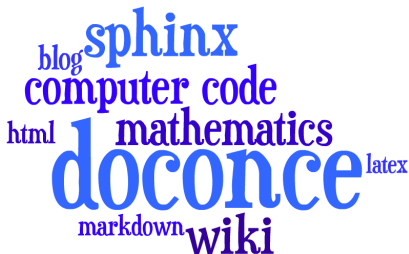


# Scientific Writing and Publishing Anno 2013

Hans Petter Langtangen

Aug 30, 2014



A word cloud of technical terms in blue, arranged in a roughly circular shape. The words are: sphinx, blog, computer, code, mathematics, latex, doconce, wiki, markdown, and html. The word 'doconce' is the largest and most central, with 'sphinx' above it and 'wiki' below it. 'computer' and 'code' are to the left of 'doconce', and 'mathematics' is to the right. 'blog' is above 'computer', 'html' is to the left of 'doconce', and 'markdown' is below 'doconce'.

# Scientific writing = $\text{\LaTeX}$

- Pre 1980: Handwriting + publisher (paper or book)
- Post 1985: scientists write  $\text{\LaTeX}$
- Post 1995: publish  $\text{\LaTeX}$  on the web and in journals and books

```
\providecommand{\shadedskip}{}
\definecolor{shadecolor}{rgb}{0.87843, 0.95686, 1.0}
\renewenvironment{shadedskip}{
\def\FrameCommand{\colorbox{shadecolor}}\FrameRule0.6pt
\MakeFramed {\FrameRestore}\vskip3mm}{\vskip0mm\endMakeFramed}
\providecommand{\shadedquoteBlue}{}
\renewenvironment{shadedquoteBlue}[1] [] {
\bggroup\rmfamily\fbboxsep=0mm\relax
\begin{shadedskip}
\list{}{\parsep=-2mm\parskip=0mm\topsep=0pt\leftmargin=2mm
\rightmargin=2\leftmargin\leftmargin=4pt\relax}
\relax}{\endlist\end{shadedskip}\egroup}\begin{shadedquoteBlue}
\fontsize{9pt}{9pt}
\begin{Verbatim}
print 'Hello, World!'
\end{Verbatim}
```

Big late 1990s question:

Will MS Word replace  $\text{\LaTeX}$ ? It never did!

# Scientific publishing needs to address new media



The book will probably survive



UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS  
ÉCOLE DOCTORALE STIC  
SCIENCES ET TECHNOLOGIES DE L'INFORMATION  
ET DE LA COMMUNICATION

## THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis

Mention : INFORMATIQUE

Présentée et soutenue par

Olivier COMMOWICK

## Création et utilisation d'atlas anatomiques numériques pour la radiothérapie

Thèse dirigée par Grégoire MALANDAIN

préparée à l'INRIA Sophia Antipolis, Projet ASCLEPIOS

Journal paper template • April 2012 • Vol. XXI, No. 1

## Long Titles Look More Impressive Than Short Ones

JONATHAN S. DOE\*

University of Technology, Delft  
frits@howtoTeX.com

### Abstract

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur magna lorem, tempor sed facilisis vel, porta et turpis. Sed et felis a massa dictum posuere. Aliquam hendrerit rhoncus ipsum sit amet placerat. Duis fringilla est eu arcu mollis faucibus non sit amet eros. Vestibulum risus nibh, dapibus vitae lacrima eget, fringilla quis nisi. Proin consequat nibh sit amet mauris suscipit tincidunt. Sed rutrum, purus nec aliquam faucibus, quam libero venenatis nisi, ut tempor mi sapien vel diam. Pellentesque sagittis elit non risus malesuada accumsan. Morbi consequat urna et lacus hendrerit sodales. Proin et urna neque, ut dapibus urna. Curabitur conenatis molestie convallis. Vestibulum blandit vulputate risus, quis sodales sapien porttitor non.*

**L**orem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur magna lorem, tempor sed facilisis vel, porta et turpis. Sed et felis a massa dictum posuere. Aliquam hendrerit rhoncus ipsum sit amet placerat. Duis fringilla est eu arcu mollis faucibus non sit amet eros. Vestibulum risus nibh, dapibus vitae lacrima eget, fringilla quis nisi. Proin consequat nibh sit amet mauris suscipit tincidunt. Sed rutrum, purus nec aliquam faucibus, quam libero venenatis nisi, ut tempor mi sapien vel diam. Pellentesque sagittis elit non risus malesuada accumsan. Morbi consequat urna et lacus hendrerit sodales. Proin et urna neque, ut dapibus urna. Curabitur venenatis molestie convallis. Vestibulum blandit vulputate risus, quis sodales sapien porttitor non.

Suspendisse id urna vel risus venenatis ultrices ut vel odio. Donec aliquet est a magna tincidunt ut rutrum lacus cursus. Praesent ultrices aliquam erat quis scelerisque. Vestibulum interdum interdum augue, at placerat turpis tempus nec. Vestibulum feugiat, tellus ultrices tempor fermentum, ipsum dolor vestibulum eros, sed vulputate felis eros eget ipsum. Fusce ultrices dapibus turpis non

\*Template by howtoTeX.com

pretium. Suspendisse potenti. Integer porttitor, lorem ac mattis fermentum, metus neque scelerisque sapien, vel lobortis orci erat at sapien. Mauris convallis nisi feugiat velit porttitor mollis. Nunc cursus est cursus erat malesuada sit amet cursus magna malesuada. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed eget dolor mauris. Aenean lobortis nunc vel velit lobortis quis tincidunt libero porta. Nunc hendrerit aliquet porttitor.

### I. SECTION TITLE EXAMPLE

Maecenas sed ultrices felis. Sed imperditi dictum arcu a egestas.

- Donec dolor arcu, rutrum id molestie in, viverra sed diam.
- Curabitur feugiat,
- turpis sed auctor facilisis,
- arcu eros accumsan lorem, at posuere mi diam sit amet tortor.
- Fusce fermentum, mi sit amet euismod rutrum,
- sem lorem molestie diam, iaculis aliquet sapien tortor non nisi.
- Pellentesque bibendum pretium aliquet.

