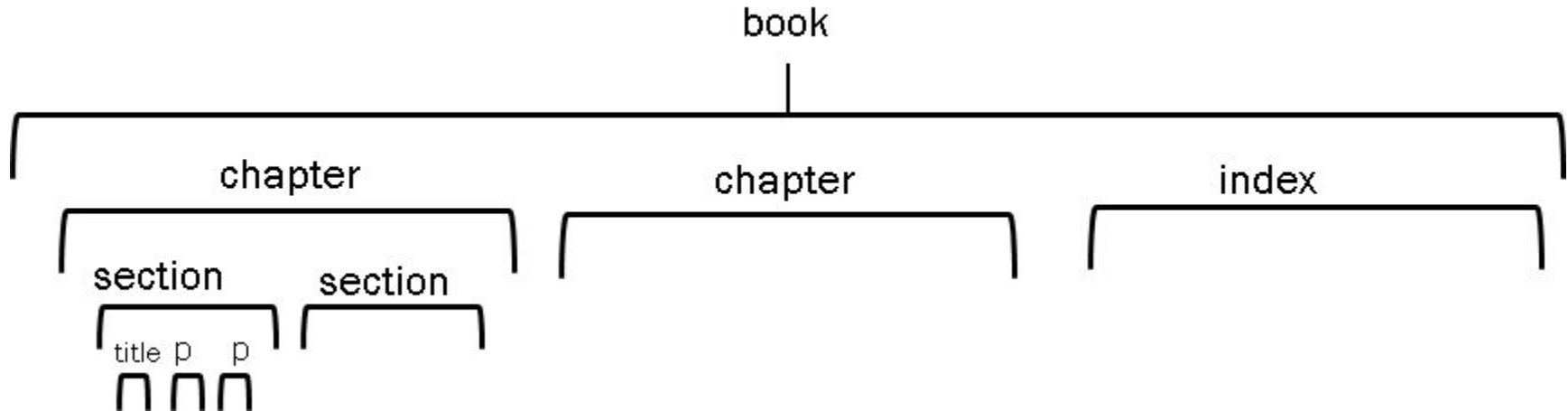
```
  <event date="2009-04-11T06:05:18" id="02" richter="3.6"/>
```

```
  <event date="2009-04-11T06:07:04" id="03" richter="5.0"/>
```

```
</seismographReadings>
```

How XML is structured

- Each element is contained by its ancestor
- An element at any level is always contained by one on the level above, until there is one element containing everything, e.g.:



Rules that all XML documents must follow

- All open (non-empty) elements must close
- A single 'root' element contains the whole document
- Elements must not overlap
- Certain characters in the *content* must be 'escaped' (e.g. <, >, &) because they form part of basic XML syntax, so for instance:
 - < becomes **<**
 - > becomes **>**
 - & becomes **&**

Spot the error...

```
<product>  
  <name>Peanut Butter Cup</name>  
  <company>Ben & Jerry's</company>  
</product>
```



```
<product>  
  <name>Peanut Butter Cup</name>  
  <company>Ben & Jerry's</company>  
</product>
```



Spot the error...

<title>

<roleName><persName>King</roleName> Lear</persName>

</title>



<title>

<persName><roleName>King</roleName> Lear</persName>

</title>



A (very simple) **well-formed** XML document

<example>

<statement>

We are holding a <event type="training">workshop</event>
in the <location>seminar room</location>.

</statement>

</example>

XML may **validate** to a schema

- A schema defines menu of elements, attributes, and values
- Defines what elements and content *may* appear in any given context
- Defines what elements and content *must* appear in any given context
- A schema enables an XML editor to:
 - check ("validate") your XML file
 - offer you a contextual menu of elements, attributes, and values available at that point in the document

Validation is *not* the same as well-formedness

- An XML document *must* be well-formed to meet the rules of XML
- It does *not* have to validate to a schema unless you want it to

What is an **XML editor**?

- An application specifically designed to work with XML files
- Comes programmed with full knowledge of XML rules
- Formats to clearly display the overall structure of the file
- Checks **well-formedness**
- Checks **validity** against a schema