The didactic package*

Daniel Bosk
daniel@bosk.se

7th April 2024

Contents

1 Introduction

This package introduces some environments that are useful for writing teaching material. It also provides a few settings for beamer and memoir to make the resulting document more readable.

2 Usage

The package automatically detects if beamer or memoir is loaded and customizes the document accordingly. That can be prevented, however, by using the nobeamer or nomemoir options. The noarticle is to prevent the article versions of the (beamer) environments to be defined.

nobeamer
noarticle
nomemoir

*This document corresponds to didactic v1.6, dated 2024/04/07. Licensed under the terms of LPPL, version 1.3c or later.

notheorems

The package also provides a few environments for `beamer` and `article` mode. Particularly, it can define theorems and definitions, which might be defined also by other packages. To prevent this, use the `notheorems` option.

inner
outer
top
bottom

When the package customizes `memoir` it sets the inner and outer margins to fit margin notes better. The `inner` and `outer` options can be used to set the inner and outer margins to be used. The `top` and `bottom` options can be used to set the top and bottom margins.

2.1 Beamer blocks and semantic environments for papers

assumption
idea
question
exercise
activity
remark

These environments provides wrappers around Beamer's `block` environment. The idea is to complement the blocks provided by Beamer, such as `theorem`, `definition` and `example`, with some more useful blocks. Each of them provides a block with an appropriate title and optional subtitle and is of a suitable colour. For instance, consider the following.

Idea 1 (These blocks)**.** The idea of these blocks is to be able to use them in both slides and text. This way, we can focus on writing just one version of the content.

That can be produced by the following code.

```
\begin{idea}[These blocks]\label{theidea}
  The idea of these blocks is to be able to use them in both slides and text.
  This way, we can focus on writing just one version of the content.
\end{idea}
```

\ProvideSemanticEnv

We can create new such block environments, that work both with `beamer` and without, by using the `\ProvideSemanticEnv` command. This command takes five arguments:

1. the name of the environment,
2. the name of the block environment to use (optional),
3. the title of the block,
4. options to pass to `thmtools` (optional),
5. the key of the translation of the refname of the block,
6. the key of the translation of the refnames of the block.
7. the key of the translation of the Refname of the block,
8. the key of the translation of the Refnames of the block.

For example, the `idea` block above was created by the following code:

```
\ProvideSemanticEnv{idea}[purpleblock]{Idea}[%
    numbered=no,style=definition
  ]{idea}{ideas}{Idea}{Ideas}
\ProvideTranslation{swedish}{idea}{idé}
\ProvideTranslation{swedish}{ideas}{idéerna}
\ProvideTranslation{swedish}{Idea}{Idé}
\ProvideTranslation{swedish}{Ideas}{Idéerna}
```

lightblock
darkblock

We provide two shades of boxes, light coloured and dark coloured boxes. These can be used to create boxes of various colours easily. These two environments take two mandatory arguments: the first one is a colour, the second is the title.

In the text (not slides) the blocks are not coloured. For instance, consider the following.

Title
This is the body.

That can be produced by the following code.

```
\begin{lightblock}{green}{Title}
  This is the body.
\end{lightblock}
```

blackblock
whiteblock
bwblock
redblock
blueblock
purpleblock
greenblock
yellowblock
orangeblock

For instance, we provide a black block, white block and a black-white block. These take only one mandatory argument: a title.

We also provide a few predefined blocks in various common colours. They all take a title as mandatory argument.

2.2 Tools for code examples

We also provide two commands for working with code examples: `lstexample` and `runpython`.

\lstexample

The `\lstexample` command allows us to typeset the source code of the example. For instance, `\lstexample{python}{hello.py}` produces the following:

```python
"""A Hello World example in Python 3"""

def main():
    print("Hello, world!")

if __name__ == "__main__":
    main()
```

We can also pass optional arguments directly to `minted` that is used to typeset the code.

Example 1. `\lstexample[linenos,highlightlines=4]{python}{hello.py}` produces the following.

```python
"""A Hello World example in Python 3"""

def main():
    print("Hello, world!")

if __name__ == "__main__":
    main()
```

\runpython

We can also run the example code and include its output using the `\runpython` macro.