# Scope of this presentation

- Focus: documents with **much** *math* and *computer code*
- Key question: What tools should I use for scientific writing?

The default answer is  $\text{\LaTeX}$ , but there are many recent popular alternative tools: HTML w/MathJax, Sphinx, Markdown, MediaWiki, IPython notebook.

$\text{\LaTeX}$



 SPHINX  
Python Documentation Generator

IP[y]: IPython  
Interactive Computing

# Does your scientific writing today need to address new media (in the future)?

- BW paper
- Color paper
- Slides
- Web w/design
- Wiki
- Blog
- Notebook
- ...



# Can we factor pieces from a heterogeneous world to one coherent piece in the future?

When I write some scientific material,

- a  $\text{\LaTeX}$  document,
- a blog post (HTML),
- some web pages (HTML),
- a Sphinx document,
- an IPython notebook,
- some Markdown files,

and later want to collect the pieces into a larger document, maybe some book, or one big web document, or a set of slides, is that at all feasible?

Probably not, but I have a solution :-)



# Can we factor pieces from a heterogeneous world to one coherent piece in the future?

When I write some scientific material,

- a  $\text{\LaTeX}$  document,
- a blog post (HTML),
- some web pages (HTML),
- a Sphinx document,
- an IPython notebook,
- some Markdown files,

and later want to collect the pieces into a larger document, maybe some book, or one big web document, or a set of slides, is that at all feasible?

Probably not, but I have a solution :-)

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support



# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support ([cf. Wikipedia/MediaWiki](#))

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support ([cf. Wikipedia/Wikibooks](#))

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support (cf. [Wikipedia/Wikipedia](#))

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support (cf. [Wikipedia/Wikipedia](#))

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support (cf. [Wikipedia/Wikipedia](#))

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support (cf. Wikipedia/Wikibooks)

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support (cf. Wikipedia/Wikibooks)

# Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-intensive documents
- **pdfLaTeX, XeLaTeX, LuaLaTeX**: takes over (figures in png, pdf) - use these!
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full"  $\text{\LaTeX}$  math, but much tagging
- **Sphinx**: somewhat limited  $\text{\LaTeX}$  math support, but great support for web design, and less tagged than HTML
- **reStructuredText**: similar to Sphinx, but no math support, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited  $\text{\LaTeX}$  math support, but minor tagging, transforms to lots of formats ( $\text{\LaTeX}$ , HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization, but requires all code snippets to sync together
- **MediaWiki**: quite good  $\text{\LaTeX}$  math support (cf. Wikipedia/Wikibooks)



# $\text{\LaTeX}$ is very rich; other tools support much less

- $\text{\LaTeX}$  inline math: works with all ( $\text{\LaTeX}$ , MathJax, Sphinx, Markdown, MediaWiki)
- $\text{\LaTeX}$  equation math:
  - **LaTeX**: equation\*, equation, align\*, align + eqnarray, split, alignat, ... (numerous!)
  - **MathJax**: equation\*, equation, align\*, align
  - **MediaWiki**: equation\*, equation, align\*, align
  - **Sphinx**: equation\*, equation, align\*
  - **Markdown**: equation\*, equation, eqnarray\*, align\* (but no labels)

# $\text{\LaTeX}$ is very rich; other tools support much less

- Figures: all
- Subfigures:  $\text{\LaTeX}$  (subfigure)
- Movies:  $\text{\LaTeX}$ , raw HTML
- Floating computer code:  $\text{\LaTeX}$ ; fixed computer code: all
- Interactive programs: Sphinx, IPython notebook, raw HTML
- Floating tables:  $\text{\LaTeX}$ ; fixed tables: all
- Algorithms:  $\text{\LaTeX}$
- Margin notes:  $\text{\LaTeX}$ , HTML with tailored css code
- Page references:  $\text{\LaTeX}$
- Footnotes:  $\text{\LaTeX}$ , Sphinx, reStructuredText, MediaWiki
- Bibliography:  $\text{\LaTeX}$ , Sphinx, reStructuredText, MediaWiki
- Hyperlinks: all (but not on paper!)

Conclusion: Highly non-trivial to translate a  $\text{\LaTeX}$  document into something based on HTML and vice versa.

# $\text{\LaTeX}$ is very rich; other tools support much less

- Figures: all
- Subfigures:  $\text{\LaTeX}$  (subfigure)
- Movies:  $\text{\LaTeX}$ , raw HTML
- Floating computer code:  $\text{\LaTeX}$ ; fixed computer code: all
- Interactive programs: Sphinx, IPython notebook, raw HTML
- Floating tables:  $\text{\LaTeX}$ ; fixed tables: all
- Algorithms:  $\text{\LaTeX}$
- Margin notes:  $\text{\LaTeX}$ , HTML with tailored css code
- Page references:  $\text{\LaTeX}$
- Footnotes:  $\text{\LaTeX}$ , Sphinx, reStructuredText, MediaWiki
- Bibliography:  $\text{\LaTeX}$ , Sphinx, reStructuredText, MediaWiki
- Hyperlinks: all (but not on paper!)

Conclusion: Highly non-trivial to translate a  $\text{\LaTeX}$  document into something based on HTML and vice versa.

# $\text{\LaTeX}$ is very rich; other tools support much less

- Figures: all
- Subfigures:  $\text{\LaTeX}$  (subfigure)
- Movies:  $\text{\LaTeX}$ , raw HTML
- Floating computer code:  $\text{\LaTeX}$ ; fixed computer code: all
- Interactive programs: Sphinx, IPython notebook, raw HTML
- Floating tables:  $\text{\LaTeX}$ ; fixed tables: all
- Algorithms:  $\text{\LaTeX}$
- Margin notes:  $\text{\LaTeX}$ , HTML with tailored css code
- Page references:  $\text{\LaTeX}$
- Footnotes:  $\text{\LaTeX}$ , Sphinx, reStructuredText, MediaWiki
- Bibliography:  $\text{\LaTeX}$ , Sphinx, reStructuredText, MediaWiki
- Hyperlinks: all (but not on paper!)

