# Doconce Quick Reference

**Hans Petter Langtangen**[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Apr 29, 2013

## Contents

**WARNING: This quick reference is very incomplete!**

**Mission.** Enable writing documentation with much mathematics and computer code *once, in one place* and include it in traditional LaTeX books, thesis, and reports, and without extra efforts also make professionally looking web versions with Sphinx or HTML. Other outlets include Google's `blogger.com`, Wikipedia/Wikibooks, IPython notebooks, plus a wide variety of formats for documents without mathematics and code.

## 0.1 Supported Formats

Doconce currently translates files to the following formats:

- LaTeX (format `latex` and `pdflatex`)
- HTML (format `html`)
- reStructuredText (format `rst`)
- plain (untagged) ASCII (format `plain`)
- Sphinx (format `sphinx`)
- IPython notebook (format `ipynb`)
- MediaWiki (format `mwiki`)
- (Pandoc extended) Markdown (format `pandoc`)
- Googlecode wiki (format `gwiki`)
- Creoloe wiki (format `cwiki`)
- Epydoc (format `epydoc`)
- StructuredText (format `st`)

For documents with much code and mathematics, the best (and most supported) formats are `latex`, `pdflatex`, `sphinx`, and `html`; and to a slightly less extent `mwiki` and `pandoc`. The HTML format supports blogging on Google and Wordpress.

## 0.2 Emacs syntax support

The file .doconce-mode.el in the Doconce source distribution gives a ”Doconce Editing Mode” in Emacs. Store the file in the home directory and add `(load-file "~/.doconce-mode.el")` to the `.emacs` file.

Besides syntax highlighting of Doconce documents, this Emacs mode provides a lot of shortcuts for setting up many elements in a document:

| Emacs key | Action |
|---|---|
| Ctrl+c f | figure |
| Ctrl+c v | movie/video |
| Ctrl+c h1 | heading level 1 (section/h1) |
| Ctrl+c h2 | heading level 2 (subsection/h2) |
| Ctrl+c h3 | heading level 2 (subsection/h3) |
| Ctrl+c hp | heading for paragraph |
| Ctrl+c me | math environment: !bt equation !et |
| Ctrl+c ma | math environment: !bt align !et |
| Ctrl+c ce | code environment: !bc !ec |
| Ctrl+c cf | code from file: @@@CODE |
| Ctrl+c table2 | table with 2 columns |
| Ctrl+c table3 | table with 3 columns |
| Ctrl+c table4 | table with 4 columns |
| Ctrl+c exer | exercise outline |
| Ctrl+c slide | slide outline |
| Ctrl+c help | print this table |

## 0.3 Title, Authors, and Date

A typical example of giving a title, a set of authors, a date, and an optional table of contents reads

```
TITLE: On an Ultimate Markup Language
AUTHOR: H. P. Langtangen at Center for Biomedical Computing, Simula Research Laboratory & Dept. of Inf
AUTHOR: Kaare Dump Email: dump@cyb.space.com at Segfault, Cyberspace Inc.
AUTHOR: A. Dummy Author
DATE: today
TOC: on
```

The entire title must appear on a single line. The author syntax is

```
name Email: somename@adr.net at institution1 & institution2
```

where the email is optional, the "at" keyword is required if one or more institutions are to be specified, and the `&` keyword separates the institutions (the keyword `and` works too). Each author specification must appear on a single line. When more than one author belong to the same institution, make sure that the institution is specified in an identical way for each author.

The date can be set as any text different from `today` if not the current date is wanted, e.g., `Feb 22, 2016`.

The table of contents is removed by writing `TOC: off`.

## 0.4 Section Types

| Section type | Syntax |
|---|---|
| chapter | `========= Heading ========` (9 =) |
| section | `======= Heading =======` (7 =) |
| subsection | `===== Heading =====` (5 =) |
| subsubsection | `=== Heading ===` (3 =) |
| paragraph | `__Heading.__` (2 _) |
| abstract | `__Abstract.__` Running text... |
| appendix | `======= Appendix: heading =======` (7 =) |
| appendix | `===== Appendix: heading =====` (5 =) |
| exercise | `======= Exercise: heading =======` (7 =) |
| exercise | `===== Exercise: heading =====` (5 =) |

Note that abstracts are recognized by starting with `__Abstract.__` or `__Summary.__` at the beginning of a line and ending with three or more = signs of the next heading.

