Module 1

Introduction — running LaTeX, chapters, special commands

Reference material

I hope you will be able to use these course notes as a handy guide in the future. However they almost certainly won't be able to meet all of your specfic requirements, so I recommend the following references. Your department will hopefully have a copy of at least one of the books listed — the ANU library also has a limited number of copies.

- *LaTeX: A Document Preparation System*
  Leslie Lamport
  2nd ed., Addison-Wesley, Reading, Massachusetts, 1994.

  A very nicely written introduction by the creator of LaTeX.

- *The LaTeX Companion*
  Goossens, Mittelbach and Samarin
  Addison-Wesley, Reading, Massachussets, 1994.

  A more complete guide to all of LaTeX's functionality, including many of the specialised packages which are now available. Chapter 8, "Higher Mathematics", is available at `http://www.ctan.org/tex-archive/info/companion-rev/ch8.pdf`

- *The TeX User's Group* (TUG)
  `http://www.tug.org`

- *The Comprehensive TeX Archive Network* (CTAN)
  `http://www.ctan.org`

A brief history of LaTeX

In the 1970s Donald Knuth invented a mathematical typesetting package called TeX, pronounced "tek" — the "X" represents the Greek letter "χ". In the 1980s Leslie Lamport significantly increased its user-friendliness and functionality with the package LaTeX, pronounced "lah-tek" or "lay-tek", which is essentially just a set of macros for the existing TeX system. Today there are numerous packages which extend LaTeX's capabilities even further, and new ones are being written all the time. (It is convention to write `TeX` and `LaTeX` when you can't use the special symbols, in email for example.)

Unlike a WYSIWYG[1] editor like Microsoft Word, LaTeX requires that we encode our typesetting instructions with a series of special commands in a plain text file. Sounds like a lot of hard work.

So why use LaTeX? For a mathematician the answer is easy: just ask Knuth why he had to invent TeX in the first place. But even if you aren't interested in typesetting complex mathematical formulæ, there are many reasons why you might use LaTeX. Here are just a few:

- LaTeX makes a lot of typesetting decisions for you: spacing, linebreaking, hyphenating, pagebreaking, location of tables etc. are all decided globally to give the neatest looking output;
- numbering of chapters, sections, pages, equations etc. is done automatically;
- cross-referencing, bibliography citations, table of contents etc. are painless;
- the layout of a table is determined by its contents;
- the source code can be split into a number of manageable files, and can also contain comments, without affecting the output.

How to run LaTeX, and what happens when we do

Table 1.1 on page 1·4 gives an overview of the most common commands you will need to know on PC or unix, and in particular how to run LaTeX.

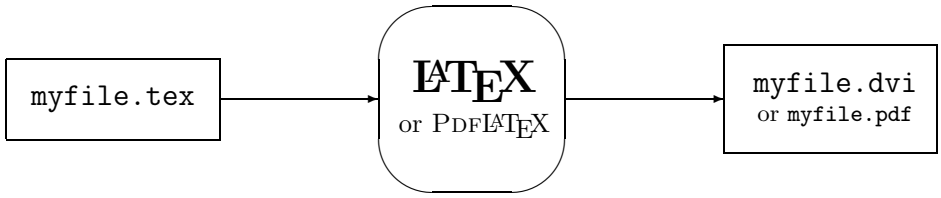


Figure 1.1: What we want LaTeX to do for us

LaTeX is designed to read a text document, say `myfile.tex`, and produce a nicely formatted output, `myfile.dvi` (or `myfile.pdf`) — see Figure 1.1. When we run LaTeX

[1] "What you see is what you get"

there will be (somewhere, depending on your platform) an interactive "running commentary" on how the compilation is proceeding, including any errors and warnings. If the compilation runs across an error, typing '`s`' asks LaTeX to press on and do the best it can, and typing '`x`' tells it to give up.

Behind the scenes the process is far more complicated than Figure 1.1. There are a number of *auxiliary* files that LaTeX uses to keep track of a vast amount of formatting information. For example `myfile.aux` ensures correct labels are used for cross-referencing and citations, while `myfile.toc` oversees the typesetting of a table of contents.

Every time we run LaTeX it *reads in* any existing auxiliary files, compiles `myfile.tex` using this additional information, and finally *writes* a new version of any auxiliary files it will need next time — see Figure 1.2.

> **Note: because it always reads in the *old* auxiliary files, it is sometimes necessary to run LaTeX twice to achieve the expected output.**

This is especially true when there are additions or changes to citations and cross-referencing (the running commentary will usually give you a warning in this case).