Example 2. For instance, `\runpython{hello.py}` produces the following.

```
Hello, world!
```

`\codebycode` The `\codebycode` command is simply two `\lstexample` commands side by side: `\codebycode[opt1]{lang1}{file1}[opt2]{lang2}{file2}`.

```python
"""A Hello World example in Python 3"""

def main():
    print("Hello, world!")

if __name__ == "__main__":
    main()
```

```python
"""A Hello World example in Python 3"""

def main():
    print("Hello, world!")

if __name__ == "__main__":
    main()
```

`\runbyrun` We can also put the output side by side using `\runbyrun{hello.py}{hello.py}`.

```
Hello, world!
```

```
Hello, world!
```

2.3 Adding margin notes of different kinds

`\indentedmarginpar` We want to be able to put quite some content in the margins. For this reason we want to have indented paragraphs in the margin notes. We can use the `\indentedmarginpar{text}` command to do this. For instance, consider the output in the margin.

This is an example of an indented margin paragraph.

It can contain multiple paragraphs.

We also want to be able to use titled notes of different kinds. We can use the `\newnotetype[fmtcmd]{type}{title}` command to define a new type of note. For instance `\ltnote` is defined as

`\newnotetype` `\ltnote`

```
\newnotetype{\ltnote}{\GetTranslationWarn{Learning theory}}.
```

Learning theory: This is an example of a learning theory note.

We can also change the formatting of the title. `\newnotetype[\textbf]{\bfnote}{Bold note}` and `\bfnote{This is a bold note.}` gives us the note in the margin.

Bold note: This is a bold note.

Learning theory: Another example of a learning theory note.

3 Implementation

Let's start with the options.

3.1 Options

We have a few negative options, that is, if specified we don't want to do them. This means that we'll need to use conditionals. So each option definition consists first of a `\newif` then setting it to true, and finnally, a `\DeclareOption`. The body of the `\DeclareOption` then sets the `\if` to false. After all the options are declared we'll use `\ProcessOptions`. After that we can have all code actually doing the work, all wrapped in `\if` statements.

`nobeamer` `nobeamertheme` `noarticle` This option is used to disable the `beamer` specific parts of the package. Meaning that we don't customize `beamer` even when we find that it is loaded.

```
\newif\if@didactic@beamer
\@ifclassloaded{beamer}{\@didactic@beamertrue}{\@didactic@beamerfalse}
```

```
\DeclareOptionX{nobeamer}{\@didactic@beamerfalse}
\newif\if@didactic@article
\@ifclassloaded{beamer}{\@didactic@articlefalse}{\@didactic@articletrue}
\newif\if@didactic@beamertheme
\@didactic@beamerthemetrue
\DeclareOptionX{nobeamertheme}{\@didactic@beamerthemefalse}
\DeclareOptionX{noarticle}{\@didactic@articlefalse}
```

nomemoir This option is used to disable the `memoir` specific parts of the package. Meaning that we don't customize `memoir` even when we find that it is loaded.

```
\newif\if@didactic@memoir
\@ifclassloaded{memoir}{\@didactic@memoirtrue}{\@didactic@memoirfalse}
\DeclareOptionX{nomemoir}{\@didactic@memoirfalse}
```

notheorems This option is used to disable defining the usual environments provided by `amsmath`: definition, theorem, etc.

```
\newif\if@didactic@theorems
\@didactic@theoremstrue
\DeclareOptionX{notheorems}{\@didactic@theoremsfalse}
```

inner We also have some key–value options to control the margins.
outer
top
bottom

```
\newcommand\@didactic@margin@inner{20mm}
\newcommand\@didactic@margin@outer{70mm}
\newcommand\@didactic@margin@top{25mm}
\newcommand\@didactic@margin@bottom{40mm}
\DeclareOptionX{inner}[25mm]{\renewcommand\@didactic@margin@inner{#1}}
\DeclareOptionX{outer}[65mm]{\renewcommand\@didactic@margin@outer{#1}}
\DeclareOptionX{top}[25mm]{\renewcommand\@didactic@margin@top{#1}}
\DeclareOptionX{bottom}[40mm]{\renewcommand\@didactic@margin@bottom{#1}}
```

marginparmargin We also have an option to set the marginpar margin. We simply pass this on to `memoir` and keep a record ourselves too. We know that the default in `memoir` is `outer`.

```
\newcommand\@didactic@marginparmargin{outer}
\DeclareOptionX{marginparmargin}{%
  \renewcommand\@didactic@marginparmargin{#1}
  \@ifclassloaded{memoir}{%
    \marginparmargin{#1}
    \strictpagechecktrue
    \checkandfixthelayout
  }{}
}
```

Now that we've declared the options we can process them.

```
\ProcessOptionsX\relax
```