The `Exercise:` keyword kan be substituted by `Problem:` or `Project:`. A recommended convention is that an exercise is tied to the text, a problem can stand on its own, and a project is a comprehensive problem.

## 0.5 Inline Formatting

Words surrounded by * are emphasized: `*emphasized words*` becomes *emphasized words*. Similarly, an underscore surrounds words that appear in boldface: `_boldface_` becomes **boldface**. Colored words are also possible: the text

```
‘color{red}{two red words}‘
```

becomes two red words.

## 0.6 Lists

There are three types of lists: *bullet lists*, where each item starts with *, *enumeration lists*, where each item starts with o and gets consqutive numbers, and *description* lists, where each item starts with - followed by a keyword and a colon.

```
Here is a bullet list:

 * item1
 * item2
  * subitem1 of item2
  * subitem2 of item2
 * item3

Note that sublists are consistently indented by one or more blanks..
Here is an enumeration list:

 o item1
 o item2
   may appear on
   multiple lines
  o subitem1 of item2
  o subitem2 of item2
 o item3
```

```
And finally a description list:

 - keyword1: followed by
   some text
   over multiple
   lines
 - keyword2:
   followed by text on the next line
 - keyword3: and its description may fit on one line
```

The code above follows.

Here is a bullet list:

- item1
- item2
  - subitem1 of item2
  - subitem2 of item2
- item3

Note that sublists are indented. Here is an enumeration list:

1. item1
2. item2 may appear on multiple lines
   (a) subitem1 of item2
   (b) subitem2 of item2
3. item3

And finally a description list:

**keyword1:** followed by some text over multiple lines

**keyword2:** followed by text on the next line

**keyword3:** and its description may fit on one line

## 0.7 Comment lines

Lines starting with # are treated as comments in the document and translated to the proper syntax for comments in the output document. Such comment lines should not appear before LaTeX math blocks, verbatim code blocks, or lists if the formats `rst` and `sphinx` are desired.

Comment lines starting with ## are not propagated to the output document and can be used for comments that are only interest in the Doconce file.

Large portions of text can be left out using Preprocess. Just place `# #ifdef EXTRA` and `# #endif` around the text. The command line option `-DEXTRA` will bring the text alive again.

When using the Mako preprocessor one can also place comments in the Doconce source file that will be removed by Mako before Doconce starts processing the file.

## 0.8 Inline comments

Inline comments meant as messages or notes, to authors during development in particular, are enabled by the syntax

```
[name: running text]
```

where `name` is the name or ID of an author or reader making the comment, and `running text` is the comment. Here goes an example. The inline comments have simple typesetting in most formats, typically boldface name and everything surrounded by parenthesis, but with LaTeX output and the `-DTOTONOTES` option to `ptex2tex` or `doconce ptex2tex`, colorful margin or inline boxes (using the `todonotes` package) make it very easy to spot the comments.

Running

```
doconce format html mydoc.do.txt --skip_inline_comments
```

removes all inline comments from the output. This feature makes it easy to turn on and off notes to authors during the development of the document.

All inline comments to readers can also be physically removed from the Doconce source by

```
doconce remove_inline_comments mydoc.do.txt
```

## 0.9 Verbatim/Computer Code

Inline verbatim code is typeset within back-ticks, as in

```
Some sentence with ‘words in verbatim style‘.
```

resulting in Some sentence with `words in verbatim style`.

