Module 4

Customising – defining commands, LaTeX packages

In this final module we will look at ways of customising the overall feel of a document, including defining your own commands, loading extra packages, trying different document classes, and fine-tuning the page layout.

There are a handful of exercises in the sequel, and I encourage you to try them yourself before looking at the solutions. Any of the `article`, `book` or `report` document classes should give the required output.

First we will look at ways of managing large amounts of source code.

Splitting the input file

When writing a large document, such as a thesis, it quickly becomes inconvenient to write everything in the one source file. Instead it is often a good idea to split the input over several files, say one per chapter.

The `\input` and `\include` commands both take a filename as their one mandatory argument — it is assumed to be a `.tex` file if no extension is explicitly specified. When LaTeX reaches the `\input` or `\include` command in your source code, it simply reads in the required file at that point, as if its content had been cut-and-pasted there. For this reason your external chapter files etc. should not contain a `\documentclass` command or a `document` environment.

These commands are very useful if you have defined a number of commands, in the file `definition.tex` say, and want to use them for a number of different documents. If your titlepage, abstract, chapters, appendices and bibliography were contained in the files `titlepage.tex`, `abstract.tex`, `introduction.tex`, `preliminaries.tex`, `results.tex`, `appendix.tex` and `references.tex`, then your master file might look something like:

```
\documentclass{book}
 \include{definitions}
\begin{document}
 \include{titlepage}
 \include{abstract}
 \tableofcontents
 \include{introduction}
 \include{preliminaries}
 \include{results}
 \include{appendix}
 \include{references}
\end{document}
```

You would only ever run LaTeX on this file, `master.tex` say, but *never* on the external files `titlepage.tex` etc.

There are a couple of subtle differences between `\input` and `\include`. The `\input` command simply treats `master.tex` as one long document and creates only one auxiliary file `master.aux` to keep track of numbering and cross-referencing.

On the other hand, with `\include` each external file gets its own `.aux` file which deals with the numbering and cross-referencing just for that file. The advantage of this approach is the additional command `\includeonly`, usually used in the preamble. This takes a comma separated list of filenames as an argument, and this specifies which of the `\include`'d files in the document body will actually appear in the output. However, if a previous run created an `.aux` file for an `\include`'d file, then all subsequent page numbering and cross-references to it will be formatted correctly even if it doesn't appear in the output. Thus you can work on one chapter at a time, say the appendix, and have it appear in the shortened output file exactly as it would in the full output:

```
\documentclass{book}
 \include{definitions}
 \includeonly{appendix}
\begin{document}
 \include{titlepage}
 \include{abstract}
 \tableofcontents
 \include{introduction}
 \include{preliminaries}
 \include{results}
 \include{appendix}
 \include{references}
\end{document}
```

Defining new commands

One of the most powerful aspects of LaTeX is the ease with which you can define and use your own commands, whether they be simple shortcuts to frequently used words or symbols, or more complicated structures with mandatory and optional arguments.

The advantage of defining your own commands is twofold. Firstly a lot of typing can be avoided if commonly used passages of text are replaced by a single short command. Secondly, and perhaps more importantly, if you change your mind about the way in which this text should be formatted, you need only change the definition *once* and every occurrence of it in the output will follow suit.

The `\newcommand` command takes two mandatory arguments: the first is the name of your new command, which must begin with a `\` character (note that capitalisation matters); the second is its definition, which may include (almost) any LaTeX source code you like.

In this simplest form, `\newcommand` simply creates a shortcut for the code in your definition:

`\newcommand{\`*name*`}{`*definition*`}`

For example, if you often found yourself typing "without loss of generality" you might define

```
\newcommand{\wlog}{without loss of generality}
```

and then use `\wlog` instead. If you wanted a quicker way to use the `equation` environment you could make the definitions

```
\newcommand{\be}{\begin{equation}}
\newcommand{\ee}{\end{equation}}
```

and then simply write `\be a=b \ee`, much like `\[ a=b \]` or `$$ a=b $$`.

Exercise 4.1. Use `\newcommand` to type the following sentence:

> I say, I say, I say… might I say that I say "I say" quite a lot? I say.

If a new command is to be used throughout the whole document, it's best to put its definition in the preamble. However, a `\newcommand` may appear at any point in the document body, but the definition will only take effect from that point on. It is sometimes useful to restrict the scope of `\newcommand`, much like you would with a declaration, if you only want a command to be valid for a certain portion of the document.