3.2 Customizing `beamer`

If `beamer` is loaded we want to customize it. However, only if the user hasn't disabled this with the `nobeamer` option.

```
\if@didactic@beamer
```

If we use `beamer` we want to use the Berlin theme. This theme guides the viewer very nicely, how must is left and where we are in the presentation. (The same is true for the presenter.)

```
\if@didactic@beamertheme
  \usetheme{Berlin}
```

However, we want to customize the footline. We want to add two lines in the footline. We want to add the author (left) and institute (right) to the first line. Then we add the title (left) and page number (right) to the bottom line.

```
  \setbeamertemplate{footline}%{miniframes theme}
  {%
    \begin{beamercolorbox}[colsep=1.5pt]{upper separation line foot}
    \end{beamercolorbox}
    \begin{beamercolorbox}[ht=2.5ex,dp=1.125ex,%
      leftskip=.3cm,rightskip=.3cm plus1fil]{author in head/foot}%
      \leavevmode{\usebeamerfont{author in head/foot}\insertshortauthor}%
      \hfill%
      {\usebeamerfont{institute in head/foot}%
        \usebeamercolor[fg]{institute in head/foot}\insertshortinstitute}%
    \end{beamercolorbox}%
    \begin{beamercolorbox}[ht=2.5ex,dp=1.125ex,%
      leftskip=.3cm,rightskip=.3cm plus1fil]{title in head/foot}%
      {\usebeamerfont{title in head/foot}\insertshorttitle}%
      \hfill%
      \insertframenumber%
    \end{beamercolorbox}%
    \begin{beamercolorbox}[colsep=1.5pt]{lower separation line foot}
    \end{beamercolorbox}
  }
```

We also want to set the transparency of the covered items. We also want to set the bibliography item to text, we need this to get proper references (not icons) in the bibliography. (We need that for certain bibliography styles.)

```
  \setbeamercovered{transparent}
  \setbeamertemplate{bibliography item}{\relax}
\fi
```

Finally, we want to add a table of contents at the beginning of each section and subsection. We want these to be shaded, so that the current section is highlighted. We also want to hide the subsections of the other sections.

```
\AtBeginSection[]{%
  \begin{frame}<beamer>
    \tableofcontents[currentsection,
                     subsectionstyle=show/show/hide]
  \end{frame}
}
\AtBeginSubsection[]{%
  \begin{frame}<beamer>
    \tableofcontents[currentsection,
                     subsectionstyle=show/shaded/hide]
  \end{frame}
}
\fi% end \if@didactic@beamer
```

3.3 Customizing `memoir`

If `memoir` is loaded we want to customize it. However, only if the user hasn't disabled this with the `nomemoir` option.

```
\if@didactic@memoir
```

We want to set up `memoir` to use the Tufte style. This means that we want to put a lot of text in the margin. We'll use margin notes and put all the footnotes in the margin. We also want to use footnotes for references so that they also appear in the margin.

We want to use the `marginfix` package to fix the margin notes.

```
\RequirePackage{marginfix}
\setlrmarginsandblock{\@didactic@margin@inner}
                     {\@didactic@margin@outer}
                     {*}
\setulmarginsandblock{\@didactic@margin@top}
                     {\@didactic@margin@bottom}
                     {*}

\footnotesinmargin

\RequirePackage{ragged2e}
\renewcommand{\sidefootform}{\RaggedRight}
\renewcommand{\foottextfont}{\footnotesize\RaggedRight}

\setmpjustification{\RaggedRight}{\RaggedRight}

% margin figure and caption typeset ragged against text block
\setfloatadjustment{marginfigure}{\mpjustification}
\setmarginfloatcaptionadjustment{figure}{\captionstyle{\mpjustification}}

% side captions
% https://tex.stackexchange.com/a/275626/17418
\sidecapmargin{outer}
\setsidecappos{t}
\checkandfixthelayout
\setsidecaps{\marginparsep}{\marginparwidth}
\renewcommand{\sidecapstyle}{%
  \captionstyle{\RaggedRight}
}

% From https://tex.stackexchange.com/a/324757/17418
% Palatino for main text and math
\RequirePackage[osf,sc]{mathpazo}
% Helvetica for sans serif
% (scaled to match size of Palatino)
\RequirePackage[scaled=0.90]{helvet}
% Bera Mono for monospaced
% (scaled to match size of Palatino)
\RequirePackage[scaled=0.85]{beramono}
\setlxvchars\setxlvchars
\checkandfixthelayout
\nouppercaseheads
```