Multi-line blocks of verbatim text, typically computer code, is typeset in between bc xxx! and ec! directives, which must appear on the beginning of the line. A specification `xxx` indicates what verbatim formatting style that is to be used. Typical values for `xxx` are nothing, `cod` for a code snippet, `pro` for a complete program, `sys` for a terminal session, `dat` for a data file (or output from a program), `Xpro` or `Xcod` for a program or code snipped, respectively, in programming `X`, where `X` may be `py` for Python, `cy` for Cython, `sh` for Bash or other Unix shells, `f` for Fortran, `c` for C, `cpp` for C++, `m` for MATLAB, `pl` for Perl. For output in `latex` one can let `xxx` reflect any defined verbatim environment in the `ptex2tex` configuration file (`.ptex2tex.cfg`). For `sphinx` output one can insert a comment

```
# sphinx code-blocks: pycod=python cod=fortran cppcod=c++ sys=console
```

that maps environments (`xxx`) onto valid language types for Pygments (which is what `sphinx` applies to typeset computer code).

The `xxx` specifier has only effect for `latex` and `sphinx` output. All other formats use a fixed monospace font for all kinds of verbatim output.

Here is an example of computer code (see the source of this document for exact syntax):

```
from numpy import sin, cos, exp, pi

def f(x, y, z, t):
    return exp(-t)*sin(pi*x)*sin(pi*y)*cos(2*pi*z)
```

Computer code can also be copied from a file:

```
 @@@CODE doconce_program.sh
 @@@CODE doconce_program.sh  fromto: doconce clean@^doconce split_rst
 @@@CODE doconce_program.sh  from-to: doconce clean@^doconce split_rst
```

The `@@@CODE` identifier must appear at the very beginning of the line. The first specification copies the complete file `doconce_program.sh`. The second specification copies from the first line matching the *regular expression* `doconce clean` up to, but not including the line matching the *regular expression* `^doconce split_rst`. The third specification behaves as the second, but the line matching the first regular expression is not copied (aimed at copying text between begin-end comment pair in the file).

The copied line from file are in this example put inside bc shpro! and ec! directives, if a complete file is copied, while the directives become bc shcod! and ec! when a code snippet is copied from file. In general, for a filename extension `.X`, the environment becomes bc Xpro! or bc Xcod! for a complete program or snippet, respectively. The enivorments (`Xcod` and `Xpro`) are only active for `latex` and `sphinx` outout.

Important warnings:

- A code block must come after some plain sentence (at least for successful output in reStructredText), not directly after a section/paragraph heading, table, comment, figure, or movie.
- Verbatim code blocks inside lists can be ugly typeset in some output formats. A more robust approach is to replace the list by paragraphs with headings.

## 0.10 LaTeX Mathematics

Doconce supports inline mathematics and blocks of mathematics, using standard LaTeX syntax. The output formats `html`, `sphinx`, `latex`, pdflatex', `pandoc`, and `mwiki` work with this syntax while all other formats will just display the raw LaTeX code.

Inline expressions are written in the standard LaTeX way with the mathematics surrounded by dollar signs, as in $Ax = b$. To help increase readability in other formats than `sphinx`, `latex`, and `pdflatex`, inline mathematics may have a more human readable companion expression. The syntax is like

```
$\sin(\norm{\bf u})$|$sin(||u||)$
```

That is, the LaTeX expression appears to the left of a vertical bar (pipe symbol) and the more readable expression appears to the right. Both expressions are surrounded by dollar signs.

Blocks of LaTeX mathematics are written within bt! and et! (begin/end TeX) directives starting on the beginning of a line:

```
!bt
\begin{align*}
\nabla\cdot \pmb{u} &= 0,\\
\nabla\times \pmb{u} &= 0.
\end{align*}
!et
```

This LaTeX code gets rendered as

$$\begin{align*}
\nabla\cdot \pmb{u} &= 0,\\
\nabla\times \pmb{u} &= 0.
\end{align*}$$

Here is a single equation:

```
!bt
\[ \frac{\partial\pmb{u}}{\partial t} + \pmb{u}\cdot\nabla\pmb{u} = 0.\]
!et
```

which results in

$$\frac{\partial\pmb{u}}{\partial t} + \pmb{u}\cdot\nabla\pmb{u} = 0.$$

Any LaTeX syntax is accepted, but if output in the `sphinx`, `pandoc`, `mwiki`, `html`, or `ipynb` formats is also important, one should follow these rules:

- Use only the equation environments `\[`, `\]`, `equation`, `equation*`, `align`, and `align*`.
- MediaWiki (`mwiki`) does not support references to equations.