LaTeX will produce an error if you have tried to define a new command with the same name as an existing command. For example

```
\newcommand{\begin}{Once upon a time}
```

produces

```
LaTeX Error: Command \begin already defined.
             Or name \end... illegal, see p.192 of the manual.

See the LaTeX manual or LaTeX Companion for explanation. Type  H
<return>  for immediate help.
 ...

l.86 \newcommand{\begin}{Once upon a time}

?
```

Here you have two choices: choose a different name for your command, or *if you're absolutely sure it's safe* use the `\renewcommand` command, which has precisely the same syntax as `\newcommand`. (By the same token, if you try to use `\renewcommand` to define something which didn't exist already, LaTeX will produce an error.)

If you don't know what a LaTeX command does, its best not to redefine it — it's quite possible that other LaTeX commands need to refer to it and would then not work properly. On the other hand, sometimes it is necessary to redefine an existing command to get the desired effect — in Module 3 we saw how to change the way numbers appear in the output by redefining `\the`*ctr* commands with `\renewcommand`.

The declaration-like scope of command definitions is particularly useful in this context: a command will revert back to its *original* definition if used outside the scope of `\renewcommand`.

Occasionally there will be situations where you will not be able to, or even want to, find out whether a command name exists already. For this reason there is a third variation `\providecommand`. If the proposed command name does *not* exist already then it is defined as in `\newcommand`, otherwise the proposed definition is ignored and the existing command remains intact.

To define a command with n mandatory arguments use `\newcommand`, `\renewcommand` or `\providecommand` with the optional argument n between the name argument and definition argument:

`\newcommand{\`*name*`}[`*n*`]{`*definition*`}`

To substitute the arguments into *definition* use the symbols `#1` for the first, `#2` for the second, etc:

```
\newcommand{\like}[2]{#1 is to #2}

\like{Man}{boy} as
  \like{woman}{girl}.\\
\like{Cold}{winter} as
  \like{hot}{summer}.
```

Man is to boy as woman is to girl.
Cold is to winter as hot is to summer.

A `#...` argument can appear any number of times in the definition, even none. A nice example which you might find handy is the following:

```
\newcommand{\ignore}[1]{}
```

This takes one mandatory argument but produces no output — in effect the argument is ignored. Thus if you needed to temporarily "comment out" a tract of source code you may simply surround it by `\ignore{` and `}`, instead of inserting a `%` at the beginning of each line.

Exercise 4.2. Define a chemical element command with three arguments to typeset the following displayed formula (recall: `$$` begins and ends a displayed formula; use `_` for subscripts and `^` for superscripts; use `\mbox` for plain text in maths-mode; `\to` is a shortcut for `\rightarrow`).

$$\mathrm{N}_7^{15} + \mathrm{H}_1^1 \to \mathrm{C}_6^{12} + \mathrm{He}_2^4$$

Exercise 4.3. Define a command with two arguments and a `\sum` command to typeset the following displayed formula (recall: ∞ is `\infty`).

$$\sum_{i=1}^{\infty} a_i = \sum_{j=1}^{\infty} b_j = \sum_{k=1}^{\infty} c_k = \sum_{l=1}^{\infty} d_l = \sum_{m=1}^{\infty} e_m = \sum_{n=1}^{\infty} f_n$$

A second optional argument *opt* to `\newcommand` allows you to specify that `#1` is an optional argument for your new command:

```
\newcommand{\name}[n][opt]{definition}
```

Here *opt* is the default value of `#1` when the optional argument is *not* given to `\name` — often it will be empty:

```
\newcommand{\hi}[2][]{``Hello#1, #2!''}
\hi{John}\\
\hi[ there]{Jenny}
```

"Hello, John!"
"Hello there, Jenny!"

```
\newcommand{\Sum}[2][i]
  {\sum_{#1=1}^{\infty}#2_{#1}}
$$
\Sum{a}=\Sum{b}=\Sum[j]{c}
$$
```

$$\sum_{i=1}^{\infty} a_i = \sum_{i=1}^{\infty} b_i = \sum_{j=1}^{\infty} c_j$$

Exercise 4.4. Define a new command with one optional argument to typeset the following:

> Hydrofluorocarbons that begin a sentence start with H, but most hydrofluorocarbons, like these hydrofluorocarbons, don't.

Defining new environments

New environments can be defined with `\newenvironment`, or when necessary with `\renewenvironment`. Both commands take three mandatory arguments: the first is the name of the new environment, *name* say (note that it doesn't begin with \); the second is the source code to be produced by the `\begin{`*name*`}` command; the third is the code produced by the `\end{`*name*`}` command. Like `\newcommand`, these commands also take up to two optional arguments: the first is the number n of arguments for the *name* environment; the second *opt* signifies that `#1` is optional and has default value *opt* if not otherwise specified:

`\newenvironment{`*name*`}[`*n*`][`*opt*`]{`*begincode*`}{`*endcode*`}`

Note that the arguments `#1`, `#2` etc. can only be used in the *begincode* part of the definition.