Conclusion: Highly non-trivial to translate a  $\text{\LaTeX}$  document into something based on HTML and vice versa.

# Typesetting concerns I

- Sphinx refers to figures by the caption (has to be short!) and strips away any math notation (avoid that!).
- Sphinx refers to sections by the title, but removes math in the reference, so avoid math in headlines.
- Tables cannot be referred to by numbers and have to appear at fixed positions in the text.
- Computer code has to appear at fixed positions in the text.
- Algorithms must be written up using basic elements like lists or paragraphs with headings.
- Recipes are often typeset as enumerated lists. For recipes with code or math blocks: drop the list (gives problems in some formats) and use paragraph (or subsubsection) headings with "Step 1.", "Step 2.", etc.

# Typesetting concerns I

- Sphinx refers to figures by the caption (has to be short!) and strips away any math notation (avoid that!).
- Sphinx refers to sections by the title, but removes math in the reference, so avoid math in headlines.
- Tables cannot be referred to by numbers and have to appear at fixed positions in the text.
- Computer code has to appear at fixed positions in the text.
- Algorithms must be written up using basic elements like lists or paragraphs with headings.
- Recipes are often typeset as enumerated lists. For recipes with code or math blocks: drop the list (gives problems in some formats) and use paragraph (or subsubsection) headings with "Step 1.", "Step 2.", etc.

# Typesetting concerns I

- Sphinx refers to figures by the caption (has to be short!) and strips away any math notation (avoid that!).
- Sphinx refers to sections by the title, but removes math in the reference, so avoid math in headlines.
- Tables cannot be referred to by numbers and have to appear at fixed positions in the text.
- Computer code has to appear at fixed positions in the text.
- Algorithms must be written up using basic elements like lists or paragraphs with headings.
- Recipes are often typeset as enumerated lists. For recipes with code or math blocks: drop the list (gives problems in some formats) and use paragraph (or subsubsection) headings with "Step 1.", "Step 2.", etc.

# Typesetting concerns I

- Sphinx refers to figures by the caption (has to be short!) and strips away any math notation (avoid that!).
- Sphinx refers to sections by the title, but removes math in the reference, so avoid math in headlines.
- Tables cannot be referred to by numbers and have to appear at fixed positions in the text.
- Computer code has to appear at fixed positions in the text.
- Algorithms must be written up using basic elements like lists or paragraphs with headings.
- Recipes are often typeset as enumerated lists. For recipes with code or math blocks: drop the list (gives problems in some formats) and use paragraph (or subsubsection) headings with "Step 1.", "Step 2.", etc.



# Typesetting concerns I

- Sphinx refers to figures by the caption (has to be short!) and strips away any math notation (avoid that!).
- Sphinx refers to sections by the title, but removes math in the reference, so avoid math in headlines.
- Tables cannot be referred to by numbers and have to appear at fixed positions in the text.
- Computer code has to appear at fixed positions in the text.
- Algorithms must be written up using basic elements like lists or paragraphs with headings.
- Recipes are often typeset as enumerated lists. For recipes with code or math blocks: drop the list (gives problems in some formats) and use paragraph (or subsubsection) headings with "Step 1.", "Step 2.", etc.

# Typesetting concerns I

- Sphinx refers to figures by the caption (has to be short!) and strips away any math notation (avoid that!).
- Sphinx refers to sections by the title, but removes math in the reference, so avoid math in headlines.
- Tables cannot be referred to by numbers and have to appear at fixed positions in the text.
- Computer code has to appear at fixed positions in the text.
- Algorithms must be written up using basic elements like lists or paragraphs with headings.
- Recipes are often typeset as enumerated lists. For recipes with code or math blocks: drop the list (gives problems in some formats) and use paragraph (or subsubsection) headings with "Step 1.", "Step 2.", etc.

- Sphinx refers to figures by the caption (has to be short!) and strips away any math notation (avoid that!).
- Sphinx refers to sections by the title, but removes math in the reference, so avoid math in headlines.
- Tables cannot be referred to by numbers and have to appear at fixed positions in the text.
- Computer code has to appear at fixed positions in the text.
- Algorithms must be written up using basic elements like lists or paragraphs with headings.
- Recipes are often typeset as enumerated lists. For recipes with code or math blocks: drop the list (gives problems in some formats) and use paragraph (or subsubsection) headings with "Step 1.", "Step 2.", etc.

## Typesetting concerns II

- Footnotes must appear as part of the running text (e.g., sentences surrounded by parenthesis), since only a few formats support footnotes.
- Sphinx does not handle code blocks where the first line is indented.
- Multiple plots in the same figure: mount the plots to one image file and include this (montage for png, gif, jpeg; pdftk, pdfnup, and pdfcrop for PDF).
- If you need several equations *numbered* in an `align` environment, recall that Sphinx, Markdown, and MediaWiki cannot handle this, although they have  $\LaTeX$  math support.
- Markdown tolerates labels in equations but cannot refer to them.

## Typesetting concerns II

- Footnotes must appear as part of the running text (e.g., sentences surrounded by parenthesis), since only a few formats support footnotes.
- Sphinx does not handle code blocks where the first line is indented.
- Multiple plots in the same figure: mount the plots to one image file and include this (montage for png, gif, jpeg; pdftk, pdfnup, and pdfcrop for PDF).
- If you need several equations *numbered* in an `align` environment, recall that Sphinx, Markdown, and MediaWiki cannot handle this, although they have  $\text{\LaTeX}$  math support.
- Markdown tolerates labels in equations but cannot refer to them.

## Typesetting concerns II

- Footnotes must appear as part of the running text (e.g., sentences surrounded by parenthesis), since only a few formats support footnotes.
- Sphinx does not handle code blocks where the first line is indented.
- Multiple plots in the same figure: mount the plots to one image file and include this (montage for png, gif, jpeg; pdftk, pdfnup, and pdfcrop for PDF).
- If you need several equations *numbered* in an `align` environment, recall that Sphinx, Markdown, and MediaWiki cannot handle this, although they have  $\text{\LaTeX}$  math support.
- Markdown tolerates labels in equations but cannot refer to them.

