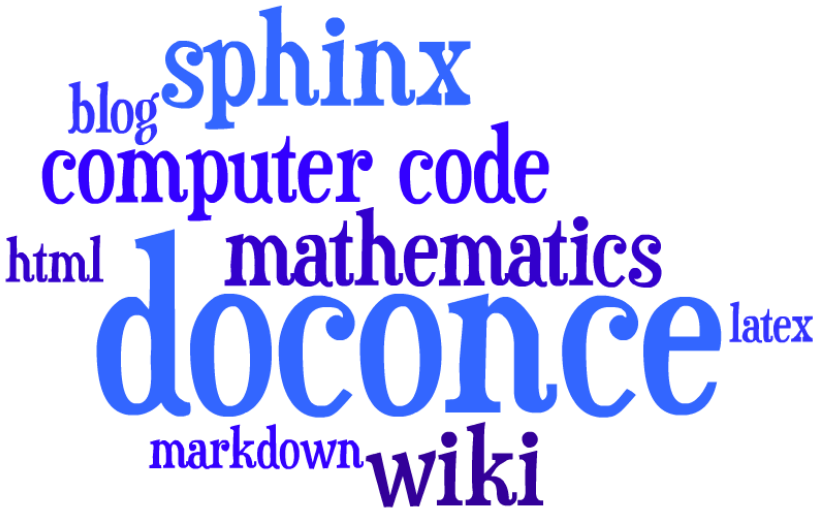
Scientific Writing Anno 2013: The Doconce Way

Hans Petter Langtangen

May 4, 2013



0.1 Scientific writing for the future needs to address many new media

0.2 The book will probably survive

0.3 The classical report will survive

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC
SCIENCES ET TECHNOLOGIES DE L'INFORMATION
ET DE LA COMMUNICATION

T H È S E

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 laoreet eget, fringilla quis nisl. 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 at urna neque, ut dapibus urna. Curabitur venenatis molestie convallis. Vestibulum blandit vulputate risus, quis sodales sapien porttitor non.

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 laoreet eget, fringilla quis nisl. 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 at 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 at magna tincidunt ut rutrum lacus cursus. Praesent ultricies 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 ultricies dapibus turpis non 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.

*Template by howtoTeX.com

I. Section title example

Maecenas sed ultricies felis. Sed imperdiet dictum arcu a egestas.

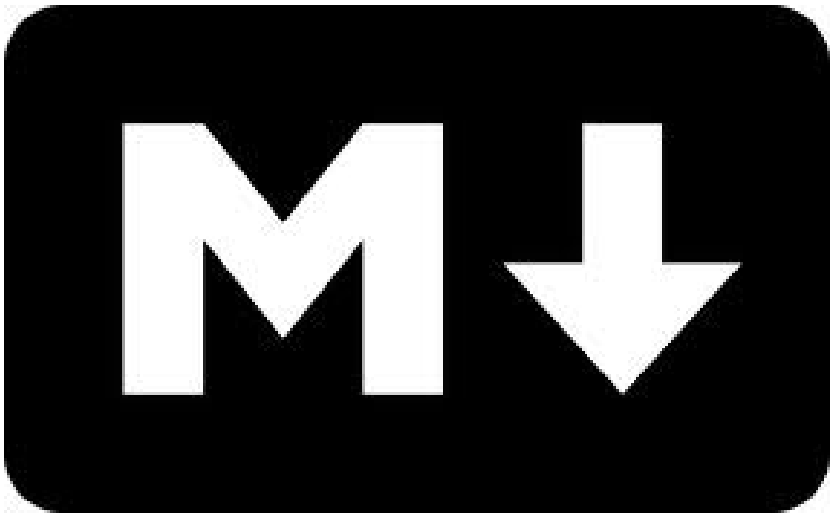
- Donec dolor arcu, rutrum id molestie in, viverra sed diam.
- Curabitur feugiat,
- turpis sed auctor facilisis,
- arcu eros accumsan lorem, at posuere mi diam sit amet tortor.
- Fusce fermentum, mi sit amet euismod rutrum,
- sem lorem molestie diam, iaculis aliquet sapien tortor non nisi.
- Pellentesque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

0.4 Scope

- Scope: documents with much *math* and *computer code*
- Key question: What tools should I use for writing?
- Default answer: LaTeX
- Alternative: MS Word w/math
- Recent popular alternative tools: HTML w/MathJax, Sphinx, Markdown, MediaWiki, IPython notebook


LATEX


SPHINX
Python Documentation Generator




MediaWiki
Because ideas want to be free.