For example, if every table in your document is centered on the page and most of them have three right-justified columns with vertical lines, you might define a new environment as follows:

```
\newenvironment{mytable}[1][|r|r|r|]
  { \begin{table}[ht!]  \centering \begin{tabular}{#1} }
  { \end{tabular} \end{table} }
```

Then it is simply a matter of entering the table entries between `\begin{mytable}` and `\end{mytable}`, and possibly overriding the layout argument for exceptional tables:

```
\begin{mytable}
\hline 1 & 2 & 3 \\
\hline 40 & 50 & 60 \\
\hline
\end{mytable}
```

1	2	3
40	50	60

```
\begin{mytable}[c||c]
12 & 431 \\
0 & 0
\end{mytable}
```

12	431
0	0

The `\newtheorem` command is another powerful tool for defining new environments. It is designed specifically for creating theorem-like environments: theorems, lemmas, corollaries, conjectures and so on. When such an environment is invoked, LaTeX prints a boldface label and number, such as **Theorem 1.**, and then typesets in italics the text which follows.

The `\newtheorem` command takes two mandatory arguments: the first is the name of the environment, and the second is the text of the label:

```
\newtheorem{thm}{Theorem}

\begin{thm}
This is an exciting theorem.
\end{thm}
```

Theorem 1. *This is an exciting theorem.*

LaTeX automatically assigns a counter and a number printing command, in this case `thm` and `\thethm` respectively.

As usual these environments can be cross-referenced with `\label` and `\ref` commands, and their counters and number printing commands can be fully customised with `\setcounter` and `\renewcommand`.

Exercise 4.5. Use `\newtheorem` and `\renewcommand` to typeset the following (refer to page 3·13 of Module 3):

Conjecture A. *Conjectures can be labelled with letters.*

Conjecture B. *I occasionally state the bleeding obvious.*

Two types of optional argument for `\newtheorem` allow you to specify how the numbering is incremented and displayed.

An optional argument *secunit* placed *after* the mandatory arguments specifies that the incrementing should take place within the *secunit* sectional unit. An optional argument *envname* placed *between* the mandatory arguments specifies that this environment will share the numbering sequence of the theorem-like environment *envname*.

Thus in the follwing example `thm` environments are incremented within chapters, and displayed accordingly, and `cor` environments share the same numbering sequence:

```
\newtheorem{thm}{Theorem}[chapter]
\newtheorem{cor}[thm]{Corollary}

\begin{thm}
This is an exciting theorem.
\end{thm}

\begin{cor}
Theorems can be exciting.
\end{cor}
```

Theorem 4.1. *This is an exciting theorem.*

Corollary 4.2. *Theorems can be exciting.*

Later we will see ways of typesetting proofs and definitions.

Boxes

A **box** is essentially a piece of output which LaTeX treats as a unit, as if it were a single character. We've already come across a handful of examples: the `tabular` and `array` environments create boxes which are generally quite large; `\mbox` creates a box which

just contains the text of its argument. Note that boxes are never broken over a page.

The `\fbox` command is similar to `\mbox`, but here a frame is put around the box containing the argument:

```
I highlight \fbox{this phrase}
with a framed box because I
misplaced my red pen.
```

I highlight this phrase with a framed box because I misplaced my red pen.

Actually, `\mbox` and `\fbox` are just special cases of the more powerful commands `\makebox` and `\framebox`. Each takes one mandatory argument which is the text to be put inside the box. There is also an optional argument which specifies the width of the box. If none is given then the default value is just the width of the argument text, so this is then equivalent to `\mbox` or `\fbox` respectively. If the first optional argument *is* given then by default the text is centered within the box. This can be overridden with a second optional argument, one of `l` or `r` which then left- or right-justifies the text:

```
ho \makebox[2cm]{ho} ho\\
ho \makebox[2cm][l]{ho} ho\\
ho \makebox[2cm][r]{ho} ho\\

\framebox[3cm][r]{indeed}
```

ho ho ho
ho ho ho
ho ho ho

indeed

Note that the mandatory text argument for a `\makebox` or `\framebox` cannot contain linebreaks, and the output they create is never broken over a line.

On the other hand, the `\parbox` command creates a box whose argument *can* be broken over several lines. It takes two mandatory arguments: the first is the width of the box and the second is the text itself. An optional argument, one of `t` or `b`, tells LaTeX whether to align the top or bottom of the box with the output line in which it appears; the default is to align the middle of the box:

```
A \parbox{2em}{B C D E F G H I}
\parbox[t]{2em}{J K L M N O P Q}
\parbox[b]{2em}{R S T U V W X Y} Z
```

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

For larger tracts of text there is the related `minipage` environment. It also takes one mandatory argument which is the width of the box to be created, and one optional `t` or `b` argument which specifies vertical alignment. All of the examples in these notes, where source code and output appear side by side such as the `\parbox` example above, were created with a pair of `minipage` environments.