We also want to adapt the citation commands of a few packages to use foot-

notes. For these, we check if the package is loaded, if it is, we do the changes. We don't load the packages ourselves, we assume the user has done that. This also means that the `didactic` package should be loaded last.

```
\@ifpackageloaded{biblatex}{%
  \ExecuteBibliographyOptions{%
    autocite=footnote,
    singletitle=false,
    %style=verbose,
    %citestyle=verbose,
    maxbibnames=99,
    isbn=false,doi=false,url=false
  }
  % from https://tex.stackexchange.com/a/374059/17418
  \DeclareCiteCommand{\fullauthorcite}
    {\usebibmacro{prenote}}
    {\usedriver
      {\setcounter{maxnames}{99}% use up to 99 authors
        \DeclareNameAlias{sortname}{default}}
      {\thefield{entrytype}}}
    {\multicitedelim}
    {\usebibmacro{postnote}}
}{}
\@ifpackageloaded{csquotes}{%
  \SetCiteCommand{\footcite}
}{}
```

And that concludes the `memoir` part.

```
\fi% end \if@didactic@memoir
```

3.4 A side-caption environment for `beamer`

`sidecaption` When using `memoir`, we can use the `sidecaption` environment to put the caption in the margin. However, when using `beamer` we don't have this environment, so we need to provide it with a suitable behaviour. It's suitable to just translate it to a normal caption, as usually used in `beamer`.

```
\ProvideDocumentEnvironment{sidecaption}{omo+b}{%
  #4
  \caption{#2\IfValueT{#3}{\label{#3}}}
}{\relax}
```

`\flushscap` We also want to provide a command to flush a figure towards the caption.

```
\NewDocumentCommand{\flushscap}{O{\centering}}{%
  \@ifclassloaded{memoir}{%
    \ifscapmargleft%
      \flushleft%
    \else%
      \flushright%
    \fi%
  }{%
    #1%
  }%
}
```

3.5 More semantic environments

We want to provide a set of environments (blocks) for `beamer`. We want the names of the blocks to be translated. We'll use the `translations` package for this. That way we get the language used through the normal use of `babel`.

We also want to have the same environments for the article mode, but in a nicer format that the default of `beamerarticle`. For instance, instead of

Example
This is an example.

we want something like

Example 3. This is an example.

This means that we'll need to do one thing if `beamer` is loaded and another if `beamerarticle` is loaded—or rather, when `beamer` is not loaded, we should be able to use this without `beamer` and `beamerarticle`.

We want to provide environments like this one:

Exercise 1. This is an exercise.

Exercise 2. This is another exercise.

We can use `\cref` to refer to them, getting something like exercise 1 or exercises 1 and 2.

We also want it to work with different languages, provided there is a translation. We do Swedish here.

Vi kan också skapa exempel på svenska.

Övningsuppgift 3. Detta är ett exempel.

Övningsuppgift 4. Detta är ytterligare ett exempel.

Vi kan referera till dem med `\cref`, vilket ger något som övningsuppgift 3 eller övningsuppgifterna 3 and 4.

Let's also add another idea, but in Swedish.

Idé 2. Detta är en idé.

Då har vi idé 2, men vi hade även tidigare idé 1. Tillsammans är de idéerna 1 and 2.

We can also refer to them in English, idea 2 and idea 1 separately and together as ideas 1 and 2.

`\ProvideSemanticEnv` We provide a command to create such environments. This way we just run this command in the preamble, if `beamer` is loaded it creates the block environments for `beamer`, otherwise it creates the environments for the article.

It can be used like this:

```
\ProvideSemanticEnv{test}[alertblock]{Test}[style=definition]
                      {test}{tests}{Test}{Tests}
```

```
154 \ProvideDocumentCommand{\ProvideSemanticEnv}{m o m o mmmm}{%
155   \@ifundefined{#1}{%
156     \@ifclassloaded{beamer}{% beamer
```

For `beamer` we want to use the `block` environment, or one of the coloured blocks that we provide below. We let the second argument (optional) be the name of the block environment to use. This will be the easiest way to set the colour of the block.