## Typesetting concerns II

- Footnotes must appear as part of the running text (e.g., sentences surrounded by parenthesis), since only a few formats support footnotes.
- Sphinx does not handle code blocks where the first line is indented.
- Multiple plots in the same figure: mount the plots to one image file and include this (`montage` for png, gif, jpeg; `pdftk`, `pdfnup`, and `pdfcrop` for PDF).
- If you need several equations *numbered* in an `align` environment, recall that Sphinx, Markdown, and MediaWiki cannot handle this, although they have  $\text{\LaTeX}$  math support.
- Markdown tolerates labels in equations but cannot refer to them.

## Typesetting concerns II

- Footnotes must appear as part of the running text (e.g., sentences surrounded by parenthesis), since only a few formats support footnotes.
- Sphinx does not handle code blocks where the first line is indented.
- Multiple plots in the same figure: mount the plots to one image file and include this (`montage` for png, gif, jpeg; `pdftk`, `pdfnup`, and `pdfcrop` for PDF).
- If you need several equations *numbered* in an `align` environment, recall that Sphinx, Markdown, and MediaWiki cannot handle this, although they have  $\text{\LaTeX}$  math support.
- Markdown tolerates labels in equations but cannot refer to them.



## Typesetting concerns II

- Footnotes must appear as part of the running text (e.g., sentences surrounded by parenthesis), since only a few formats support footnotes.
- Sphinx does not handle code blocks where the first line is indented.
- Multiple plots in the same figure: mount the plots to one image file and include this (`montage` for png, gif, jpeg; `pdftk`, `pdfnup`, and `pdfcrop` for PDF).
- If you need several equations *numbered* in an `align` environment, recall that Sphinx, Markdown, and MediaWiki cannot handle this, although they have  $\text{\LaTeX}$  math support.
- Markdown tolerates labels in equations but cannot refer to them.

- Index words can appear anywhere in  $\text{\LaTeX}$ , but should be outside paragraphs in other tools.
- References to tables, program code and algorithms can only be made in  $\text{\LaTeX}$ .
- Figures are floating in  $\text{\LaTeX}$ , but fixed in other tools, so place figures exactly where they are needed the first time.
- Curve plots with color lines do not work well in black-and-white printing. Make sure plots makes sense in color and BW (e.g., by using colors *and* markers).

- Index words can appear anywhere in  $\text{\LaTeX}$ , but should be outside paragraphs in other tools.
- References to tables, program code and algorithms can only be made in  $\text{\LaTeX}$ .
- Figures are floating in  $\text{\LaTeX}$ , but fixed in other tools, so place figures exactly where they are needed the first time.
- Curve plots with color lines do not work well in black-and-white printing. Make sure plots makes sense in color and BW (e.g., by using colors *and* markers).

- Index words can appear anywhere in  $\text{\LaTeX}$ , but should be outside paragraphs in other tools.
- References to tables, program code and algorithms can only be made in  $\text{\LaTeX}$ .
- Figures are floating in  $\text{\LaTeX}$ , but fixed in other tools, so place figures exactly where they are needed the first time.
- Curve plots with color lines do not work well in black-and-white printing. Make sure plots makes sense in color and BW (e.g., by using colors *and* markers).

- Index words can appear anywhere in  $\text{\LaTeX}$ , but should be outside paragraphs in other tools.
- References to tables, program code and algorithms can only be made in  $\text{\LaTeX}$ .
- Figures are floating in  $\text{\LaTeX}$ , but fixed in other tools, so place figures exactly where they are needed the first time.
- Curve plots with color lines do not work well in black-and-white printing. Make sure plots makes sense in color and BW (e.g., by using colors *and* markers).

- Index words can appear anywhere in  $\text{\LaTeX}$ , but should be outside paragraphs in other tools.
- References to tables, program code and algorithms can only be made in  $\text{\LaTeX}$ .
- Figures are floating in  $\text{\LaTeX}$ , but fixed in other tools, so place figures exactly where they are needed the first time.
- Curve plots with color lines do not work well in black-and-white printing. Make sure plots makes sense in color and BW (e.g., by using colors *and* markers).

## Solution I: Use a format that translates to many

- Sphinx can do nice HTML,  $\text{\LaTeX}$ , epub, (almost) plain text, man pages, Gnome devhelp files, Qt help files, texinfo, JSON
- Markdown can do  $\text{\LaTeX}$ , HTML, MS Word, OpenOffice, XML, reStructuredText, epub, DocBook, ... but not Sphinx
- IPython notebook: can do  $\text{\LaTeX}$ , reStructuredText, HTML, PDF, Python script
- Sphinx and Markdown has some limited math support

## Solution II: Use DocOnce

DocOnce offers minimalistic typing, great flexibility wrt format, especially for scientific writing with *much math and code*.

- Can generate  $\text{\LaTeX}$ , HTML, Sphinx, Markdown, MediaWiki, Google wiki, Creole wiki, reST, plain text
- Made for large science books *and* small notes
- Targets paper and screen
- Many special features (code snippets from files, embedded movies, admonitions, modern  $\text{\LaTeX}$  layouts, ...)
- Very effective for generating slides from ordinary text
- Applies Mako: DocOnce text is a program (!)
- Much like Markdown, less tagged than  $\text{\LaTeX}$ , HTML, Sphinx



[http://hplgit.github.com/teamods/writing\\_reports/](http://hplgit.github.com/teamods/writing_reports/)

- $\text{\LaTeX}$ -based PDF for screen, for printing, for phone
- Plain HTML or with a template or another template or solarized
- Sphinx: agni, pyramid, classy, fenics, redcloud
- HTML for Google or Wordpress for blog posts
- MediaWiki (Wikipedia, Wikibooks, etc)
- DocOnce source code and tutorial

- Based on text transformations (reg.exp.) so valid syntax may occasionally give problems

## DocOnce divorce.

At any time one can divorce from DocOnce and marry one of the output formats, such as  $\text{\LaTeX}$  or Sphinx. The generated code is clean.

## DocOnce experience: code generation is a great thing

Regardless of what format you write in, introduce a step where you can generate (parts of) the syntax.