LaTeX also creates a log file, `myfile.log`, which contains a transcript of the running commentary.

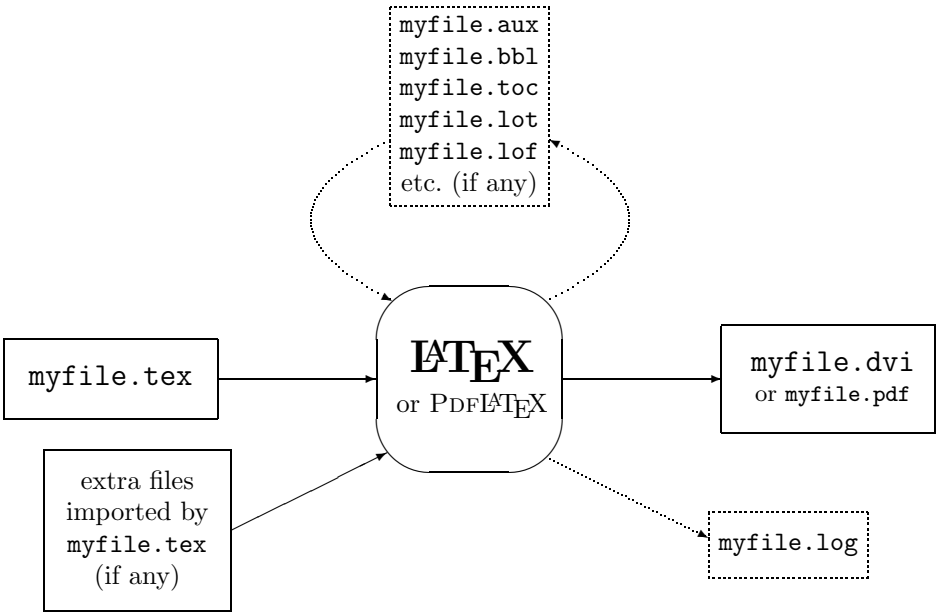


Figure 1.2: What LaTeX actually does

You needn't worry about what's actually going on in the auxiliary files — just be aware that LaTeX needs to use them, and that you should avoid creating files of your own with the extensions `.aux`, `.bbl` etc.

	PC (`WinEdt`)	**Unix** (`emacs`)
Edit `myfile.tex`	Open `WinEdt` and start new document, or double click on `myfile.tex` in folder	`> emacs myfile.tex &` (recommended, but you could use any other editor)
Compile `myfile.tex` to DVI file with LaTeX	Click on  button, or choose `LaTeX` from `Accessories` menu, or type `Ctrl+Shift+L`	`> latex myfile.tex` or choose `LaTeX` from `Command` menu in `emacs`
Compile `myfile.tex` to PDF file with PdfLaTeX	Click on  button, or choose `PDFLaTeX` from `Accessories/PDF` menu	`> pdflatex myfile.tex`
Run BibTeX	Choose `BibTeX` from `Accessories` menu, or type `Ctrl+Shift+B`	`> bibtex myfile.tex` or choose `BibTeX` from `Command` menu in `emacs`
Convert `myfile.dvi` to PDF file	Click on  button, or choose `dvi2pdf` from `Accessories/PDF` menu	`> dvipdf myfile.dvi`
Convert `myfile.dvi` to postscript (`.ps`) file	Click on  button, or choose `DVIPS` from `Accessories` menu, or type `Ctrl+Shift+D`	`> dvips myfile.dvi`
Convert `myfile.ps` to PDF file	Click on  button, or choose `ps2pdf` from `Accessories/PDF` menu	`> ps2pdf myfile.ps`
View `myfile.dvi`	Click on  button, or double click on `myfile.dvi` in folder (opens `Yap`)	`> xdvi myfile.dvi &`
View `myfile.pdf`	Click on  button, or double click on `myfile.pdf` in folder (opens `Acrobat Reader`)	`> acroread myfile.pdf &`
View `myfile.ps`	Click on  button, or double click on `myfile.ps` in folder (opens `GSView`)	`> gv myfile.ps &` or `> ghostview myfile.ps &`
Print output	Choose `Print` from `Yap`, `Acrobat Reader` or `GSView`	`> lpr myfile.ps` or choose `Print` from `gv`, `ghostview` or `acroread`

Table 1.1: Summary of commands — apologies to Mac users!