(Doconce performs extensions to `sphinx` and other formats such that labels in `align` environments work well.)

### Notice

LaTeX supports lots of fancy formatting, for example, multiple plots in the same figure, footnotes, margin notes, etc. Allowing other output formats, such as `sphinx`, makes it necessary to only utilze very standard LaTeX and avoid, for instance, more than one plot per figure. However, one can use preprocessor if-tests on the format (typically `if FORMAT in ("latex", "pdflatex")`) to include special code for `latex` and `pdflatex` output and more straightforward typesetting for other formats. In this way, one can also allow advanced LaTeX features and fine tuning of resulting PDF document.

**LaTeX Newcommands.** The author can define `newcommand` statements in files with names `newcommands*.tex`. Such commands should only be used for mathematics (other LaTeX constructions are only understood by LaTeX itself). The convention is that `newcommands_keep.tex` contains the newcommands that are kept in the document, while those in `newcommands_replace.tex` will be replaced by their full LaTeX code. This conventions helps make readable documents in formats without LaTeX support. For `html`, `sphinx`, `latex`, `pdflatex`, `mwiki`, `ipynb`, and `pandoc`, the mathematics in newcommands is rendered nicely anyway.

## 0.11 Figures and Movies

Figures and movies have almost equal syntax:

```
FIGURE: [relative/path/to/figurefile, width=500] Here goes the caption which must be on a single line.

MOVIE: [relative/path/to/moviefile, width=500] Here goes the caption which must be on a single line. I
```

Note three important syntax details:

1. A mandatory comma after the figure/movie filename,
2. all of the command must appear on a single line,
3. there must be a blank line after the command.

The figure file can be listed without extension. Doconce will then find the version of the file with the most appropriate extension for the chosen output format. If not suitable version is found, Doconce will convert another format to the needed one.

Movie files can either be a video or a wildcard expression for a series of frames. In the latter case, a simple device in an HTML page will display the individual frame files as a movie.

Combining several image files into one can be done by the

```
doconce combine_images image1 image2 ... output_image
```

This command applies `montage` or PDF-based tools to combine the images to get the highest quality.

YouTube and Vimeo movies will be embedded in `html` and `sphinx` documents and otherwise be represented by a link. The syntax is

```
MOVIE: [http://www.youtube.com/watch?v=_O7iUiftbKU, width=420 height=315] YouTube movie.

MOVIE: [http://vimeo.com/55562330, width=500 height=278] Vimeo movie.
```

The latter results in

Vimeo movie. `http://vimeo.com/55562330`

## 0.12 Tables

The table in Section 0.4 was written with this syntax:

```
|----------------c--------|------------------c---------------------|
|      Section type       |        Syntax                          |
|----------------l--------|------------------l---------------------|
| chapter                 | ‘========= Heading ========‘ (9 ‘=‘)   |
| section                 | ‘======= Heading =======‘    (7 ‘=‘)   |
| subsection              | ‘===== Heading =====‘        (5 ‘=‘)   |
| subsubsection           | ‘=== Heading ===‘            (3 ‘=‘)   |
| paragraph               | ‘__Heading.__‘               (2 ‘_‘)   |
|------------------------------------------------------------------|
```

Note that

- Each line begins and ends with a vertical bar (pipe symbol).
- Column data are separated by a vertical bar (pipe symbol).
- There may be horizontal rules, i.e., lines with dashes for indicating the heading and the end of the table, and these may contain characters 'c', 'l', or 'r' for how to align headings or columns. The first horizontal rule may indicate how to align headings (center, left, right), and the horizontal rule after the heading line may indicate how to align the data in the columns (center, left, right).
- If the horizontal rules are without alignment information there should be no vertical bar (pipe symbol) between the columns. Otherwise, such a bar indicates a vertical bar between columns in LaTeX.
- Many output formats are so primitive that heading and column alignment have no effect.