- Use a preprocessor a la Mako
- Write your own read-and-generate code
- or both (like DocOnce)

Advantages:

- Less writing
- Repository of syntax for nice constructions
- Implements structure/rules across documents
- Easier to change layout/structure

# Example: generate reveal.js or deck.js slides from HTML

- Write the content of each slide in plain HTML(5)
- Use e.g. `#slide` as delimiter between slides
- Read file, splitting wrt `#slide` yields a list of slides (HTML code)
- For a specific format (reveal.js, deck.js, csss, ...):
  - write header
  - for slide in slides:
    - embed slide in correct HTML code
  - write footer

```
<h2>Scope of this presentation</h2>
```

```
<ul>
```

```
  <li>Focus: documents with much <em>math</em> and  
    <em>computer code</em>
```

```
  <li>Key question: What tools should I use for scientific writing?
```

```
</ul>
```

```
<p><div class="alert">
```

```
The default answer is LaTeX.
```

```
</div>
```

# Title, authors, date, toc

```
TITLE: Some Title
AUTHOR: name1 at institution1, with more info & institution2
AUTHOR: name2 email:name2@web.com at institution
DATE: today

# A table of contents is optional:
TOC: on
```

## Notice.

Title and authors must have all information *on a single line!*

# Abstract

```
--Abstract.--  
Here goes the abstract...
```

Or:

```
--Summary.--  
Here goes the summary...
```

# Section headings

Headings are surrounded by = signs:

```
===== This is an H1/chapter heading =====
```

```
===== This is an H2/section heading =====
```

```
===== This is an H3/subsection heading =====
```

```
==== This is an H4/paragraph heading ====
```

```
__This is a paragraph heading.__
```

Result:

# **This is an H1/chapter heading**

## **This is an H2/section heading**

### **This is an H3/subsection heading**

#### **This is an H4/paragraph heading. This is a paragraph heading.**

# Markup and lists

```
* Bullet list items start with '*'
  and may span several lines
* *Emphasized words* are possible
* _Boldface words_ are also possible
* color{red}{colored words} too
* 'inline verbatim code' is featured
  o and sublists with enumerated items starting with 'o'
  o items are just indented as you would do in email
```

This gets rendered as

- Bullet lists start with \* and may span several lines
- *Emphasized words* are possible
- **Boldface words** are also possible
- colored words too
- inline verbatim code is featured
  - ❶ and sublists with enumerated items starting with o
  - ❷ items are just indented as you would do in email



# Labels, references, index items

```
# Insert index items in the source  
idx{key word1} idx{key word2}
```

```
# Label  
===== Some section =====  
label{this:section}
```

```
# Make reference  
As we saw in Section ref{this:section}, references, index  
items and labels follow a syntax similar to LaTeX  
but without backslashes.
```

```
# Make reference to equations  
See (ref{eq1})-(ref{myeq}).
```

```
# Make hyperlink  
"some link text": "https://github.com/hplgit/doconce"
```

```
# Hyperlink with complete URL as link text  
URL: "https://github.com/hplgit/doconce"
```

## Important:

Figures with HTML and  $\text{\LaTeX}$  size info, and caption: *everything on one line*

FIGURE: [figdir/myfig, width=300 frac=1.2] My caption. label{fig1}

Movies are also supported:

MOVIE: [http://youtu.be/IDeGDFZSYo8, width=420 height=315]

and rendered as

<http://youtube.com/IDeGDFZSYo8>

Inline math as in  $\text{\LaTeX}$ :

...where  $a = \int_{\Omega} f dx$  is an integral.

gets rendered as ...where  $a = \int_{\Omega} f dx$  is an integral.

An equation environment is surrounded by `!bt` and `!et` tags, the rest is plain  $\text{\LaTeX}$ :

```
!bt
\begin{align}
\frac{\partial u}{\partial t} &= \nabla^2 u,
\label{a:eq}\
\nabla \cdot \mathbf{v} &= 0
\label{b:eq}
\end{align}
!et
```

which is rendered as

$$\frac{\partial u}{\partial t} = \nabla^2 u, \tag{1}$$

$$\nabla \cdot \mathbf{v} = 0 \tag{2}$$

# Math flexibility

Limit math environments to

```
\[ ... \]
```

```
\begin{equation*}  
\end{equation*}
```

```
\begin{equation}  
\end{equation}
```

```
\begin{align*}  
\end{align*}
```

```
\begin{align}  
\end{align}
```

DocOnce fix of shortcomings.

- Sphinx, Markdown, and MediaWiki cannot have `align` with labels
- MathJax (HTML, Sphinx, Markdown, Mediawiki, ...) cannot handle equation references across web pages

# Displaying code

Code is enclosed in `!bc` and `!ec` tags:

```
!bc pycod
def solver(I, a, T, dt, theta):
    """Solve  $u' = -a*u$ ,  $u(0)=I$ , for  $t$  in  $(0, T]$  with steps of  $dt$ ."""
    dt = float(dt); N = int(round(T/dt)); T = N*dt
    u = zeros(N+1); t = linspace(0, T, N+1)

    u[0] = I
    for n in range(0, N):
        u[n+1] = (1 - (1-theta)*a*dt)/(1 + theta*dt*a)*u[n]
    return u, t
!ec
```

This gets rendered as

```
def solver(I, a, T, dt, theta):
    """Solve  $u' = -a*u$ ,  $u(0)=I$ , for  $t$  in  $(0, T]$  with steps of  $dt$ ."""
    dt = float(dt); N = int(round(T/dt)); T = N*dt
    u = zeros(N+1); t = linspace(0, T, N+1)

    u[0] = I
    for n in range(0, N):
        u[n+1] = (1 - (1-theta)*a*dt)/(1 + theta*dt*a)*u[n]
    return u, t
```

# Copying code from source files

We recommend to copy as much code as possible directly from the source files:

```
@@@CODE path/to/file  
@@@CODE path/to/file    fromto: start-regex@end-regex
```

For example, copying a code snippet starting with `def solver(` and ending with (line not included) `def next(x, y,` is specified by start and end regular expressions:

```
@@@CODE src/dc_mod.py    fromto: def solver\(@def next\(x,\s*y,
```