A first LaTeX document

Before going too heavily into the details, let's get the hang of compiling files. In the editor of your choice type something along the lines of

```
\documentclass{article}

\begin{document}

This is my very first attempt writing a LaTeX file. Or you can write
something different if that's not true...


This is    a  new paragraph, and one of the words might need some
hyphenating.

\end{document}
```

Make sure you use *curly braces* where I have! Now save the file as "*something*`.tex`" and, referring to Table 1.1, compile the code.

A couple of things to notice already:

- LaTeX treats single linebreaks like spaces.
- Blank lines tell LaTeX to start a new paragraph.
- Indenting is done automatically, with the possible exception of the first paragraph.
- LaTeX ignores multiple spaces and multiple blank lines; for example the text "`...is    a  new...`" compiles as "...is a new...".
- LaTeX puts extra space at the end of sentences.
- LaTeX breaks long words over a line to avoid bad spacing.

If all you ever wanted to do is write paragraph after paragraph of plain text, then that's all there is to it... more or less.

Special characters

There are ten characters which have special meanings, so putting them directly into your source code will not do what you might expect. Table 1.2 lists these characters, gives a brief description of their special meaning to LaTeX, and most importantly tells us how to achieve the actual symbol in the output.

Character	Usage	Command	Output
`%`	followed by comment	`\%`	%
`~`	non-breaking space	`\~{}`	~
`\`	start of command name	`$\backslash$`	\
`{`	delimiter, eg for command arguments	`\{`	{
`}`	delimiter	`\}`	}
`#`	part of argument name in definition of new command	`\#`	#
`&`	column marker in tables	`\&`	&
`$`	start or end of maths-mode	`\$`	$
`^`	superscript in maths-mode	`\^{}`	^
`_`	subscript in maths-mode	`\_`	_

Table 1.2: LaTeX's special characters

The character `%` is not just useful for comments. LaTeX ignores everything between a `%` and the end of the line[2]. Therefore it may also be used to break the input over a line without introducing a space in the output. For example

```
This editor may not have 100\% of the room required to avoid line%
breaking.
```

produces

This editor may not have 100% of the room required to avoid linebreaking.

Of course in this case we could get around the problem by breaking the input line between `avoid` and `line`, but from time to time situations will arise where it really is best to use this trick — we will see an example in Module 4.

By avoiding special characters, or replacing them by their corresponding command from Table 1.2 if necessary, you're now in a position to produce... a rather dull looking novel, say.

That's the basics. The rest of the course will be devoted to learning ways of producing more elaborate output.

[2] unless preceded by a '\'

Commands, declarations and environments

The special characters `{` and `}` are **delimiters** — they must always match each other,[2] and anything between them is treated by LaTeX as a single entity. Placing them around some plain text, like `{text}`, will have no effect on the output (but this can be useful sometimes).

A **command** can be described as anything in the source code which LaTeX interprets in a special way. There are three main types of command, and we have seen examples of each already:

- A special character, eg `$`
- A `\` followed immediately by a single non-number, eg `\#`
- A `\` followed immediately by a string of letters, eg `\begin`

Note that command names can never contain numbers, so even if we tried to define one called `\ps2pdf`, say, LaTeX would complain.

Some commands just produce output, for example `\today` is a shortcut for today's date. Some are applied to **arguments**, for example `\begin{document}`. Here are some rough guidelines for arguments:

- mandatory arguments follow the command name and are placed between `{` and `}`
- if there is more than one mandatory argument they are listed successively in this way, as in `\`*command*`{`*arg1*`}{`*arg2*`}{`*arg3*`}`...
- optional arguments are placed between `[` and `]`
- if there is more than one optional argument, *usually* they are all listed within one set of `[ ]` and separated by commas, as in `\`*command*`[`*opt1*`,`*opt2*`,`*opt3*`...]`
- if there are mandatory *and* optional arguments, the optional ones *usually* go first, as in `\`*command*`[`*opt*`]{`*arg*`}`

There are a few exceptions to these rules — you may be able to spot one or two in Table 1.3 below.

The `\begin` command always takes at least one mandatory argument, *envname* say, and must always be paired with a corresponding `\end{`*envname*`}` command. The area between `\begin{`*envname*`}` and `\end{`*envname*`}` is called an *envname* **environment**, and this is usually treated as a separate paragraph. One environment can be *nested* inside another, as long as their respective `\end` commands are placed in reverse order.

Like commands, environments can take extra mandatory and optional arguments; the guidelines above apply in a similar way (replace `\`*command* with `\begin{`*envname*`}`). Table 1.3 gives some common examples of commands and environments with mandatory and optional arguments.