## 0.13 Labels and References

The notion of labels and references (as well as bibliography and index) is adopted from LaTeX with a very similar syntax. As in LaTeX, a label can be inserted anywhere, using the syntax

```
label{name}
```

with no backslash preceding the label keyword. It is common practice to choose `name` as some hierarchical name, say `a:b:c`, where `a` and `b` indicate some abbreviations for a section and/or subsection for the topic and `c` is some name for the particular unit that has a label.

A reference to the label `name` is written as

```
ref{name}
```

again with no backslash before `ref`.

Use labels for sections and equations only, and preceed the reference by "Section" or "Chapter", or in case of an equation, surround the reference by parenthesis.

## 0.14 Citations and Bibliography

Single citations are written as

```
cite{name}
```

where `name` is a logical name of the reference (again, LaTeX writers must not insert a backslash). Bibliography citations often have `name` on the form `Author1_Author2_YYYY`, `Author_YYYY`, or `Author1_etal_YYYY`, where `YYYY` is the year of the publication. Multiple citations at once is possible by separating the logical names by comma:

```
cite{name1,name2,name3}
```

The bibliography is specified by a line `BIBFILE: papers.pub`, where `papers.pub` is a publication database in the Publish format. BibTeX `.bib` files can easily be combined to a Publish database (which Doconce needs to create bibliographies in other formats than LaTeX).

## 0.15 Generalized Citations

There is a *generalized referencing* feature in Doconce that allows a reference with `ref` to have one formulation if the label is in the same document and another formulation if the reference is to an item in an external document. This construction makes it easy to work with many small, independent documents in parallel with a book assembly of some of the small elements. The syntax of a generalized reference is

```
ref[internal][cite][external]

# Example:
As explained in
ref[Section ref{subsec:ex}][in cite{testdoc:12}][a "section":
"testdoc.html#___sec2" in the document
"A Document for Testing Doconce": "testdoc.html" cite{testdoc:12}],
Doconce documents may include movies.
```

The output from a generalized reference is the text `internal` if all `ref{label}` references in `internal` are references to labels in the present document. Otherwise, if cite is non-empty and the format is `latex` or `pdflatex` one assumes that the references in `internal` are to external documents declared by a comment line `# Externaldocuments: testdoc, mydoc` (usually after the title, authors, and date). In this case the output text is `internal cite` and the LaTeX package `xr` is used to handle the labels in the external documents. If none of the two situations above applies, the `external` text will be the output.

## 0.16 Index of Keywords

Doconce supports creating an index of keywords. A certain keyword is registered for the index by a syntax like (no backslash!)

```
index{name}
```

It is recommended to place any index of this type outside running text, i.e., after (sub)section titles and in the space between paragraphs. Index specifications placed right before paragraphs also gives the doconce source code an indication of the content in the forthcoming text. The index is only produced for the `latex`, `pdflatex`, `rst`, and `sphinx` formats.