# Typesetting of code is implied by the file extension

- `.py`: `pypro` if complete file, `pycod` if snippet
- `.pyopt`: visualized execution via the [Online Python Tutor](#)
- `.f`, `.f90`, `f.95`: `fpro` and `fcod`
- `.cpp`, `.cxx`: `cpppro` and `cppcod`
- `.c`: `cpro` and `ccod`
- `.*sh`: `shpro` and `shcod`
- `.m`: `mpro` and `mcod`
- `ptex2tex`: between 40+ code styles in  $\text{\LaTeX}$
- `pygments` is used for code in HTML (ca 10 styles)

# Demonstrating code execution; Online Python Tutor

With `!bc pyoptpro` or a file `*.pyopt`, the code applies the [Online Python Tutor](#) for displaying program flow and state of variables:

```
def solver(I, a, T, dt, theta):
    dt = float(dt)
    N = int(round(T/dt))
    T = N*dt
    u = [0.0]*(N+1)
    t = [i*dt for i in range(N+1)]

    u[0] = I
    for n in range(0, N):
        u[n+1] = (1 - (1-theta)*a*dt)/(1 + theta*dt*a)*u[n]
    return u, t

u, t = solver(I=1, a=1, T=3, dt=1., theta=0.5)
print u
```

(Visualize execution)



# Demonstrating code execution; Sage Cell Server

With `!bc pyscpro` or a file `*.pysc`, the code is typeset in a sage cell:

```
a = 2
b = 3
print 'a+b:', a + b

# In a sage cell we can also plot
from matplotlib.pyplot import *
from numpy import *
x = linspace(0, 4*pi, 101)
y = exp(-0.1*x)*cos(x)
plot(x, y)
xlabel('x'); ylabel('y')
show()
```

Warning.

Works only in Sphinx documents (but HTML support is possible).

# Demonstrating code execution; IPython notebook

Can take a [DocOnce source](#) and transform to an [IPython notebook](#) with [source](#)

# Tables

time	velocity	acceleration
0.0	1.4186	-5.01
2.0	1.376512	11.919
4.0	1.1E+1	14.717624

Gets rendered as

<i>time</i>	<i>velocity</i>	<i>acceleration</i>
<i>0.0</i>	<i>1.4186</i>	<i>-5.01</i>
<i>2.0</i>	<i>1.376512</i>	<i>11.919</i>
<i>4.0</i>	<i>1.1E+1</i>	<i>14.717624</i>

# Newcommands for math

- `newcommands*.tex` files contain newcommands
- Used directly in  $\text{\LaTeX}$
- Substitution made for many other formats

# Labels, citations, index, bibliography

Labels, citations, index, and bibliography follow the ideas of  $\text{\LaTeX}$ , but without backslashes:

```
===== My Section =====  
label{sec:mysec}
```

```
idx{key equation} idx{ $\u$  conservation}
```

We refer to Section `ref{sec:yoursec}` for background material on the `*key equation*`. Here we focus on the extension

```
!bt  
\begin{equation}  
\Ddt{\u} = \mycommand{v} label{mysec:eq:Dudt}  
\end{equation}  
!et  
Equation (ref{mysec:eq:Dudt}) is important, see  
cite{Larsen_et_al_2002,Johnson_Friedman_2010a}.  
Also, cite{Miller_2000} supports such a view.
```

Figure `ref{mysec:fig:myfig}` displays the features.

FIGURE: `[fig/myfile, width=600]` My figure. `label{mysec:fig:myfig}`

```
===== References =====
```

```
BIBFILE: papers.pub
```

# Exercises

DocOnce offers a special format for *exercises*, *problems*, *projects*, and *examples*:

```
===== Problem: Flip a Coin =====
label{demo:ex:1}
files=flip_coin.py, flip_coin.pdf
solutions=mysol.txt, mysol_flip_coin.py
keywords = random numbers; Monte Carlo simulation

!bsubex
Make a program that simulates flipping a coin  $N$  times.

!bhint
Use 'r = random.random()' and define head as 'r <= 0.5'.
!ehint
!esubex

!bsubex
Compute the probability of getting heads.

!bans
0.5.
!eans
!esubex
```

**a)** Make a program that simulates flipping a coin  $N$  times.

**Hint.** Use `r = random.random()` and define head as `r <= 0.5`.

**b)** Compute the probability of getting heads.

**Answer.** 0.5.

Filenames: `flip_coin.py`, `flip_coin.pdf`.

All *exercises*, *problems*, and *projects* in a document are parsed and available in a data structure (list of dicts) for further processing (e.g., making a book of problems).

```
[{'answer': '',
  'closing_remarks': '',
  'file': ['flip_coin.py', 'flip_coin.pdf'],
  'hints': [],
  'keywords': ['random numbers', 'Monte Carlo simulation'],
  'label': 'demo:ex:1',
  'solution_file': ['mysol.txt', 'mysol_flip_coin.py'],
  'subex': [{'answer': '',
              'file': None,
              'hints': ['Use 'r = random.random()' ...'],
              'solution': '',
              'text': 'Make a program that simulates ...'}],
  'title': 'Flip a Coin',
  'type': 'Problem'}
```



# Use of preprocessors

- Simple if-else tests a la the C/C++ preprocessor
- `FORMAT` variable can be used to test on format, e.g.,
  - if latex/pdflatex do one sort of code (raw `LATEX`)
  - if html, do another type of code (raw `HTML`)
- Easy to comment out large portions of text
- Easy to make different versions of the document
- The mako preprocessor is really powerful - gives a complete programming language inside the document!

# DocOnce admonitions

## Use with caution!

Such environments may light up the document, but can be disturbing too. Some admon styles have icons.

## Going deeper.

More details can be separated from the rest.

## Time for review!

Tasks:

- Maybe ask a question?
- Or two?

Conclusion:

- A special "block" admonition has less pronounced typesetting and can be used when no special icon is desired. Good for slides.