Lastly, we let the third argument be the title of the block. This is the English title and also the key used to translate the title of the block.

```
    \IfValueTF{#2}{%
      \ProvideDocumentEnvironment{#1}{o}{%
        \begin{#2}{\GetTranslationWarn{#3}\IfValueT{##1}{: ##1}}
      }{%
        \end{#2}
      }
    }{%
      \ProvideDocumentEnvironment{#1}{o}{%
        \begin{block}{\GetTranslationWarn{#3}\IfValueT{##1}{: ##1}}
      }{%
        \end{block}
      }
    }
```

If we don't use `beamer`, we want to use the `thmtools` package to define the environments. The fourth argument (optional), will be passed as options to `\declaretheorem`. The fifth to eighth arguments are the keys of (meaning English) translations of the refnames of the block.

```
  }{% not beamer
    \IfValueTF{#4}{%
      \declaretheorem[
        name=\GetTranslationWarn{#3},
        refname={{\GetTranslationWarn{#5}},{\GetTranslationWarn{#6}}},
        Refname={{\GetTranslationWarn{#7}},{\GetTranslationWarn{#8}}},
        #4
      ]{#1}
    }{%
      \declaretheorem[
        style=definition,
        name=\GetTranslationWarn{#3},
        refname={{\GetTranslationWarn{#5}},{\GetTranslationWarn{#6}}},
        Refname={{\GetTranslationWarn{#7}},{\GetTranslationWarn{#8}}}
      ]{#1}
    }
  }
```

Since the `refname` and `Refname` options of `thmtools` doesn't seem to work we will add the necessary `\crefname` and `\Crefname` commands at the beginning of the document, for both `beamer` and non-`beamer` case.

```
  \AtBeginDocument{%
    \@ifpackageloaded{cleveref}{%
      \crefname{#1}
        {\GetTranslationWarn{#5}}
        {\GetTranslationWarn{#6}}
      \Crefname{#1}
        {\GetTranslationWarn{#7}}
        {\GetTranslationWarn{#8}}
    }{%
```

```
        \relax
      }
    }
  }{\relax} % \@ifundefined{#1}
} % \ProvideSemanticEnv
```

assumption idea question exercise activity remark summary

Then let's define some useful environments.

```
\ProvideSemanticEnv{assumption}[alertblock]{Assumption}
  {assumption}{assumptions}{Assumption}{Assumptions}
\ProvideSemanticEnv{idea}[greenblock]{Idea}
  {idea}{ideas}{Idea}{Ideas}
\ProvideSemanticEnv{question}[orangeblock]{Question}
  {question}{questions}{Question}{Questions}
\ProvideSemanticEnv{exercise}[yellowblock]{Exercise}
  {exercise}{exercises}{Exercise}{Exercises}
\ProvideSemanticEnv{activity}[yellowblock]{Activity}
  {activity}{activities}{Activity}{Activities}
\ProvideSemanticEnv{remark}[alertblock]{Remark}[%
    numbered=no,style=remark
  ]{remark}{remarks}{Remark}{Remarks}
\ProvideSemanticEnv{summary}[block]{Summary}[%
    numbered=no,style=remark
  ]{summary}{summaries}{Summary}{Summaries}
```

We also want to provide the normal environments for theorems and definitions and such.

```
\if@didactic@theorems
  \ProvideSemanticEnv{definition}[block]{Definition}
    {definition}{definitions}{Definition}{Definitions}
  \ProvideSemanticEnv{theorem}[block]{Theorem}[%
      numbered=unless unique,style=theorem
    ]{theorem}{theorems}{Theorem}{Theorems}
  \ProvideSemanticEnv{corollary}[block]{Corollary}[%
      numbered=unless unique,style=theorem
    ]{corollary}{corollaries}{Corollary}{Corollaries}
  \ProvideSemanticEnv{lemma}[block]{Lemma}[%
      numbered=unless unique,style=theorem
    ]{lemma}{lemmas}{Lemma}{Lemmas}
  \ProvideSemanticEnv{proof}[block]{Proof}[%
      numbered=no,style=proof
    ]{proof}{proofs}{Proof}{Proofs}
  \ProvideSemanticEnv{solution}[block]{Solution}[%
      numbered=no,style=proof
    ]{solution}{solutions}{Solution}{Solutions}
  \ProvideSemanticEnv{example}[exampleblock]{Example}
    {example}{examples}{Example}{Examples}
\fi
```