## 0.17 Capabilities of The Program `doconce`

The `doconce` program can be used for a number of purposes besides transforming a `.do.txt` file to some format. Here is the list of capabilities:

```
Usage: doconce command [optional arguments]
commands: format help sphinx_dir subst replace replace_from_file clean spellcheck ptex2tex expand_comm


# transform doconce file to another format
doconce format html|latex|pdflatex|rst|sphinx|plain|gwiki|mwiki|cwiki|pandoc|st|epytext file.do.txt

# substitute a phrase by another using regular expressions
doconce subst [-s -m -x --restore] regex-pattern regex-replacement file1 file2 ...
(-s is the re.DOTALL modifier, -m is the re.MULTILINE modifier,
 -x is the re.VERBOSE modifier, --restore copies backup files back again)

# replace a phrase by another literally
doconce replace from-text to-text file1 file2 ...
(exact text substutition)

# doconce replace using from and to phrases from file
doconce replace_from_file file-with-from-to file1 file2 ...
(exact text substitution, but a set of from-to relations)

# gwiki format requires substitution of figure file names by URLs
doconce gwiki_figsubst file.gwiki URL-of-fig-dir

# remove all inline comments in a doconce file
doconce remove_inline_comments file.do.txt

# create a directory for the sphinx format
doconce sphinx_dir author='John Doe' title='Long title' \
    short_title="Short title" version=0.1 \
    dirname=sphinx-rootdir theme=default logo=mylogo.png \
    do_file [do_file2 do_file3 ...]
(requires sphinx version >= 1.1)

# replace latex-1 (non-ascii) characters by html codes
doconce latin2html file.html

# walk through a directory tree and insert doconce files as
# docstrings in *.p.py files
doconce insertdocstr rootdir

# remove all files that the doconce format can regenerate
doconce clean

# print the header (preamble) for latex file
doconce latex_header

# print the footer for latex files
doconce latex_footer

# change encoding
doconce change_encoding utf-8 latin1 filename

# guess the encoding in a text
doconce guess_encoding filename

# transform a .bbl file to a .rst file with reST bibliography format
doconce bbl2rst file.bbl
```

```
# split a sphinx/rst file into parts
doconce format sphinx complete_file
doconce split_rst complete_file        # !split delimiters
doconce sphinx_dir complete_file
python automake_sphinx.py

# edit URLs to local files and place them in _static
doconce sphinxfix_local_URLs file.rst

# split an html file into parts according to !split commands
doconce split_html complete_file.html

# create slides from a (doconce) html file
doconce slides_html slide_type complete_file.html

# replace bullets in lists by colored bullets
doconce html_colorbullets file1.html file2.html ...

# grab selected text from a file
doconce grab   --from[-] from-text [--to[-] to-text] somefile

# remove selected text from a file
doconce remove --from[-] from-text [--to[-] to-text] somefile

# run spellcheck on a set of files
doconce spellcheck [-d .mydict.txt] *.do.txt

# transform ptex2tex files (.p.tex) to ordinary latex file
# and manage the code environments
doconce ptex2tex mydoc -DMINTED pycod=minted sys=Verbatim \
        dat=\begin{quote}\begin{verbatim};\end{verbatim}\end{quote}

# make HTML file via pandoc from Markdown (.md) file
doconce md2html file

# make LaTeX file via pandoc from Markdown (.md) file
doconce md2latex file

# expand short cut commands to full form in files
doconce expand_commands file1 file2 ...

# combine several images into one
doconce combine_images image1 image2 ... output_file

# insert a table of exercises in a latex file myfile.p.tex
doconce latex_exercise_toc myfile

# list all labels in a document (for purposes of cleaning them up)
doconce list_labels myfile

# translate a latex document to doconce (requires usually manual fixing)
doconce latex2doconce latexfile

# check if there are problems with translating latex to doconce
doconce latex_dislikes latexfile

# typeset a doconce document with pygments (for pretty print of doconce itself)
doconce pygmentize myfile [pygments-style]

# generate a make.sh script for translating a doconce file to various formats
doconce makefile docname doconcefile [html sphinx pdflatex ...]

# fix common problems in bibtex files for publish import
doconce fix_bibtex4publish file1.bib file2.bib ...

# find differences between two files
doconce diff file1.do.txt file2.do.txt [diffprog]
(diffprog can be difflib, diff, pdiff, latexdiff, kdiff3, diffuse, ...)
```

```
# find differences between the last two Git versions of several files
doconce gitdiff file1 file2 file3 ...
```

## 0.18 Exercises

Doconce supports *Exercise*, *Problem*, *Project*, and *Example*. These are typeset as ordinary sections and referred to by their section labels. Exercise, problem, project, or example sections contains certain *elements*:

- a headline at the level of a subsection containing one of the words "Exercise:", "Problem:", "Project:", or "Example:", followed by a title (required)
- a label (optional)
- a solution file (optional)
- name of file with a student solution (optional)
- main exercise text (required)
- a short answer (optional)
- a full solution (optional)
- one or more hints (optional)
- one or more subexercises (subproblems, subprojects), which can also contain a text, a short answer, a full solution, name student file to be handed in, and one or more hints (optional)

A typical sketch of a a problem without subexercises goes as follows:

```
===== Problem: Derive the Formula for the Area of an Ellipse =====
label{problem:ellipsearea1}
file=ellipse_area.pdf
solution=ellipse_area1_sol.pdf

Derive an expression for the area of an ellipse by integrating
the area under a curve that defines half of the allipse.
Show each step in the mathematical derivation.

!bhint
Wikipedia has the formula for the curve.
!ehint

!bhint
"Wolframalpha": "http://wolframalpha.com" can perhaps
compute the integral.
!ehint
```

If the exercise type (Exercise, Problem, Project, or Example) is enclosed in braces, the type is left out of the title in the output. For example, the if the title line above reads

```
===== {Problem}: Derive the Formula for the Area of an Ellipse =====
```

the title becomes just "Derive the ...".