Finally there is one other type of command: a **declaration** (usually) takes no arguments but affects *all* of the source code which follows it — at least until we tell it to

no arguments	`\today` `\begin{document} ... \end{document}`
0 mandatory, 1 optional	`\item[1.]` `\begin{figure}[ht!] ... \end{figure}`
1 mandatory, 0 optional	`\author{Chris Wetherell}` `\begin{array}{rl} ... \end{array}`
1 mandatory, 1 optional	`\sqrt[3]{64}` `\begin{minipage}[b]{10cm} ... \end{minipage}`
1 mandatory, 2 optional	`\documentclass[12pt,a4paper]{book}`
2 mandatory, 0 optional	`\setlength{\textwidth}{5pt}` `\begin{list}{\dag}{\setlength{\parsep}{0pt}} ... \end{list}`
2 mandatory, 1 optional	`\parbox[t]{1in}{hello\\there}` `\begin{tabular*}{9ex}[t]{|c|} ... \end{tabular*}`
2 mandatory, 2 optional	`\raisebox{.4ex}[1.5ex][.75ex]{text}`

Table 1.3: Mandatory and optional arguments

stop. The extent of the affected text is called the **scope** of the declaration. The most common way of defining the scope is to place the declaration and text between `{` and `}`, as in

```
{\large This is large text}, and this is normal size.
```

Note also that the `\end{`*envname*`}` command usually ends the scope, if necessary, of a declaration inside the *envname* environment.

Often there is a command and a declaration (or a declaration and an environment) which produce the same output. For example, to achieve **bold text** we could either use the command `\textbf{bold text}` or the declaration `{\bfseries bold text}`. In this case there is no real advantage in doing it one way over the other — so why have the choice? The short answer is that commands with arguments work best for short passages of text, and environments are best for long passages of text — declarations sit somewhere in the middle to take up the slack.

Also note that many commands and environments have a "starred" variation:

`\`*command*`*`

`\begin{`*envname*`*} ... \end{`*envname*`*}`

The starred variation often suppresses the number which would normally be associated with the un-starred one.

We have already met the most important command and environment in LaTeX: `\documentclass` and `\begin{document}` ... `\end{document}`. **Every LaTeX document must contain these**, as in the example on page 1·5.

The area between `\documentclass` and `\begin{document}` is called the **preamble**. This is where we can give "behind the scenes" instructions about the *overall* look of the output. For example, extra packages must be imported in the preamble, and new commands are usually defined there (although it is sometimes useful to do this later on also).

The area between `\begin{document}` and `\end{document}` is called the **body**, and this is where the input text goes. It will also contain typesetting commands that control the *local* look of the output, for example if some text is to appear in italics or a different font size.

Any input text (other than comments) above `\documentclass` will produce an error, and any text below `\end{document}` will be ignored.

The `\documentclass` command takes one mandatory argument, and many optional arguments — see page 1·5 and Table 1.3. The mandatory argument chooses the class of document and this choice influences how all other commands are interpreted. The most common examples are `article`, `book` and `report`, although we will also meet `letter` and `slides` later in the course. It is important to note that some commands, declarations and environments are only available in certain document classes. For example the command for starting a new chapter, `\chapter`, is not available in the `article` class.

The optional arguments for `\documentclass` override some of the default settings for a given document class. For example, a `book` document would normally have a standard font size of 10pt, have left and right margins set for double-sided printing, and contain just one column of text. We could instead specify

```
\documentclass[12pt,oneside,twocolumn]{book}
```

to override these settings, if we were interested in typesetting a newsletter, say.

We will visit this topic again later, but for the moment we will be happy to let LaTeX keep its default settings.

Sectional units

In LaTeX it is easy to break the output into chapters, sections and so on. The commands in Table 1.4 define successively smaller sectional units. They each take one mandatory argument, namely the heading of that sectional unit.

`\part`	`\chapter`	`\section`	`\subsection`
`\subsubsection`	`\paragraph`	`\subparagraph`	

Table 1.4: Sectioning commands

In the `book` and `report` document classes, `\part` and `\chapter` both start a new page. In the `article` class, `\part` does *not* start a new page and `\chapter` is not available.

Numbering of sectional units is handled automatically. A sectioning command can behave a little like a declaration, in that it can influence the numbering of things which follow it in the source code. The scope in this case is delimited by the next sectioning command of equal or higher rank. Here is an example of what to expect:

```
\chapter{A chapter}

In this chapter...


\section{A section}

Now we look at...


\subsection{A subsection}

On the other hand...


\subsubsection{A subsubsection}

Finally...
```

Chapter 4

A chapter

In this chapter...