Exercise 4.6. Use the `\texttt` command and a `minipage` environment, with width `10em`, to typeset the following:

This minipage is created with the `minipage` environment.

Note that a `\footnote` command inside a `minipage` environment creates a footnote *at the bottom of the minipage*, and the `mpfootnote` counter is used to handle the numbering.

Text can be raised or lowered with the `\raisebox` command. It takes two mandatory arguments: the second is the text, and the first is the amount by which it should be raised — this may be negative to lower the text instead:

```
\raisebox{-0.8em}{s}\raisebox{-0.6em}{t}%
\raisebox{-0.4em}{a}\raisebox{-0.2em}{i}%
r\raisebox{0.2em}{c}\raisebox{0.4em}{a}%
\raisebox{0.6em}{s}\raisebox{0.8em}{e}
```

$s^{t^{a^{i^{r^{c^{a^{s^{e}}}}}}}}$

(Note the use of `%` to avoid inserting extra space at the end of an input line.)

Exercise 4.7. Use `\raisebox` and the `ex` unit of measurement to typeset the following:

$x_{x_{x^{x}}}$

The `\raisebox` command also takes two optional arguments which, for the purposes of line spacing, tell LaTeX to pretend that the box has a specified height and depth:

```
\raisebox{-0.5cm}[1.5cm][1cm]{ahoy there}
```

tells LaTeX to lower "ahoy there" by 0.5cm, and also pretend that it actually stretches 1.5cm above the line and 1cm below.

Packages

A lot of extra functionality is available in LaTeX by loading additional packages with the `\usepackage` command. This takes the name of a package (`.sty` file) as its one mandatory argument, or a comma separated list of package names. There are dozens and dozens of packages, but we'll briefly describe only a few of them here.

`latexsym`, `amssymb`, `amstex` and `amsthm`

These packages provide a vast array of additional mathematical tools and symbols. See Chapter 8 of *The LaTeX Companion* for a detailed account.

In particular the `amsthm` package provides a `proof` environment which begins a new paragraph with "*Proof.*", unless this is overridden by an optional argument, and ends the last paragraph with a □ symbol:

```
\begin{proof}
Firstly \[ a=b \] so it follows
that $b=a$.
\end{proof}
```

Proof. Firstly

$$a = b$$

so it follows that $b = a$. □

```
\begin{proof}[Alternative proof.]
Obvious.
\end{proof}
```

Alternative proof. Obvious. □

The `amsthm` package also allows you to define theorem-like environments which are *not* typeset in italics, such as you might use for a definition or example. The declaration `\theoremstyle{definition}` tells LaTeX that any `\newtheorem` which follows should be typeset in this way:

```
\newtheorem{thm}{Theorem}[chapter]
\theoremstyle{definition}
\newtheorem{defn}[thm]{Definition}

\begin{defn}
A \emph{group} is a set with a
nice rule for multiplication.
\end{defn}

\begin{thm}
Groups are very interesting.
\end{thm}
```

Definition 4.1. A *group* is a set with a nice rule for multiplication.

Theorem 4.2. *Groups are very interesting.*

`color`

The `color` package allows you to change the colour of text in the output. Note that some DVI viewers may not be able to handle this and will instead show plain black text. In this case you can convert to PDF and then view with Acrobat Reader to see exactly what's going on.

There are a number of predefined colours in the `color` package: `black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan` and `magenta`. It is also possible to define your own with the `\definecolor` command. This command takes three mandatory arguments: the first is a name you choose for the colour, and the second is one of `gray`, `rgb` or `cmyk`.

If the second argument is `gray`, then the third is *one* number between `0` (black) and `1` (white) which specifies the shade of grey.

If the second argument is `rgb`, then the third is a list of *three* numbers between `0` and `1` which specify respectively the amounts of red, green and blue light required to produce the colour.

If the second argument is `cmyk`, then the third is a list of *four* numbers between `0` and `1` which specify respectively the amounts of cyan, magenta, yellow and black in the colour.

```
\definecolor{midgrey}{gray}{0.5}
\definecolor{purple}{rgb}{0.5,0,1}
\definecolor{orange}{cmyk}{0,0.7,1,0}
```

These definitions would most likely appear in the preamble.

Once your colours have been defined, if necessary, it is simply a matter of using either the `\textcolor` command or `\color` declaration. The first takes two arguments, the name of the colour and the text to be coloured; the second takes just the name of the colour as an argument and then affects all of the text in its scope:

```
As you can plainly see:
{\color{red} this is red,}
\textcolor{purple}{this is
purple,} \textcolor{orange}
{and this is orange.}
```

As you can plainly see: this is red, this is purple, and this is orange.

graphics

We have already used the `graphics` package to include graphics with the aptly named `\includegraphics`. There are a number of other commands that are available which allow you to scale, resize, rotate and reflect parts of the output in your document. Again most DVI viewers cannot handle this, so look at the PDF version instead.