IP[y]: IPython
Interactive Computing

0.5 Scientific writing for the future needs to address many new media

Old days (1985-2005): LaTeX for BW paper output, but now

1. BW books
2. Colorful PDF books (printed and screen)
3. Designed web pages
4. Wikis
5. Bloggs
6. Next new fancy format (iBook w/LaTeX?)

0.6 Fundamental question

When I write some scientific material,

- a LaTeX document,

- a blogg (HTML),
- some web pages (HTML),
- a Sphinx document,
- some Markdown files,

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

Probably not, but I have a solution :-)

0.7 Popular tools anno 2013 and their math support

- **LaTeX**: de facto standard for math-instensive documents
- **pdfLaTeX**, **XeLaTeX**, **LuaLaTeX**: takes over (figures in png, pdf)
- **MS Word**: too clicky math support and ugly fonts, but much used
- **HTML with MathJax**: "full" LaTeX *math*, but too much tagging
- **Sphinx**: somewhat limited LaTeX math support, but great support for web design
- **reStructuredText**: no math support, but transforms to lots of formats (LaTeX, HTML, XML, Word, OpenOffice, ...)
- **Markdown**: somewhat limited LaTeX math support, but minor tagging, transforms to lots of formats (LaTeX, HTML, XML, Word, OpenOffice, ...)
- **IPython notebooks**: Markdown code/math, combines Python code, interactivity, and visualization
- **MediaWiki**: quite good LaTeX math support (cf. Wikipedia)
- Other **wiki** formats: no math support, great for collaborative editing
- **Wordpress**: supports LaTeX *formulas* only, but good blogging support
- **Google blogger**: supports full HTML with MathJax
- **Epydoc**: old tool for Python code documentation
- **Plain text for email**: no math, just raw LaTeX, and no tagging

0.8 LaTeX is very rich; other tools support only some elements

- LaTeX inline math: works with all (LaTeX, MathJax, Sphinx, Markdown, MediaWiki)
- 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)

0.9 LaTeX is very rich; other tools support only some elements

- Figures: all
- Subfigures: LaTeX (`subfigure`)
- Movies: LaTeX (can run separately), just raw embedded HTML in others
- Floating computer code: LaTeX
- Fixed computer code: all
- Floating tables: LaTeX
- Fixed tables: all
- Algorithms: LaTeX
- Margin notes: LaTeX
- Page references: LaTeX
- Footnotes: LaTeX, Sphinx, reStructuredText, MediaWiki
- Bibliography: LaTeX, Sphinx, reStructuredText, MediaWiki
- Hyperlinks: all (but not on paper!)

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

0.10 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.

0.11 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).
- Keys for items in the bibliography are made visible by Sphinx so "bibitems" a la BibTeX must look sensible and consistent.
- 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.

0.12 Concerns III

- Index words can appear anywhere in LaTeX, but should be outside paragraphs in other tools.
- References to tables, program code and algorithms can only be made in LaTeX.
- Figures are floating in 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).

0.13 Solution I: Use a format that translates to many

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

0.14 Solution II: Use Doconce

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

- Can generate LaTeX, HTML, Sphinx, Markdown, MediaWiki, Google wiki, Creole wiki, reST, plain text
- Made for large science books *and* small teaching modules
- Targets books, electronic PDF, PDF for phones, designed web pages, blogs
- Many special features (code snippets from files, embedded movies, admonitions, ...)
- Very effective for generating slides from ordinary text
- Applies Mako: Doconce text is a program
- Less tagged than LaTeX, HTML, Sphinx

0.15 Doconce demo

`http://hplgit.github.com/teamods/writing_reports/`

- 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 blogs
- MediaWiki (Wikipedia, Wikibooks, etc)
- Doconce source code and tutorial

0.16 A tour of Doconce

0.17 Doconce: title, authors, date, toc

```
TITLE: Some Title
AUTHOR: name1 at institution1, with more info, and 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*!

0.18 Doconce: abstract

```
__Abstract.__
Here goes the abstract...
```

Or:

```
__Summary.__
Here goes the summary...
```

0.19 Doconce: 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:

Chapter 1

This is an H1/chapter heading

1.1 This is an H2/section heading

1.1.1 This is an H3/subsection heading

This is an H4/paragraph heading.

This is a paragraph heading.

1.2 Doconce: 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
    1. and sublists with enumerated items starting with `o`
    2. items are just indented as you would do in email

1.3 Doconce: 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": "http://code.google.com/p/doconce/"

# Hyperlink with complete URL as link text
URL: "http://code.google.com/p/doconce/"
```