4.1 A section

Now we look at...

4.1.1 A subsection

On the other hand...

A subsubsection

Finally...

To suppress numbering of a sectional unit use the starred variation. The headings in this document are all defined by `\section*{`*heading*`}` commands.

Titlepages and abstracts

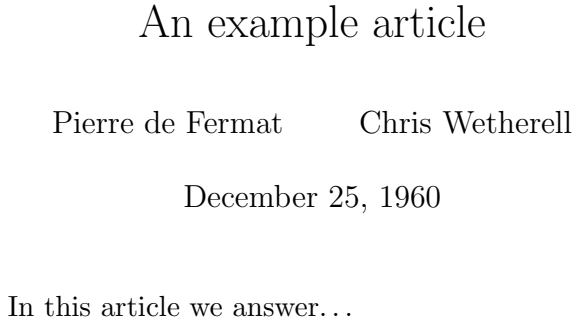
A `titlepage` environment near the top of the document can be used, not surprisingly, to typeset a titlepage. Its contents will appear alone on an unnumbered page, but you are entirely responsible for visual formatting (we will see how to do this in the next few sections, where we look at ways of customising spacing, justification and font sizes).

Alternatively, LaTeX can do all the typesetting for you with the `\maketitle` command. It relies on three additional commands, `\title`, `\author` and `\date`, each of which takes one mandatory argument. The argument for `\author` can contain one name, or a list of names separated by an `\and` command — this helps with spacing but does not produce "and" in the output. If the `\date` command is missing, LaTeX displays today's date, equivalent to typing `\date{\today}`. To suppress the date, use `\date{}`.

The exact layout prescribed by `\maketitle` depends on the document class. In an `article` this information appears at the top of the first page, and the text of the document starts directly below:

```
\title{An example article}
\author{Pierre de Fermat
     \and Chris Wetherell}
\date{December 25, 1960}
\maketitle

In this article we
   answer...
```



In the `book` and `report` classes, the titlepage information appears on a separate unnumbered page. These defaults can be reversed with optional arguments:

```
\documentclass[titlepage]{article}
\documentclass[notitlepage]{book}
\documentclass[notitlepage]{report}
```

An abstract can be provided in the `article` or `report` classes with the `abstract` environment (it's not available in `book`, so you might use `\chapter*{Abstract}`). In `article` the abstract is typeset in `\small` font size and appears wherever you put it in the source code — usually just after the `\maketitle` command:

```
\begin{abstract}
During this course we
    will...
\end{abstract}
```

Abstract

During this course we will...

In `report` the abstract appears on a page by itself. These defaults are also reversed by the appropriate `titlepage` or `notitlepage` argument.

Spacing

Throughout the rest of the course we will occasionally use the symbol ␣ to denote a single white space in the source code.

LaTeX is very good at deciding how to manage the spacing of a document, whether it's inter-word spacing on a line, inter-paragraph spacing on a page, or inter-section spacing within the whole document. It's not infallable though, so from time to time you'll need to do some fine-tuning to get the output just right.

> **Note: spacing throughout the *whole* document may change if you add or remove even a small amount of text, so don't do any manual fine-tuning until you're absolutely sure the content is final.**

This said, there are a number of tricks you should get used to using that will help LaTeX make the right decisions.

LaTeX puts a little more space between sentences than it does between words, so you don't need to do this yourself in the source code — multiple spaces are ignored anyway, so it wouldn't make any difference if you tried. For typographical purposes it interprets the strings '.␣', '?␣', '!␣' and ':␣' as the end of a sentence, *unless* they were immediately preceded by a capital letter. This way initials like C. Wetherell are handled correctly. However, occasionally you will want to end a sentence with a capital letter, like C. Or use abbreviations like Prof. in the middle of a sentence, without adding the extra space. To achieve these you would type

```
...letter,␣like␣C\@.␣Or␣use␣abbreviations␣like␣Prof.\␣in␣the...
```

The `\@` command followed by a punctuation character tells LaTeX explicitly that the sentence has ended. The `\␣` command inserts a normal inter-word space, so in particular it may be used repeatedly to insert extra space:

```
...may␣be␣used\␣\␣\␣\␣repeatedly...
```

Another useful space command is `\,` which inserts a small space, roughly half the width of the inter-word space `\␣`.

Non-breaking spaces are achieved with the `~` special character. It has the same size as a regular inter-word space, but LaTeX will not break a line at that point. This is especially useful for names and labels, like `Dr~C.~Wetherell` or `Equation~(1)`, so this is a good habit to get into even if you don't think the text will be near the end of a line.