The `\scalebox` command takes two mandatory arguments: the second is the text to be scaled, and the first is the factor by which it should be scaled. Thus *any* font size can be achieved in LaTeX, in addition to those specified by the standard sizing declarations such as `\small` and `\Large` (see Table 1.9).

By default the text is scaled by the same factor horizontally and vertically, but the vertical factor can be specified separately with an optional argument:

```
\scalebox{5}{Big}\\

\scalebox{3}[1]{stretched}
```

Big

stretched

The `\resizebox` command has essentially the same effect as `\scalebox`, but here you specify the absolute dimensions of the text, in terms of the units of your choice. It takes three arguments: the first and second are respectively the horizontal and vertical dimensions, and the third is the text itself. The aspect ratio can be automatically maintained by replacing either of the first two arguments with a `!` character:

```
\resizebox{3cm}{1cm}{3cm by 1cm size}\\

\resizebox{10pc}{!}{10pc wide,
  and aspect ratio maintained}
```

3cm by 1cm size

10pc wide, and aspect ratio maintained

The `\rotatebox` command takes two mandatory arguments: the second is the text to be rotated, and the first is the angle measured in degrees *anti-clockwise*:

```
This sentence seems
\rotatebox{10}{a bit bent.}
```

This sentence seems a bit bent.

```
\rotatebox{180}{This
one's upside down!}
```

¡uʍop əpısdn s,ǝuo sıɥT

The `\reflectbox` command takes just one argument, the text to be reflected:

```
\reflectbox{Reflections aren't
easily read.}
```

.pɐǝɹ ʎlıɐsǝ ʇ'nɹǝ suoıʇɔǝlɟǝᴚ

All of the above commands can be used in combination with each other, and with the `\raisebox`, `\framebox` and `\textcolor` commands we saw earlier, to achieve quite complex output.

Exercise 4.8. With `\usepackage{graphics}` in the preamble, typeset the following:

Note that all of these `graphics` commands create a box much like `\mbox` would, and therefore cannot contain text which is broken over a line. To apply the transformations to a larger object, such as a whole paragraph, simply place it within a `\parbox` first.

Exercise 4.9. Use `\rotatebox`, a `\parbox` of width `16ex` and the `tabular` environment to typeset the following:

Item	Price
Eggs	30c ea
Tea	?

Other document classes

To date we have dealt almost entirely with the `book` and `article` document classes, although we have briefly touched on the `report` class also. There are many other document classes (`.cls` files) which can be invoked with the `\documentclass` command. We'll look at the remaining two standard classes here, namely `letter` and `slides`.

`letter`

The `letter` document class can be used to produce any number of letters from the one source file. Usually you begin by defining the sender's name and address in the preamble with the `\signature` and `\address` commands respectively. They both take one argument, which can contain `\\` commands to start new lines if necessary.

Each letter is then written within a `letter` environment. This takes one mandatory argument which is the name and address of the recipient (again including `\\`'s as necessary). Inside the environment you begin with the `\opening` command; it takes one argument such as "Dear Sir," or "To whom it may concern," — you are responsible for punctuation here.

After this point you proceed with the content of your letter, and then finish with the `\closing` command with one argument, such as "Yours sincerely," and so on. An example follows:

```
\documentclass{letter}
  \address{MSI, ANU\\ ACT 0200}
  \signature{Chris Wetherell}
\begin{document}

\begin{letter}{The Convenor\\
Information Literacy Program\\
The Libraries, ANU\\ ACT 0200}

\opening{Dear Sir/Madam,}

I regret to inform you that the
\LaTeX\ course has not been well
received.

\closing{Yours sincerely,}

\cc{Goossens, Lamport}
\encl{Letter of resignation}

\end{letter}

\end{document}
```

MSI, ANU
ACT 0200

16 April, 2003

The Convenor
Information Literacy Program
The Libraries, ANU
ACT 0200

Dear Sir/Madam,

I regret to inform you that the LaTeX course has not been well received.

Yours sincerely,

Chris Wetherell

cc: Goossens, Lamport

encl: Letter of resignation

LaTeX formats the positioning of the following for you:

- the sender's return address, as provided with the `\address` command;
- today's date underneath the return address — for a different date, redefine the `\today` command: eg `\renewcommand{\today}{17 March, 1975}`;
- the recipient's address, as provided as the argument to the `letter` environment;

- the closing and sender's name, as provided by `\closing` and `\signature` respectively.