3.6 Translations

We also want to provide translations for the environments. We want to provide those even if none of the packages above are loaded.

```
\ProvideTranslation{swedish}{Assumption}{Antagande}
```

```
\ProvideTranslation{swedish}{Assumptions}{Antagandena}
\ProvideTranslation{swedish}{assumption}{antagande}
\ProvideTranslation{swedish}{assumptions}{antagandena}
\ProvideTranslation{swedish}{Idea}{Idé}
\ProvideTranslation{swedish}{Ideas}{Idéerna}
\ProvideTranslation{swedish}{idea}{idé}
\ProvideTranslation{swedish}{ideas}{idéerna}
\ProvideTranslation{swedish}{Question}{Fråga}
\ProvideTranslation{swedish}{Questions}{Frågor}
\ProvideTranslation{swedish}{question}{fråga}
\ProvideTranslation{swedish}{questions}{frågor}
\ProvideTranslation{swedish}{Exercise}{Övningsuppgift}
\ProvideTranslation{swedish}{Exercises}{Övningsuppgifterna}
\ProvideTranslation{swedish}{exercise}{övningsuppgift}
\ProvideTranslation{swedish}{exercises}{övningsuppgifterna}
\ProvideTranslation{swedish}{Activity}{Aktivitet}
\ProvideTranslation{swedish}{Activities}{Aktiviteter}
\ProvideTranslation{swedish}{activity}{aktivitet}
\ProvideTranslation{swedish}{activities}{aktiviteter}
\ProvideTranslation{swedish}{Note}{Anmärkning}
\ProvideTranslation{swedish}{Notes}{Anmärkningar}
\ProvideTranslation{swedish}{note}{anmärkning}
\ProvideTranslation{swedish}{notes}{anmärkningar}
\ProvideTranslation{swedish}{Remark}{Anmärkning}
\ProvideTranslation{swedish}{Remarks}{Anmärkningar}
\ProvideTranslation{swedish}{remark}{anmärkning}
\ProvideTranslation{swedish}{remarks}{anmärkningar}
\ProvideTranslation{swedish}{Summary}{Sammanfattning}
\ProvideTranslation{swedish}{Summaries}{Sammanfattningar}
\ProvideTranslation{swedish}{summary}{sammanfattning}
\ProvideTranslation{swedish}{summaries}{sammanfattningar}
\ProvideTranslation{swedish}{definition}{definition}
\ProvideTranslation{swedish}{definitions}{definitionerna}
\ProvideTranslation{swedish}{Definition}{Definition}
\ProvideTranslation{swedish}{Definitions}{Definitionerna}
\ProvideTranslation{swedish}{theorem}{sats}
\ProvideTranslation{swedish}{theorems}{satserna}
\ProvideTranslation{swedish}{Theorem}{Sats}
\ProvideTranslation{swedish}{Theorems}{Satserna}
\ProvideTranslation{swedish}{corollary}{följdsats}
\ProvideTranslation{swedish}{Corollary}{Följdsats}
\ProvideTranslation{swedish}{corollaries}{följdsatser}
\ProvideTranslation{swedish}{Corollaries}{Följdsatser}
\ProvideTranslation{swedish}{lemma}{hjälpsats}
\ProvideTranslation{swedish}{lemmas}{hjälpsatserna}
\ProvideTranslation{swedish}{Lemma}{Hjälpsats}
\ProvideTranslation{swedish}{Lemmas}{Hjälpsatserna}
\ProvideTranslation{swedish}{proof}{bevis}
\ProvideTranslation{swedish}{Proof}{Bevis}
\ProvideTranslation{swedish}{proofs}{bevisen}
\ProvideTranslation{swedish}{Proofs}{Bevisen}
\ProvideTranslation{swedish}{Solution}{Lösningsförslag}
\ProvideTranslation{swedish}{Solutions}{Lösningsförslagen}
\ProvideTranslation{swedish}{solution}{lösningsförslag}
```

```
\ProvideTranslation{swedish}{solutions}{lösningsförslagen}
\ProvideTranslation{swedish}{Example}{Exempel}
\ProvideTranslation{swedish}{Examples}{Exempelen}
\ProvideTranslation{swedish}{example}{exempel}
\ProvideTranslation{swedish}{examples}{exempelen}
```

3.7 Coloured blocks

lightblock
darkblock
blackblock
whiteblock
bwblock
redblock
blueblock
purpleblock
greenblock
yellowblock
orangeblock