Very effective way to generate slides from running text:

- Take a copy of your DocOnce prose
- Strip off as much text as possible
- Emphasize key points in bullet items
- Focus on key equations, figures, movies, key code snippets
- Insert `!split` wherever you want a new slide to begin
- Insert `!bpop` and `!epop` around elements to pop up in sequence
- Use `7 =` or `5 =` in headings (H2 or H3)
- Supported slide types: Beamer, HTML, HTML5 (reveal.js, deck.js, csss, dzslides)

# Example on slide code

```
!split
===== Headline =====

* Key point 1
* Key point 2
* Key point 3: Although long
  bullet points are not recommended in general, we need
  it here for demonstration purposes to investigate
  what happens with the slide layout where there is
  so much text under one point
```

```
FIGURE: [fig/teacher1, width=100 frac=0.4]
```

Key equation:

```
!bt
\[ -\nabla^2 u = f \quad \hbox{in } \Omega \]
!et
```

And maybe a final comment?

```
!split
===== Next slide... =====
```

# Example on slide code

Last page gets rendered to

## Headline

- Key point 1
- Key point 2



# Grid layout of slide: MxN cells

Example with a bullet list to the left and a figure to the right (two cells: 00 and 01):

```
!split
===== Headline =====

!bslidecell 00
!bpop
* Key point 1
* Key point 2
* Key point 3
!epop

!bpop
!bt

$$[-\nabla^2 u = f \quad \text{in } \Omega]$$

!et
!epop

!eslidecell

!bslidecell 01
FIGURE: [fig/broken_pen_and_paper, width=400, frac=0.8]
!eslidecell

!split
===== Next slide... =====
```

Last page gets rendered to

## Headline

- Key point 1
- Key point 2
- Key point 3

$$-\nabla^2 u = f \quad \text{in } \Omega$$



Last page gets rendered to

## Headline

- Key point 1
- Key point 2
- Key point 3

$$-\nabla^2 u = f \quad \text{in } \Omega$$





Last page gets rendered to

## Headline

- Key point 1
- Key point 2
- Key point 3

$$-\nabla^2 u = f \quad \text{in } \Omega$$



# Classic slide types

- $\text{\LaTeX}$  Beamer
- Plain HTML w/various styles
  - separate slides w/navigation
  - one big slide

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - slides.js
  - slides
- **Problem:** each package has its own syntax (though similar)
- **Problem:** reveal and deck have numerous styles
- **Problem:** HTML5 slides need many style files
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- **Problem:** each package has its own syntax (though similar)
- **Problem:** reveal and deck have numerous styles
- **Problem:** HTML5 slides need many style files
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- Problem: each package has its own syntax (though similar)
- Problem: reveal and deck have numerous styles
- Problem: HTML5 slides need many style files
- Problem: original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- **Problem:** each package has its own syntax (though similar)
- **Problem:** reveal and deck have numerous styles
- **Problem:** HTML5 slides need many style files
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - CSSS
- **Problem:** each package has its own syntax (though similar)
  - *Solution:* slide code is autogenerated from Documen
- **Problem:** reveal and deck have numerous styles
- **Problem:** HTML5 slides need many style files
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- **Problem:** each package has its own syntax (though similar)
  - **Solution:** slide code is autogenerated from DocOnce
- **Problem:** reveal and deck have numerous styles
- **Problem:** HTML5 slides need many style files
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code



- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- **Problem:** each package has its own syntax (though similar)
  - **Solution:** slide code is autogenerated from DocOnce
- **Problem:** reveal and deck have numerous styles
  - **Solution:** way to automatically all styles for a talk
- **Problem:** HTML5 slides need many style files
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- **Problem:** each package has its own syntax (though similar)
  - **Solution:** slide code is autogenerated from DocOnce
- **Problem:** reveal and deck have numerous styles
  - **Solution:** easy to autogenerate all styles for a talk
- **Problem:** HTML5 slides need many style files
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

# HTML5 slide types

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- **Problem:** each package has its own syntax (though similar)
  - **Solution:** slide code is autogenerated from DocOnce
- **Problem:** reveal and deck have numerous styles
  - **Solution:** easy to **autogenerate all styles** for a talk
- **Problem:** HTML5 slides need many style files
  - **Solution:** **autogenerate all styles from DocOnce**
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

# HTML5 slide types

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- **Problem:** each package has its own syntax (though similar)
  - **Solution:** slide code is autogenerated from DocOnce
- **Problem:** reveal and deck have numerous styles
  - **Solution:** easy to [autogenerate all styles](#) for a talk
- **Problem:** HTML5 slides need many style files
  - **Solution:** autocopy all files to talk directory
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

# HTML5 slide types

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- **Problem:** each package has its own syntax (though similar)
  - **Solution:** slide code is autogenerated from DocOnce
- **Problem:** reveal and deck have numerous styles
  - **Solution:** easy to [autogenerate all styles](#) for a talk
- **Problem:** HTML5 slides need many style files
  - **Solution:** autocopy all files to talk directory
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code

# HTML5 slide types

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
- **Problem:** each package has its own syntax (though similar)
  - **Solution:** slide code is autogenerated from DocOnce
- **Problem:** reveal and deck have numerous styles
  - **Solution:** easy to [autogenerate all styles](#) for a talk
- **Problem:** HTML5 slides need many style files
  - **Solution:** autocopy all files to talk directory
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code
  - **Solution:** DocOnce contains adjusted css files

# HTML5 slide types

- Supported HTML5 packages:
  - `reveal.js`
  - `deck.js`
  - `dzslides`
  - `csss`
- **Problem:** each package has its own syntax (though similar)
  - **Solution:** slide code is autogenerated from DocOnce
- **Problem:** reveal and deck have numerous styles
  - **Solution:** easy to [autogenerate all styles](#) for a talk
- **Problem:** HTML5 slides need many style files
  - **Solution:** autocopy all files to talk directory
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code
  - **Solution:** DocOnce contains adjusted css files