If, for a particular letter, you need to specify a sender's name or address which is different from the default values set in the preamble, you can override them with a `\signature` or `\address` command between `\begin{letter}{...}` and `\opening{...}`.

There are three additional commands which produce output below the closing of a letter, namely `\cc`, `\encl` and `\ps`. They each take one argument, and are used for a "courtesy copy" list, list of enclosed items, and post-script respectivley.

The `letter` document class is particularly useful if you wish to send multiple copies of the same letter to different recipients. Simply write the content of the letter (including `\opening`, `\closing` etc.) in a separate file, and then use the `\input` or `\include` command to insert it into a separate `letter` environment for each recipient.

Exercise 4.10. Define a command which takes two arguments (a recipient's details, and the name of a letter file) and produces a `letter` environment with the letter file inserted, as described above.

`slides`

LaTeX freely admits to not being terribly good at visual formatting, but there are a couple of reasons you might want to use it to produce slides: your slides might be based on an existing LaTeX document, or they may contain a lot of complicated mathematics.

The `slides` document class differs from other classes in a number of important respects:

- the default font size is much larger, so it can be read from a distance;
- the normal roman family is very similar to sans serif, but only the roman and typewriter families are actually available;
- only the upright and italics font shapes are available;
- page breaking commands are (usually) not allowed;
- there are no floating objects, ie no `table` or `figure` environments.

There are three environments provided by the `slides` document class: `slide`, `overlay` and `note`.

The slides themselves are produced inside a `slide` environment, one for each slide, and these are numbered 1, 2, 3 etc. If the content does not fit on a single page then LaTeX will give a warning, and the excess pages will not be numbered.

The `overlay` environment is used to create a slide which will be placed on top of a previous one. The difference here is that any `overlay` environments between, say,

the fourth and fifth `slide` environment will produce slides numbered 4-a, 4-b, 4-c etc. (A nice way of ensuring that everything lines up correctly in a slide with overlays is to create many copies of the overall effect first, and then "white out" the required regions in each slide with the `color` package and `\textcolor{white}{...}` command.)

The `notes` environment is used to make accompanying notes for the speaker. Any such notes between the fourth and fifth slide will be numbered 4-1, 4-2, etc. You would normally print these notes on plain paper.

The `\onlyslides` command takes one argument, which is a comma separated list of slide numbers or ranges of numbers, and the effect is to produce in the output only those listed slides and any associated overlays. Similarly, the `\onlynotes` command sees that only those notes associated with listed slide numbers are produced. These can be very useful for printing: first use `\onlyslides{1-999}` to produce only the slides and overlays for printing on overhead transparencies, and then `\onlynotes{1-999}` to produce all of the notes to be printed on plain paper.

Document class options

There are a number of optional arguments for the `\documentclass` command which influence the look of the output, for example

```
\documentclass[12pt,draft,a4paper,landscape]{article}
```

The following is a list of the options available for the five standard document classes, `book`, `report`, `article`, `letter` and `slides`. Where there is more than one listed, the first one is the default unless stated otherwise.

`10pt, 11pt, 12pt :` Specify the size of normal type. Not available for `slides`.

`oneside, twoside :` Format the document for one- or two-sided printing. Two-sided output has reflected margins and page numbers. The default for `book` is `twoside`. Not available for `slides`.

`onecolumn, twocolumn :` Produce one or two columns in the output. Not available for `letter` or `slides`.

`landscape :` Interchange the paper width and height, for printing sideways.

`letterpaper, legalpaper, executivepaper, a4paper, a5paper, b5paper :` Specify the paper size. The default for LaTeX is `letterpaper` but, depending on their local configuration, commands like `dvips` and `dvipdf` which produce printable documents will most likely default to A4.

`final, draft :` The `draft` option marks with a black box the location of any output text which overhangs a margin, as a visual aid for fine-tuning the spacing.

`openright`, `openany` : Specify whether chapters begin on right-hand pages or on any page. The default for `report` is `openany`. Not available for `article`, `letter` or `slides`.

`titlepage`, `notitlepage` : Specify whether titlepages and abstracts appear on a separate page. The default for `article` is `notitlepage`. Not available for `letter`.

`openbib` : Specify that the `\newblock` command (used by BibTeX) starts a new line in a bibliographic entry. Not available for `letter` or `slides`.

`leqno` : Number equations on the left.

`fleqn` : Align displayed formulas on the left.