Now we have the coloured blocks. These we want to define even if `beamer` is not loaded, but for this we need `beamerarticle` instead.

```
\ProvideDocumentEnvironment{lightblock}{mm}{%
  \setbeamercolor{block body}{bg=#1!10,fg=black}
  \setbeamercolor{block title}{bg=#1,fg=black}
  \setbeamercolor{local structure}{fg=#1}
  \begin{block}{#2}
}{%
  \end{block}
}
\ProvideDocumentEnvironment{darkblock}{mm}{%
  \setbeamercolor{block body}{bg=#1!10,fg=black}
  \setbeamercolor{block title}{bg=#1,fg=white}
  \setbeamercolor{local structure}{fg=#1}
  \begin{block}{#2}
}{%
  \end{block}
}
\ProvideDocumentEnvironment{blackblock}{m}
{\begin{darkblock}{black}{#1}}
{\end{darkblock}}
\ProvideDocumentEnvironment{whiteblock}{m}{%
  \setbeamercolor{block body}{bg=white!10,fg=black}
  \setbeamercolor{block title}{bg=white,fg=black}
  \setbeamercolor{local structure}{fg=black}
  \begin{block}{#1}
}{%
  \end{block}
}
\ProvideDocumentEnvironment{bwblock}{m}{%
  \setbeamercolor{block body}{bg=white!10,fg=black}
  \setbeamercolor{block title}{bg=black,fg=white}
  \setbeamercolor{local structure}{fg=black}
  \begin{block}{#1}
}{%
  \end{block}
}
\ProvideDocumentEnvironment{redblock}{m}
{\begin{darkblock}{red}{#1}}
{\end{darkblock}}
\ProvideDocumentEnvironment{blueblock}{m}
{\begin{darkblock}{blue}{#1}}
{\end{darkblock}}
\ProvideDocumentEnvironment{purpleblock}{m}
{\begin{darkblock}{purple}{#1}}
{\end{darkblock}}
```

```
\ProvideDocumentEnvironment{greenblock}{m}
{\begin{lightblock}{green}{#1}}
{\end{lightblock}}
\ProvideDocumentEnvironment{yellowblock}{m}
{\begin{lightblock}{yellow}{#1}}
{\end{lightblock}}
\ProvideDocumentEnvironment{orangeblock}{m}
{\begin{lightblock}{orange}{#1}}
{\end{lightblock}}
```

3.8 Adding margin notes of different kinds

We want to add titled margin notes. These margin notes should use indentation since they might contains several paragraphs.

`\indentedmarginpar` We start with the indented margin notes. They simply take a text as argument and typeset it in the margin: `\indentedmarginpar{this is the text}`.

this is the text

```
% Gives us indentation in the margin notes.
% Adapted from https://tex.stackexchange.com/a/257171
\NewDocumentCommand{\indentedmarginpar}{+m}{%
  \@ifclassloaded{memoir}{\strictpagecheck}{\relax}%
  \marginpar{%
    \setlength{\parindent}{1.0em}\footnotesize
    \@afterindentfalse\@afterheading #1
  }%
}
```

`\newnotetype` Now, let's add a command to add margin notes with a title. The `\newnotetype{\titlenote}{Title]` command creates a new command `\titlenote` that takes a text as argument and typesets it in the margin with the title `Title`: `\titlenote{this is the text}`.

Title: this is the text

```
\NewDocumentCommand{\newnotetype}{omm}{%
  \IfValueTF{#1}
    {\NewDocumentCommand{#2}{+m}{%
      \indentedmarginpar{#1{#3:} ##1}%
    }}
    {\NewDocumentCommand{#2}{+m}{%
      \indentedmarginpar{\emph{#3:} ##1}%
    }}%
}
```

`\ltnote` We provide a few different types of margin notes. That way we can also include translations for them.

```
\ProvideTranslation{swedish}{Learning theory}{Lärandeteori}
\newnotetype{\ltnote}{\GetTranslationWarn{Learning theory}}
```

3.9 Listing and running code examples

Let's turn our focus to `\lstexample` and `\runpython`. We want to make it easy to typeset and print the output of example Python programs in slides and texts. We want to use PythonTeX to automatically run the code and typeset the source code using `minted`.

These are the same for both `beamer` and article mode.

\lstexample The first part is easy. To typeset the source file we simply need to use the minted package. We provide an optional argument to pass options to minted.

```
\NewDocumentCommand{\lstexample}{omm}{%
  \IfValueTF{#1}
    {\inputminted[escapeinside=||,#1]{#2}{#3}}
    {\inputminted[escapeinside=||]{#2}{#3}}
}
```

\runpython Now, for the second part, printing the output, we do this in two steps. First, we create a function using PythonTeX that runs a Python program and prints it output. We can use it like `\runpython{hello.py}` to run the program in `hello.py`. The output is printed in verbatim mode. Alternatively we could use `\runpython[opt]{hello.py}` to pass `opt` to the underlying `fancyvrb` environment.

