Advanced LaTeX course; first session

TEXniCie
Jan Jitse Venselaar

February 6th, 2007

Welcome to the Advanced LaTeX course!

Nine sessions, every tuesday between 17:00 and 19:00.

Course overview

1. Commands & environments
2. Counters & lengths
3. Error messages
4. Test
5. Bibtex
6. Moving stuff
7. Pagestyles
8. To be decided
9. Final test

What are we going to do?

1. Organisation
2. User definitions
3. Packaging

New commands

New commands may be defined or redefined under LaTeXwith the commands

`\newcommand{\⟨name⟩}[narg][opt]{def}`

`\renewcommand{\⟨name⟩}[narg][opt]{def}`

The first version is used to define a command that does not yet exist, the second version is used to redefine a command that already exists. A command name may be any combination of letters.

Example: command without arguments

The structure $x_1, \ldots, x_n$ often occurs in mathematical formulas. and is formed in math mode with `x_1,\ldots,x_n`.

```
\newcommand{\\xvec}{x_1,\ldots,x_n}
```

This definition of `\xvec` contains a math command (the underscore). It may only be called within math mode. So:

```
\newcommand{\\xvec}{$x_1,\ldots,x_n$}
```

But now `\xvec` may only be called in text mode, and never in math mode! A trick to solve this is:

```
\newcommand{\\xvec}{\ensuremath{x_1,\ldots,x_n}}
```

Example: command with arguments

We may want to write $a_2, \ldots, a_7$ instead of $x_1, \ldots, x_n$ sometimes. So let us make things variable.

```
\newcommand{\\subvec}[3]{\ensuremath{#1_#2,\ldots,#1_#3}}
```

Now `\subvec{a}{i}{j}` produces $a_i, \ldots, a_j$. However, `\subvec{A}{ij}{kl}` produces $A_ij, \ldots, A_kl$ instead of the wanted $A_{ij}, \ldots, A_{kl}$. Why? Not enough curly brackets!

Solution:

```
\newcommand{\\subvec}[3]{
\ensuremath{#1_{#2},\ldots,#1_{#3}}}
```

Example: command with optional argument

We had the following definition of \subvec, with three arguments:

```
\newcommand{\\subvec}[3]{
\ensuremath{#1_{#2},\ldots,#1_{#3}}}
```

If you want, you can make the first argument optional, in the following way:

```
\newcommand{\\subvec}[3][x]{
\ensuremath{#1_{#2},\ldots,#1_{#3}}}
```

Now \subvec{i}{j} produces $x_i, \ldots, x_j$, while \subvec[a]{i}{j} produces $a_i, \ldots, a_j$.

Spaces after commands

A common mistake is forgetting an extra \ after a command, for example:
\LaTeX is a typesetting language, gives:
LaTeXis a typesetting language.

\xspace, defined in \usepackage{xspace} fixes this. After the bulk of the command, put in \xspace and everything will be fine.

```
\newcommand{\\texnicie}{\TeX nicie\xspace}
```

New environments

New environments may be defined or redefined under LaTeX with the commands

`\newenvironment{\⟨name⟩}[narg][opt]{beg_def}{end_def}`

`\renewenvironment{\⟨name⟩}[narg][opt]{beg_def}{end_def}`

In the `beg_def` you can use arguments, but in `end_def` you cannot!

Example: environment with arguments (1/2)

```
\newenvironment{\comment}[1]{
\noindent\slshape Comment: #1
\begin{quote}\small\itshape}
{\end{quote}}
```

Now \begin{comment}{Anonymous} Let this be a comment.\end{comment} produces:

Comment: Anonymous

> *Let this be a comment.*

Example: environment with arguments (2/2)

You cannot use arguments in the `end_def`. But what if we want to put the name of the person *after* his/her comment? There is a trick to solve this problem:

```
\newsavebox{\comname}
\newenvironment{\comment}[1]{\noindent\slshape
Comment:
\sbox{\comname{#1}}\begin{quote}\small\itshape}
{\hspace*\fill\usebox{comname}\end{quote}}
```

More on user definitions (1/3)

User definitions are scoped. Command and environment definitions made inside an environment are unknown to LaTeX outside the environment in which they were defined.

User definitions may contain other user structures that are defined afterwards. The other structure needs to be defined before the first command is invoked.

You have to work like this: `\newcommand{\\C}{\A \B}`
normal text, without calling `\C`
`\newcommand{\\A}{Command A} \newcommand{\\B}{Command B}`
Now `\C` works without errors.

More on user definitions (2/3)

User definitions may be nested inside one another. For example:

```
\newcommand{\\five}{\newcommand{\\one}{I like \LaTeX}
 \one\one\one\one\one}
```

Now `\five` produces five times the text "I like LaTeX". The command `\one` cannot be used outside the command `\five`.

More on user definitions (3/3)

If you want to provide arguments to nested commands, the symbols for the arguments need to be distinguished. The symbols for the outer definition are the normal #1, . . . , #9, while those for the outer one are ##1, . . . , ##9. For each level of nesting, the amount
of #'s is to be doubled.

Example:

```
\newcommand{\\five}[1]{\newcommand{\\one}[1]{\texttt{##1}}
 \one{#1}\one{#1}\one{#1}\one{#1}\one{#1}}
```

Dirty hacks

To create multiple commands that almost do the same, you can use `\@ifnextchar` as follows:
`\newcommand{\\fancy}{\@ifnextchar*{cmd with star}{cmd without star} Other stuff}`

This command can then be used as follows:
`\fancy` or `\fancy*`

Also really useful if you want to break out of the 1 optional arguments and then required arguments box.
`http://tug.ctan.org/tex-archive/support/newcommand/` provides you with a handy way of doing this.

Structuring user definitions

There are two ways to make a structured collection of your user definitions:

- making a package, *.sty;
- making a class, *.cls.

A class contains information about how to turn logical structure (like `\chapter`) into formatting (like '18pt bold ragged right'). A package contains features independent of the class (such as color or included graphics).

We will go into the details of classes and packages in a following lecture. For now, we will concentrate on making a basic package.

Structure of a package

The outline of a class or package file is:

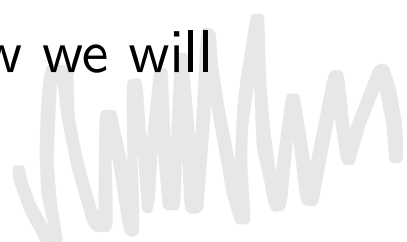
identification the file says it is a LaTeX 2ε package or class, and gives a short description of itself;

declarations the file declares some commands and loads other files. These commands will be just those needed for the code used in the next part;

options the file declares and processes its options;

declarations the file does most of its work: declaring new variables, commands, and fonts, and loading other files.

We will tell more about options in a next lecture, for now we will concern ourselves with the first and fourth item.



Identification

Package files identify themselves by means of the following commands:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{⟨package⟩}[⟨date⟩ ⟨other information⟩]
```

For example:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackag{latexsym}[1993/06/01 Standard LaTeX package]
```

The date is for checking whether your version of the package is outdated.

Declarations (the fourth item)

You can put basically everything in the declaration part of your package. However, packages are loaded in the preamble. Therefore, you cannot put anything in the package that directly generates output.

Usually, this part of a package (or a class) is one long list of newcommands, newenvironments, counters (next lecture), lengths (next lecture), pagestyles (further on), etcetera, etcetera, etcetera.