An exercise with subproblems, answers and full solutions has this setup-up:

```
===== Exercise: Determine the Distance to the Moon =====
label{exer:moondist}

Intro to this exercise. Questions are in subexercises below.

!bsubex
Subexercises are numbered a), b), etc.

file=subexer_a.pdf

!bans
Short answer to subexercise a).
!eans

!bhint
First hint to subexercise a).
!ehint

!bhint
Second hint to subexercise a).
!ehint
!esubex

!bsubex
Here goes the text for subexercise b).

file=subexer_b.pdf

!bhint
A hint for this subexercise.
!ehint

!bsol
Here goes the solution of this subexercise.
!esol
!esubex

!bremarks
At the very end of the exercise it may be appropriate to summarize
and give some perspectives. The text inside the !bremarks-!eremarks
directives is always typeset at the end of the exercise.
!eremarks

!bsol
Here goes a full solution of the whole exercise.
!esol
```

By default, answers, solutions, and hints are typeset as paragraphs. The command-line arguments `--without_answers` and `--without_solutions` turn off output of answers and solutions, respectively, except for examples.

## 0.19 Environments

Doconce environments start with benvirname! and end with eenvirname!, where `envirname` is the name of the environment. Here is a listing of the environments:

- `c`: computer code (or verbatim text)
- `t`: math blocks with LaTeX syntax
- `subex`: sub-exercise
- `ans`: short answer to exercise or sub-exercise

- `sol`: full solution to exercise or sub-exercise
- `quote`: indented text
- `notice`, `summary`, `warning`, `question`, `hint`: admonition boxes with custom title, special icon, and (frequently) background color
- `pop`: text to gradually pop up in slide presentations
- `slidecell`: indication of cells in a grid layout for elements on a slide

## 0.20 Preprocessing

Doconce documents may utilize a preprocessor, either `preprocess` and/or `mako`. The former is a C-style preprocessor that allows if-tests and including other files (but not macros with arguments). The `mako` preprocessor is much more advanced - it is actually a full programming language, very similar to Python.

The command `doconce format` first runs `preprocess` and then `mako`. Here is a typical example on utilizing `preprocess` to include another document, ”comment out” a large portion of text, and to write format-specific constructions:

```
# #include "myotherdoc.do.txt"

# #if FORMAT in ("latex", "pdflatex")
\begin{table}
\caption{Some words... label{mytab}}
\begin{tabular}{lrr}
\hline\noalign{\smallskip}
\multicolumn{1}{c}{time} & \multicolumn{1}{c}{velocity} & \multicolumn{1}{c}{acceleration} \\
\hline
0.0          & 1.4186       & -5.01        \\
2.0          & 1.376512     & 11.919       \\
4.0          & 1.1E+1       & 14.717624    \\
\hline
\end{tabular}
\end{table}
# #else
  |--------------------------------|
  |time  | velocity | acceleration |
  |--l--------r-----------r--------|
  | 0.0  | 1.4186   | -5.01        |
  | 2.0  | 1.376512 | 11.919       |
  | 4.0  | 1.1E+1   | 14.717624    |
  |--------------------------------|
# #endif

# #ifdef EXTRA_MATERIAL
....large portions of text...
# #endif
```

With the `mako` preprocessor the if-else tests have slightly different syntax. An example document contains some illustrations on how to utilize `mako` (clone the GitHub project and examine the Doconce source and the `doc/src/make.sh` script).

## 0.21 Resources

- Excellent ”Sphinx Tutorial” by C. Reller: ”http://people.ee.ethz.ch/ creller/web/tricks/reST.html”