1.4 Doconce: figures and movies

NOTICE

Figure with HTML and LaTeX info, and caption, *all on one line*:

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

# This figure will be 300 pixels wide in HTML and span 1.2 times
# the linewidth in LaTeX.
```

Movies are also supported:

```
MOVIE: [http://www.youtube.com/embed/P8VcZzgdfSc, width=420 height=315]
```

and rendered as
`http://www.youtube.com/watch?v=P8VcZzgdfSc`

1.5 Doconce: math

Inline math as in LaTeX:

```
...where $a=\int_{\Omega}fdx$ 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 LaTeX:

```
!bt
\begin{align}
\frac{\partial u}{\partial t} &= \nabla^2 u,
label{a:eq}\\
```

```
\nabla\cdot\pmb{v} & = 0
label{b:eq}
\end{align}
!et
```

which is rendered as

$$\frac{\partial u}{\partial t} = \nabla^2 u, \tag{1.1}$$

$$\nabla \cdot \boldsymbol{v} = 0 \tag{1.2}$$

1.6 Doconce: math flexibility

Limit math environments to

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

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

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

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

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

Even though Sphinx, Markdown, and MediaWiki have problems with the latter, Doconce splits it into separate, single equations such that align with labels works accross formats.

1.7 Doconce: 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)            # avoid integer division
    N = int(round(T/dt))      # no of time intervals
    T = N*dt                  # adjust T to fit time step dt
    u = zeros(N+1)            # array of u[n] values
    t = linspace(0, T, N+1)   # time mesh

    u[0] = I                  # assign initial condition
    for n in range(0, N):     # n=0,1,...,N-1
        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)            # avoid integer division
```

```
N = int(round(T/dt))      # no of time intervals
T = N*dt                  # adjust T to fit time step dt
u = zeros(N+1)            # array of u[n] values
t = linspace(0, T, N+1)   # time mesh

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

The bc! command can be followed by a specification of the computer language: `pycod` for Python code snippet, `pypro` for complete Python program, `fcod` for Fortran snippet, `fpro` for Fortran program, and so forth (`c` for C, `cpp` for C++, `sh` for Unix shells, `m` for Matlab).

1.8 Doconce: 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` can be used to choose between 40+ typesettings of computer code in LaTeX
- `pygments` is used for code typesetting in HTML (about 10 different styles)

1.9 Doconce: displaying interactive demo code

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)

1.10 Doconce: tables

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

Gets rendered as

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

1.11 Doconce: newcommands for math

- `newcommands*.tex` files contain newcommands
- Used directly in LaTeX
- Substitution made for many other formats

1.12 Doconce: labels, citations, index, bibliography

Lables, citations, index, and bibliography follow the ideas of 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
where $\Ddt{\u}$ is the material derivative of $\u$.
Equation (ref{mysec:eq:Dudt}) is important in a number
of contexts, see cite{Larsen_et_al_2002,Johnson_Friedman_2010a}.
Also, cite{Miller_2000} supports such a view.

As see in Figure ref{mysec:fig:myfig}, the key equation
features large, smooth regions *and* abrupt changes.

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

===== References =====

BIBFILE: papers.pub
```

The `papers.pub` file must be in Publish format (easy to make from BibTeX).

1.13 Doconce: 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.


\begin{hintadmon}[Hint]
Use `r = random.random()` and define head as `r <= 0.5`.
\end{hintadmon}
!esubex

!bsubex
Compute the probability of getting heads.

!bans
A short answer: 0.5.
!eans

!bsol
A full solution to this subexercise can go here.
!esol
!esubex

!bsubex
Make another program that computes the probability
of getting at least three heads out of 5 throws.
!esubex
```

Solutions/answers can easily be left out of the document.

1.14 Doconce: exercises

Last page gets rendered as follows:

Problem 1: Flip a Coin

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. A short answer: 0.5.

Solution. A full solution to this subexercise can go here.

c) Make another program that computes the probability of getting at least three heads out of 5 throws.