A blank line or `\par` command in the source code tells LaTeX to start a new paragraph. Multiple blank lines or `\par`'s do *not* produce blank lines in the output, however.

A new paragraph may or may not be indented automatically, depending on the document class. The `\indent` and `\noindent` commands at the beginning of a paragraph will override the default in the obvious way. They will be ignored if LaTeX was already going to do what you wanted.

By default LaTeX justifies text on both left and right margins. Table 1.5 gives the declarations and equivalent environments which justify the output differently (note the spelling of "center").

left justified	`\raggedright`	`\begin{flushleft}` ... `\end{flushleft}`
right justified	`\raggedleft`	`\begin{flushright}` ... `\end{flushright}`
centered	`\centering`	`\begin{center}` ... `\end{center}`

Table 1.5: Justification commands

In addition to starting a new paragraph with `\noindent` , there are three commands that can be used to start a new line without indenting. The `\linebreak` command tells LaTeX to stretch the inter-word spacing so that the text is left-right justified. For example

```
This is stretched.\linebreak A new line here.
```

produces

This is stretched.
A new line here.

On the other hand, `\newline` and `\\` both start a new line *without* stretching the previous one. In combination with a blank line in the source code, we can now produce a blank line in the output:

```
This is followed by a blank line.\\

This follows a blank line.
```

LaTeX will give an error if you try to use `\newline` or `\\` by itself after a blank line.

Similarly there are commands for manually starting a new page, namely `\pagebreak` and `\newpage`. The first stretches the inter-paragraph spacing so that the preceding text fills out a full page; the latter does not.

Finally there are a number of useful commands which allow you to insert horizontal and vertical space of whatever length you wish. The commands `\hspace` and `\vspace` take one mandatory argument, namely the desired length (which may be negative). There are seven basic units of measurement:

`mm`	a millimetre	`pc`	1 pc = 12 pt
`cm`	1 cm = 10 mm	`ex`	the height of the letter 'x' in the current font
`in`	1 in = 2.54 cm		
`pt`	1 in = 72.27 pt	`em`	the width of the letter 'M' in the current font

Thus `space\hspace{0.5in}here` produces "space here" as we would hope. The `\vspace` command is usually used between paragraphs; if not then the extra space is added *after* the line in which the command appears. The commands `\smallskip`, `\medskip` and `\bigskip` insert vertical space of a predefined length.

There is an extremely useful length called `\fill`. When used as the argument of `\hspace` (or `\vspace`), LaTeX will insert *all* of the space which is available on the current line (or page). When two or more such commands appear on the one line (or page) the available space is shared evenly among them. Thus

```
This\hspace{\fill}\hspace{\fill}is\hspace{\fill}stretched.
```

produces

This is stretched.

The command `\hfill` is a shortcut for `\hspace{\fill}`, and `\vfill` is a shortcut for `\vspace{\fill}`

LaTeX will ignore any excess space that `\hspace` or `\vspace` try to insert over a linebreak or pagebreak respectively. The starred forms `\hspace*` and `\vspace*` are *never* ignored. For example, adding `\hfill` at the end of a line will have no effect, so instead you would use `\hspace*{\fill}`.

Accents, special symbols and the like

The quotation marks ‘ and ’ work just how you would expect, but in LaTeX we very rarely use the " character. Instead we use ‘‘ and ’’. To differentiate between single and double quotes, for example in writing

> “‘Diet Coke’ is an oxymoron,” she said.

we would use the `\,` command to insert a small space: `‘‘\,‘Diet...`

Table 1.6 shows how to create accents in LaTeX. The special commands `\i` and `\j` remove the dot from `i` and `j` respectively, specifically so that these letters can also be accented: `\‘{\i}` produces ì.

è	`\‘{e}`	ẽ	`\~{e}`	ě	`\v{e}`	ę	`\c{e}`
é	`\’{e}`	ē	`\={e}`	ő	`\H{e}`	ẹ	`\d{e}`
ê	`\^{e}`	ė	`\.{e}`	e͡e	`\t{ee}`	e̲	`\b{e}`
ë	`\"{e}`	ĕ	`\u{e}`				

Table 1.6: Accents in LaTeX

Table 1.7 lists a number of special symbols, including a reminder of how to typeset the ten special characters. LaTeX will ignore any white space that immediately follows a command whose name ends in a letter. For example `\LaTeX␣will␣ignore...` produces “LaTeXwill ignore...”, thus an extra space must be added explicitly with `\␣` if desired.