Customising page layout

The layout of a page is partly determined by the default values of a number of **lengths**. Much like `\setcounter` and `\addtocounter`, there are commands `\setlength` and `\addtolength` which can be used to set the value of a length. They each take two arguments: the first is the name of a length; the second is the length to be set. Any of the usual units `mm`, `cm`, `in`, `pt`, `pc`, `ex` and `em` can be used in the second argument, but you can also use the name of another length as a unit! For example

```
\setlength{\textwidth}{2\parindent}
```

tells LaTeX to set the `\textwidth` length to double the length of `\parindent`. You can even set a length in terms of itself:

```
\addtolength{\textwidth}{0.5\textwidth}
```

increases the length of `\textwidth` by 50%. In a similar vein, any defined length can be used as a unit in the argument of commands such as `\makebox`, `\parbox` etc.

Three additional commands for setting the value of a length are `\settowidth`, `\settoheight` and `\settodepth`. Here the length is set to, respectively, the width, height or depth of the second argument:

```
\settowidth{\textwidth}{This will be quite a narrow page.}
```

Figure C.3 on page 182 of *LaTeX: A Document Preparation System* gives a very nice diagram of the standard page layout lengths. The following is a brief summary:

`\paperheight` and `\paperwidth` : The dimensions of the piece of paper LaTeX is planning for, usually set by the papersize (eg `a4paper` or `letterpaper`)

`\textheight` and `\textwidth` : The dimensions of the **Body**, that is, the rectangle into which LaTeX puts the main text of your document. This is essentially the usable printable area.

`\topmargin` : No, not the distance from the top of the page to the top of the **Head**. LaTeX actually automatically puts 1 in. of space at the top of the page, and the top of the Head is `\topmargin` below this.

`\headheight` : The height of the Head part of the page. (The width is the same as `\textwidth`.)

`\headsep` : The amount of space LaTeX leaves between the bottom of the Head and the top of the Body.

`\footskip` : The distance from the bottom of the Body to the bottom of the **Foot**.

`\evensidemargin` and `\oddsidemargin` : Like at the top of the page, LaTeX leaves 1 inch of space on the left. The left-hand edge of the Body starts `\evensidemargin` or `\oddsidemargin` from this, depending what page number it is (useful for double-sided printing, to allow space for binding).

`\marginparsep` : The space between the right-hand edge of the Body and the left-hand edge of any **Marginal Note**.

`\marginparwidth` : The width of any Marginal Note.

Usually you would override the default values of these lengths, with `\setlength` etc, in the preamble.

Another useful length is `\arraycolsep` which adds space between the columns of a `tabular` or `array` environment. Closely related is the command `\arraystretch` which allows you to specify the overall spacing of a table or array. It is *not* a length; its default value is `1` and this can be reset with `\renewcommand` to a factor by which tables and arrays should be stretched:

```
\renewcommand{\arraystretch}{2}
```

The `\newlength` command allows you to define your own lengths. It takes one argument, the name of the length which must start with a `\` character.

This can be very useful for typesetting similar structures throughout your document. Perhaps you want all of your important displayed formulas to appear inside a framed box of width 10cm:

```
$$ \framebox[10cm]{$a=b$} $$
$$ \framebox[10cm]{$e^{x}\not=1$} $$
```

If you then came across a formula which was wider than 10cm and you wanted all of the boxes to be consistent, you would have to change each occurrence by hand. Instead you could start with this:

```
\newlength{\eqwidth}
\setlength{\eqwidth}{10cm}
$$ \framebox[\eqwidth]{$a=b$} $$
$$ \framebox[\eqwidth]{$e^{x}\not=1$} $$
```

Then when the troublesome wider formula comes along you need only change the length of `\eqwidth` *once*, say with `\setlength{\eqwidth}{12cm}`, and then every occurrence of boxed formulas will follow suit.

Customising page style

Recall that the `\pagenumbering` command specifies how page numbers are to be printed. Here the argument is one of `arabic`, `roman`, `Roman`, `alph` or `Alph`.

Another way of customising the look of your document is with the `\pagestyle` command in the preamble. This determines what information goes into the head and foot of each page. There are four arguments which can be used with `\pagestyle`:

`empty` : The head and foot are both empty. LaTeX does still assign each page a number, but it is not printed.

`plain` : The page number is in the foot and the head is empty. This is the default for `article` and `report`.

`headings` : The page number and other information (determined by the document class) is put in the head, and the foot is empty. The "other information" can be overridden using the `\markboth` and `\markright` commands.

`myheadings` : Like `headings`, except that you have to specify what the "other information" is that goes into the head, using the `\markboth` and `\markright` commands.

The `\markboth` command takes two arguments: what goes in the head of left- and right-hand pages respectively. The `\markright` command takes only one argument: what goes in right-hand pages. Note that in one-sided printing, say in the `report` document class, every page is considered to be a right-hand page.

The style of an individual page can be overridden with the `\thispagestyle` command. It only applies to the page in which the command appears, but its syntax is the same as `\pagestyle`.