# HTML5 slide types

- Supported HTML5 packages:
  - `reveal.js`
  - `deck.js`
  - `dzslides`
  - `csss`
- **Problem:** each package has its own syntax (though similar)
  - **Solution:** slide code is autogenerated from DocOnce
- **Problem:** reveal and deck have numerous styles
  - **Solution:** easy to [autogenerate all styles](#) for a talk
- **Problem:** HTML5 slides need many style files
  - **Solution:** autocopy all files to talk directory
- **Problem:** original versions of the styles have too large fonts, centering, and other features not so suitable for lectures with much math and code
  - **Solution:** DocOnce contains adjusted css files



Run in terminal window:

```
doonce format html dooncefile
```

```
# Solarized HTML style
```

```
doonce format html dooncefile --html_solarized
```

```
# Control pygments typesetting of code
```

```
doonce format html dooncefile --pygments_html_style=native
```

```
# Or use plain <pre> tag for code
```

```
doonce format html dooncefile --no_pygments_html
```

```
# Further making of slides
```

```
doonce slides_html dooncefile reveal --html_slide_theme=darkgray
```

# Output for blog posts

Two formats of blog posts are supported:

- Google's [blogspot.com](http://blogspot.com): just paste the raw HTML (full support of math and code)
- [Wordpress](#): despite limited math, DocOnce manipulates the math such that even `equation` and `align` work in Wordpress :-)

For wordpress, add `--wordpress`:

```
doconce format html doconcefile --wordpress
```

and paste the code into the text area.

```
doconce format pdflatex doconcefile

# Result: doconcefile.p.tex (ptex2tex file)
# Run either
ptex2tex doconcefile
# or
doconce ptex2tex doconcefile -DHELVETICA envir=minted

pdflatex doconcefile
bibtex doconcefile
pdflatex doconcefile

# More control of how code is typeset
doconce format pdflatex doconcefile --minted_latex_style=trac
doconce ptex2tex doconcefile envir=minted

doconce format pdflatex doconcefile
doconce ptex2tex doconcefile envir=ans:nt
```

```
doconce format sphinx doconcefile  
  
# Autocreate sphinx directory  
doconce sphinx_dir theme=pyramid doconcefile  
  
# Copy files and build HTML document  
python automake-sphinx.py  
  
google-chrome sphinx-rootdir/_build/html/index.html
```

Much easier than running the Sphinx tools manually!

Only MediaWiki supports math.

```
doconce format mwiki doconcefile
```

Recommended site:

- [ShoutWiki](#) for standard wikis

Publishing of "official" documents:

- [Wikibooks](#) (can test code in the [sandbox](#))
- Wikipedia

# DocOnce to other formats

```
doconce format pandoc doconcefile # (Pandoc extended) Markdown
doconce format gwiki doconcefile # Googlecode wiki
doconce format cwiki doconcefile # Creole wiki (Bitbucket)
doconce format rst doconcefile # reStructuredText
doconce format plain doconcefile # plain, untagged text for email
```

# Installation

- Ubuntu: `sudo apt-get install python-doconce` (old!)
- Source at [GitHub](#) (recommended!)
  - `hg clone + sudo python setyp.py install`
- Many [dependencies](#)...
  - Must have preprocess and mako
  - Need latex, sphinx, pandoc, etc. (see the [Installation description](#))
  - Easy for slides: only preprocess is needed :-)

# Writing tips for L<sup>A</sup>T<sub>E</sub>X writers who want to convert to DocOnce

- `doconce latex2doconce` helps the translation
- Use `\[ \]`, `equation`, `equation*`, `align`, `align*` and nothing more for equations
- Figures: avoid subfigures (combine image files instead), use `\includegraphics`, have captions after graphics, use short figure captions, position exactly where needed
- Tables: have them inline (not floating), with no caption
- Computer codes: have them inline (not floating)
- Avoid footnotes, `pageref`
- Do not use *algorithm* environments, use simple list formatting instead
- Avoid math in section headings
- Use `pdflatex` or `xetex`
- Use BIB<sub>T</sub>E<sub>X</sub> (can easily be converted to `publish` used by DocOnce)
- Use `\href` for links (and insert links frequently)
- Use the `bm` package for boldface **u**



# DocOnce writing tips

## Figures and movies:

- Prepare figures in the right format: EPS for latex, PDF for pdflatex, PNG, GIF or JPEG for HTML formats (`html`, and HTML output from `sphinx`, `rst`, `pandoc`). One can omit the figure file extension and `doconce` will pick the most appropriate file for the given output format.
- Let plotting programs produce both PDF/EPS and PNG files. (Recall that PDF and EPS are vector graphics formats that can scale to any size with much higher quality than PNG or other bitmap formats.)
- Use `doconce combine_images` to combine several images into one.
- Store all figures in a directory (tree) with name `fig` or `fig-X`, where `X` is some short logical name for the current document.
- Store all movies in a directory (tree) with name `mov` or `mov-X`.
- Favor the movie formats MP4, WebM, and Ogg (best suited for modern browsers).

## DocOnce writing tips

- `\bm{u}` gives nicer boldface typesetting of math symbols than the alternatives `\boldsymbol{u}` and `\pmb{u}`.
- For HTML-based formats using MathJax, `\bm{u}` is not supported and therefore automatically replaced by `\boldsymbol{u}` by DocOnce.
- Use `\\textcolor{blue}{formula}` in math expressions to color a part.
- Not all  $\LaTeX$  math is supported by MathJax. Some legal  $\LaTeX$  math might give MathJax problems - then one has to rewrite the expression to find a syntax that works both with  $\LaTeX$  and MathJax.
- Use `doonce spellcheck *.do.txt` to automatically spellcheck files.
- Avoid page references and footnotes.

# Writing tips for sphinx and other formats

For output formats different from `latex`, `pdflatex`, and `html`:

- Use labels only right after section headings and in equations.
- Be careful with labels in `align` math environments: `pandoc` and `mwiki` cannot refer to them.
- `sphinx` output requires
  - no math in section headings or figure captions (gets removed in references).
  - running text to start in column 1.
  - progressive section headings: after chapter (9 =) comes section (7 =), then subsection (5 =), then paragraph (3 =). Do not make jumps in this progression.
  - index entries (`\index{keyword}`) before the paragraph where they are introduced and place them *before* subsubsection headings (`=== ... ===`) and after subsection and section headings.
  - a line of text and no comment or math before code or list.