%	`\%`	~	`\~{}`	{	`\{`	}	`\}`
\	`$\backslash$`	#	`\#`	&	`\&`	$	`\$`
^	`\^{}`	_	`\_`	œ	`\oe`	Œ	`\OE`
æ	`\ae`	Æ	`\AE`	å	`\aa`	Å	`\AA`
ø	`\o`	Ø	`\O`	ł	`\l`	Ł	`\L`
ß	`\ss`	¿	`?‘`	¡	`!‘`	...	`\ldots`
†	`\dag`	‡	`\ddag`	§	`\S`	¶	`\P`
©	`\copyright`	£	`\pounds`	TeX	`\TeX`	LaTeX	`\LaTeX`

Table 1.7: Special symbols

Playing with fonts

In LaTeX there are many ways to modify the look of output text.

Font styles are governed by three attributes: *family*, *shape* and *series*. Table 1.8 shows how to specify each of these as well as how to return to the default settings (which are determined by the document class). Each choice can be made either by a command with argument or by a declaration, and they can be made in any combination (LaTeX may not always be able to create the intended font, but it will try to choose something which *it* thinks is reasonably close).

Roman	`\textrm{Roman}`	`{\rmfamily Roman}`
Typewriter	`\texttt{Typewriter}`	`{\ttfamily Typewriter}`
Sans Serif	`\textsf{Sans Serif}`	`{\sffamily Sans Serif}`
Upright	`\textup{Upright}`	`{\upshape Upright}`
Italic	`\textit{Italic}`	`{\itshape Italic}`
Slant	`\textsl{Slant}`	`{\slshape Slant}`
Small Caps	`\textsc{Small Caps}`	`{\scshape Small Caps}`
Medium	`\textmd{Medium}`	`{\mdseries Medium}`
Boldface	`\textbf{Boldface}`	`{\bfseries Boldface}`
Normal Font	`\textnormal{Normal Font}`	`{\normalfont Normal Font}`

Table 1.8: Changing font style

Note that the scope of a font declaration will only be delimited by a later font declaration which alters the *same* font attribute. For example

```
\slshape This is Slant, \bfseries this is Bold Slant,
\upshape and this is Bold Upright.
```

produces

This is Slant, ***this is Bold Slant*****, and this is Bold Upright.**

The scope of `\slshape` ends at `\upshape`, but the scope of `\bfseries` does not.

Two slightly different ways of changing the style of fonts are with *emphasised text* and `verbatim text` — at a glance they look just like italics and typewriter font, but they behave a little differently.

Emphasising is achieved with the `\emph` command or `\em` declaration, for example `\emph{emphasised}` or `{\em emphasised}`. How LaTeX actually interprets this is determined by the document class — it just happens that in the `book` class, which is used for these notes, they are interpreted as `\textit` and `\itshape` respectively.

On the other hand the `\verb` command and `verbatim` environment are always typeset in typewriter font, but in each case the formatting of output is determined *exactly* by the formatting of the source code. This means in particular that all special characters are treated as normal characters, and extra spaces in the source code also appear in the output. This is particularly useful for displaying things like computer programs, and indeed this is how all the sample code is achieved in these notes: the example following Table 1.8 was created with

```
\begin{verbatim}
\slshape This is Slant, \bfseries this is Bold Slant,
\upshape and this is Bold Upright.
\end{verbatim}
```

The only text following the statement `\begin{verbatim}` which *is* interpreted in a special way is the 14 character string `\end{verbatim}` which ends the environment.

The command `\verb` is applied to one mandatory argument as we would expect — the difference here is that the argument is *not* delimited by `{` and `}`, but instead by a pair of identical non-letters which do not appear in the argument. For example, to typeset `\end{verbatim}`, which cannot appear inside a `verbatim` environment, we would use something like

```
\verb+\end{verbatim}+
```

I tend to use a pair of `+`'s as delimiters, but a pair of `2`'s or a pair of `@`'s would be just as good, as long as `2` or `@` didn't appear in the argument.

Note that `\verb+`*arg*`+` *must* appear on a single line of the source code, but it *cannot* appear in the argument of another command.

Only the non-letter `*` cannot be used as the delimiter for `\verb`, because of the starred variation. The command `\verb*` and environment `verbatim*` work in exactly the same way as their un-starred counterparts, except that white spaces are replaced by the symbol ␣.