Let's create that function in PythonTeX to run the example program file. We simply execute it with Python 3 and capture its output. Then we print the output to stdout, which is then captured by PythonTeX. We use `pytex.add_dependencies` to add the file as a dependency to rerun the code when necessary.

```
\begin{pycode}
import subprocess

def run_module(m):
  result = subprocess.run(["python3", m], capture_output=True)
  print(result.stdout.decode("utf8").strip())
  pytex.add_dependencies(m)

def mintedoutput(m, opt):
  if opt:
      print(r'\begin{minted}[{}]{{text}}'.format(opt))
  else:
      print(r'\begin{minted}{text}')
  run_module(m)
  print(r'\end{minted}')
\end{pycode}
```

Then we simply create a command that run that function with the file as argument. Then print the output in verbatim mode.

```
\NewDocumentCommand{\runpython}{om}{%
  \setpythontexautoprint{false}%
  \pyc{run_module("#2")}%
  \printpythontex[verbatim]%
}
```

3.10 Contrasting code examples

Sometimes we want to contrast two code examples side by side.

\textbytext For example `\textbytext{\lipsum[1]}{\lipsum[1]}` should typeset the two examples side by side.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

If we use the `memoir` class and our settings (`\if@didactic@memoir`), we want to use the `adjustwidth` environment to expand the text block to better fit them. Otherwise we use the standard text block.

To do this we first add a `fullwidth` environment[1]. We can get the `adjustwidth` environment from the `changepage` package whenever `memoir` isn't used.

```
\RequirePackage{changepage}
\newlength{\@didactic@textbytext@oldcolumnwidth}
\NewDocumentEnvironment{fullwidth}{+b}{%
  \setlength{\@didactic@textbytext@oldcolumnwidth}{\columnwidth}
  \setlength{\columnwidth}{\textwidth+2em+\marginparwidth+\marginparsep}
  \IfStrEqCase{\@didactic@marginparmargin}{%
    {inner}{\begin{adjustwidth*}{-\marginparwidth-\marginparsep}{-2em}}
    {left}{\begin{adjustwidth}{-\marginparwidth-\marginparsep}{-2em}}
    {outer}{\begin{adjustwidth*}{-2em}{-\marginparwidth-\marginparsep}}
    {right}{\begin{adjustwidth}{-2em}{-\marginparwidth-\marginparsep}}
  }[\relax]
    #1
  \IfStrEqCase{\@didactic@marginparmargin}{%
    {inner}{\end{adjustwidth*}}
    {left}{\end{adjustwidth}}
    {outer}{\end{adjustwidth*}}
    {right}{\end{adjustwidth}}
  }[\relax]
  \setlength{\columnwidth}{\@didactic@textbytext@oldcolumnwidth}
}{\relax}
```

Now to the `\textbytext` command. We simply set up a `tabularx` environment with two columns and use it inside a `fullwidth` environment.

```
\NewDocumentCommand{\textbytext}{+m+m}{%
  \begin{fullwidth}
    \begin{tabularx}{\columnwidth}{XX}
      #1 & #2
    \end{tabularx}
  \end{fullwidth}
```

[1] Similarly to `https://tex.stackexchange.com/a/350944/17418`.

```
}
```

`\codebycode` When we deal with code, we deal with verbatim data. The easiest way to deal with this is to simply keep the code in files and supply the file names to `\inputminted`. The `\codebycode` command is simply two `\inputminted` commands side by side: `\codebycode[opt1]{lang1}{file1}[opt2]{lang2}{file2}`.

```
"""A Hello World example in Python 3"""

def main():
    print("Hello, world!")

if __name__ == "__main__":
    main()
```

```
"""A Hello World example in Python 3"""

def main():
    print("Hello, world!")

if __name__ == "__main__":
    main()
```

This gives us the following implementation. We can reuse `\textbytext` to typeset the two examples side by side.

```
\RequirePackage{tabularx}
\RequirePackage{minted}
\NewDocumentCommand{\codebycode}{ommomm}{%
  \textbytext{%
    \IfValueTF{#1}
      {\lstexample[#1]{#2}{#3}}
      {\lstexample{#2}{#3}}%
  }{%
    \IfValueTF{#4}
      {\lstexample[#4]{#5}{#6}}
      {\lstexample{#5}{#6}}%
  }%
}
```

`\runbyrun` We also want to typeset the runs (`\runpython`). We can use `\textbytext` to typeset the two outputs side by side.

```
\NewDocumentCommand{\runbyrun}{mm}{%
  \textbytext{\runpython{#1}}{\runpython{#2}}%
}
```