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

1.15 Doconce: exercises

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'],
  'heading': '=====',
  'hints': [],
  'keywords': ['random numbers', 'Monte Carlo simulation'],
  'label': 'demo:ex:1',
  'no': 1,
  'solution': '',
  'solution_file': ['mysol.txt', 'mysol_flip_coin.py'],
  'subex': [{'answer': '',
             'file': None,
             'hints': ['Use `r = random.random()` ...'],
             'solution': '',
             'text': 'Make a program that simulates ...'},
            {'answer': 'A short answer: 0.5.',
             'file': None,
             'hints': [],
             'solution': 'A full solution to this ...',
             'text': 'Compute the probability of ...'},
            {'answer': '',
             'file': None,
             'hints': [],
             'solution': '',
             'text': 'Make another program that computes ...'}],
  'text': '',
  'title': 'Flip a Coin',
  'type': 'Problem'}]
```

1.16 Doconce: use of preprocessors

- Simple if-else tests a la C preprocessor
- `FORMAT` variable can be used to test on format
  - 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!

1.17 Doconce: 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 figures and movies
- Focus on key equations
- Focus on 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)
- Slides are made with HTML5 tools such as reveal.js, deck.js, csss, or dzslides

1.18 Doconce: example on slide code

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

 * Key point 1
 * Key point 2
 * Key point 3 takes very much more text to explain because
   this point is really comprehensive, and although long
   bullet points are not recommended in general, we need
   it here for demonstration purposes

FIGURE: [fig/teacher1, width=100]

Key equation:

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

And maybe a final comment?

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

1.19 Doconce: example on slide code

Last page gets rendered to

1.20 Headline

- Key point 1
- Key point 2

Key equation:

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

And maybe a final comment?

1.21 Doconce: example on slide code with cells

One can introduce a table-like layout with MxN cells and put slide elements in various cell. A cell with position MN is surrounded by bslidecell MN! and eslidecell! tags. Below is an example with a bullet list to the left and a figure to the right (two cells, numbered 00 and 01).

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

!bslidecell 00
!bpop
 * Key point 1
 * Key point 2
 * Key point 3 takes very much more text to explain because
   this point is really comprehensive, and although long
   bullet points are not recommended in general, we need
   it here for demonstration purposes
!epop
```

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

!eslidecell

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

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

1.22 Doconce: example on slide code

Last page gets rendered to

1.23 Headline

- Key point 1
- Key point 2
- Key point 3 takes very much more text to explain because this point is really comprehensive, and although long bullet points are not recommended in general, we need it here for demonstration purposes

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

1.24 Doconce: slide styles

- Supported HTML5 packages:
  - reveal.js
  - deck.js
  - dzslides
  - csss
  - html5slides (experimental)
- **Problem**: each package has its own syntax (though similar)
  - **Solution**: slide code is autogenerated from compact Doconce syntax
- **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

1.25 Doconce: output in HTML

Run in terminal window:

```
doconce format html doconcefile

# Solarized HTML style
doconce format html doconcefile --html_solarized

# Control pygments typesetting of code
doconce format html doconcefile --pygments_html_style=native

# Or use plain <pre> tag
doconce format html doconcefile --no_pygments_html

# Further making of slides
doconce slides_html doconcefile reveal --html_slide_theme=darkgray
```

1.26 Doconce: output for blogging

Two types of blogs are supported:

- Google's 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.

1.27 Doconce: output in pdfLaTeX

```
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
```

1.28 Doconce: output in Sphinx

```
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!

1.29 Doconce: output for wiki

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": `http://en.wikibooks.org/wiki/Wikibooks`:Sandbox)
- Wikipedia

1.30 Doconce: output in 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
```

1.31 Doconce: installation

- Ubuntu: `sudo apt-get install python-doconce` (old version!)
- Source at Googlecode (recommended!)
- Check out source, `sudo python setyp.py install`
- Many dependencies...
- Must have `preprocess` and `mako`
- Need `latex`, `sphinx`, `pandoc`, etc. (see Installation in manual)
- Easy for slides: only `preprocess` is needed :-)

1.32 Writing tips

- See the previous *Concerns I, II and III* slides for issues when writing for multiple formats. However: Doconce makes a fix so that Sphinx and other formats works with labels in align environments :-)
- 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.
- Use `doconce spellcheck *.do.txt` to automatically spellcheck files.
- Avoid page references and footnotes.

1.33 More 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.

1.34 Writing tips for sphinx and other formats

For output formats different from `latex`, `pdflatex`, and `html`, the following points are important:

- Do not use math in section headings or figure captions if output in `sphinx` is wanted (such math are removed in references).
- Let all running text start in column 1 (`sphinx` is annoyed by intended lines).
- Use progressive section headings: after chapter (`=========`) comes section (`=======`), and then subsection (`=====`) before paragraph heading (`===`). "Jumps", say `===` after `======` works fine for `latex`, `pdflatex`, and `html`, but not for `rst` and `sphinx`.
- Place index entries (`\index{keyword}`) before the paragraph where they are introduced (not inside the text). Also place index entries before `===` headings.
- 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.
- Always have a line of running text before a code snippet or program.
- Do not insert comments before intented text, such as computer code and lists.