In LaTeX the standard font size is set by default, or with an optional argument to `\documentclass`. These notes have `11pt` as the standard — it's the size of the "normal" text in paragraphs. LaTeX can create text of an arbitrary size (we will see how later in the course), but for most purposes it suffices to use one of ten predefined font sizes: the standard itself, four which are smaller, and five which are larger. These are produced with declarations, as outlined in Table 1.9. The scope of one such declaration will be delimited, if necessary, by another which follows it. Note also that the various font sizes can all be used in combination with the commands and declarations in Table 1.8.

`{\tiny Sample text}`	Sample text
`{\scriptsize Sample text}`	Sample text
`{\footnotesize Sample text}`	Sample text
`{\small Sample text}`	Sample text
`{\normalsize Sample text}`	Sample text
`{\large Sample text}`	Sample text
`{\Large Sample text}`	Sample text
`{\LARGE Sample text}`	Sample text
`{\huge Sample text}`	Sample text
`{\Huge Sample text}`	Sample text

Table 1.9: Changing font size

Hyphens

There are three hyphens you can use in LaTeX:

- `-` intra-word hyphen, as in "up-to-date"
- `--` for ranges, as in "20–53" or "A–Z"
- `---` for interjections — like so!

You never need to use `-` to break a long word at the end of a line, though. LaTeX uses an extensive dictionary to do this automatically if it thinks it necessary. If it doesn't recognise a word it will try to guess the most logical place for a hyphen, but you can override this choice at any time with the `\-`, `\mbox` and `\hyphenation` commands.

The `\-` command typed within a word does *not* produce a hyphen, but tells LaTeX where in the word it is allowed to break over a line if needed. Therefore typing `hyp\-hena\-tion` will, depending on whether it is to be broken over a line, only produce one of "hyphenation", "hyp- henation" or "hyphena- tion". Occasionally you will want to suppress LaTeX's urge to hyphenate. You can do this with the `\mbox` command, although the output may look the worse for it: typing `\mbox{hyphenation}` forces "hyphenation" to hang over the end of the line (actually, I had to trick LaTeX into doing that with non-breaking spaces too).

These options are fine if you use the offending word once or twice, but any more often and it's better to use the `\hyphenation` command in the preamble:

```
\hyphenation{hyp-hena-tion,Weth-er-ell,Fermat}
```

Now whenever we type `Wetherell`, LaTeX will know that it can only produce one of "Wetherell", "Weth- erell" or "Wether- ell". It now also knows that "Fermat" can never be broken over a line.

Note that these commands can be applied to *any* words we like, even if LaTeX thought it knew how to deal with them already.

What to do when things go wrong

The running commentary LaTeX produces while compiling contains a swathe of technical information — have a close look at a `.log` file sometime. To the user the most important messages are **warnings** and **errors**.

Warnings are just that: LaTeX will continue compiling but it thinks something is slightly amiss. Probably the most common warnings are of the form

```
Overfull \hbox (1.38033pt too wide) in paragraph at lines 270--270

Underfull \hbox (badness 10000) in paragraph at lines 459--462
```

The first tells you that text in the output, corresponding to line 270 of the source file, has run past the right margin by 1.38033 pt. The second is just the opposite: LaTeX thinks

that the inter-word spacing has been stretched too much while trying to accommodate left-right justification. In these situations you might need to fine-tune the spacing near that text, but remember that you should only do this to the very final version of your document!

Other common warnings occur when changes are made to cross-referencing and numbering — refer to Figure 1.2 and the comments nearby on page 1·3.

Errors are more serious. The compilation will cease and LaTeX will await your instructions. The ones to know are:

`x`	quit LaTeX
`h`	give more information about this error
`Enter`	try to ignore this error and keep compiling
`s`	scrollmode: like pressing `Enter` whenever an error occurs

Sometimes LaTeX will prompt you to correct the error as you go, but this will *not* correct the source file. I suggest you quit and make the correction in the file instead.

Many errors are simple misspellings of command names or environments:

```
! Undefined control sequence. l.1147 \chatper
                {Introduction}
?
```

Occasionally we `\end` nested environments in the wrong order, or forget to `\end` them altogether:

```
! LaTeX Error:
\begin{titlepage} on input line 21 ended by \end{document}.

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

l.1160 \end{document}

?
```

And sometimes we forget a `}` delimiter:

```
Runaway argument? { \par \par \noindent And sometimes we \verb +\end
+ nested en\ETC. ! File ended while scanning use of \textbf .
<inserted text>
                \par
<*> module1.tex

?
```

The moral of the story: LaTeX can at times be a very perplexing package, but spending just a little time trying to understand the warning and error messages will help you get the document you want much more efficiently.