You can also switch between one- and two-column output for an individual page with the `\onecolumn` and `\twocolumn` declarations (compare with the `onecolumn` and `twocolumn` optional arguments for `\documentclass`). They each start a new page and affect the format of all subsequent pages within their scope. An optional argument for `\twocolumn` will be printed at the top of the page, spanning both columns:

```
\twocolumn[\Huge\centering \LaTeX\ course almost over]

ANU: Students breathed a sigh of relief today when...
```

The `list` environment

The `list` environment produces a list, much like `enumerate`, `itemize` and `description`, but here you have complete control over the spacing and labelling of list items, and which counter should be used, if any.

If there doesn't already exist an appropriate counter, you can define one yourself with the `\newcounter` command. It takes one mandatory argument, the name of the counter (which does *not* start with \):

```
\newcounter{banana}
```

Here `banana` will start with a value of `0`. A number printing command is also defined, in this case `\thebanana`, and by default this will print in arabic form.

The `list` environment takes two mandatory argument: the first specifies what label is to be printed (which may depend on a counter); the second is a list of commands telling LaTeX which counter to use and what lengths to set. The counter is chosen with the `\usecounter` command, and the lengths are set with `\setlength` or `\addtolength`:

```
 Here's some text to end the
  previous paragraph.
\begin{list}
 {\textbf{B--\arabic{banana}:} }
 {\usecounter{banana}
  \setlength{\itemindent}{.5cm}
  \setlength{\leftmargin}{1in}
  \setlength{\rightmargin}{\leftmargin}
  \setlength{\parsep}{0cm}
  \setlength{\itemsep}{0cm} }
\item Bananas are terribly nice to eat.
\item Especially when they come in pears.
\end{list}
```

Here's some text to end the previous paragraph.

B–1: Bananas are terribly nice to eat.

B–2: Especially when they come in pears.

Figure 6.3 on page 113 of *LaTeX: A Document Preparation System* gives a very nice diagram of the available list lengths. The following is a brief summary:

`\labelsep` : The distance from the right edge of the label to the start of the item text.

`\labelwidth` : The width of the label.

`\itemindent` : How far the first line *of text* of each item (not the label) is indented from the other lines.

`\leftmargin`, `\rightmargin` : How far each item is indented from the margins on the left and right.

`\parsep` : Vertical distance between paragraphs in an item.

`\itemsep` : Extra vertical distance between items.

`\listparindent` : How far the paragraphs of an item are indented.

Not every length need be defined in the second argument of a `list` environment; those which aren't will take on their default values. There is also no need to use a counter with `\usecounter` if the label in the first argument doesn't depend on it.

Solutions to exercises

These are just some of the possible solutions — you may have come up with something slightly different which is just as valid. Remember that all multiple spaces are ignored, but it is sometimes helpful to set out your source code in a logical way.

(4.1)
```
\newcommand{\s}{I say}
\s, \s, \s\ldots\ might \s\ that \s\ ``\s'' quite a lot? \s.
```

(4.2)
```
\newcommand{\el}[3]{\mbox{#1}_{#2}^{#3}}
$$\el{N}{15}{7}+\el{H}{1}{1}\to\el{C}{12}{6}+\el{He}{4}{2}$$
```

(4.3)
```
\newcommand{\Sum}[2]{\sum_{#1=1}^{\infty}#2_{#1}}
$$\Sum{i}{a}=\Sum{j}{b}=\Sum{k}{c}=\Sum{l}{d}=\Sum{m}{e}=\Sum{n}{f}$$
```

(4.4)
```
\newcommand{\hydro}[1][h]{#1ydrofluorocarbons}
\hydro[H] that begin a sentence start with H, but most \hydro, like
these \hydro, don't.
```

(4.5)
```
\newtheorem{cnj}{Conjecture}
\renewcommand{\thecnj}{\Alph{cnj}}

\begin{cnj} Conjectures can be labelled with letters. \end{cnj}
\begin{cnj} I occasionally state the bleeding obvious. \end{cnj}
```

(4.6)
```
\begin{minipage}{10em}
This minipage is created with the \texttt{minipage} environment.
\end{minipage}
```

(4.7)
```
\raisebox{1ex}{x}x\raisebox{-1ex}{x}x\raisebox{1ex}{x}
```

(4.8)
```
\rotatebox{45}{\fbox{\reflectbox{\scalebox{4}{J}}}}
```

(4.9)
```
\rotatebox{90}{ \parbox{16ex}{
  \begin{tabular}{|l|c|}
   \hline Item & Price \\
   \hline Eggs & 30c ea\\
          Tea  & ?     \\
   \hline
  \end{tabular} } }
```

(4.10)
```
\newcommand{\ltr}[2]{\begin{letter}{#1}\include{#2}\end{letter}}
```