Module 2

Layout – lists, tables, figures, mathematics

In the first module we covered the basics: how to create a document with a title and an abstract, chapters and sections, some fine-tuned spacing, and with a few fancy fonts thrown in.

Today you will be introduced to ways of producing output with more sophisticated layout: lists, tabstops, tables, figures, mathematics and quotations.

Throughout these notes you will find exercises that are designed to make you more familiar with new material. Some will be easy, some will be a little more challenging. There are solutions at the end of the notes, but I encourage you to give all of the exercises a good try before consulting them. (I recommend you use the `book` document class with no optional arguments for these exercises, because different settings may give you slightly different output.)

The three basic lists

In LaTeX there are a number of ways of creating lists. The three most common are the `enumerate`, `itemize` and `description` environments, and they each have the following syntax:

```
\begin{listname}
  \item[opt] text
  \item[opt] text
        ⋮
  \item[opt] text
\end{listname}
```

Here *listname* is one of `enumerate`, `itemize` or `description`, each `\item` command takes an optional argument *opt* which overrides the default label for that item, and the *text* which follows can contain just about anything you like, including blank lines (new paragraphs) and other nested environments.

The `enumerate` environment automatically numbers the items (unless an optional argument is given); the `itemize` command bullets the items (unless an optional argument is given); for the `description` environment there is no default label, but if an optional argument *opt* is given for `\item` then it is typeset in bold font and the *text* is indented. Table 2.1 gives the basic layout of each — they were all created with

```
\begin{listname}
  \item First item.
  \item[Hello] Second item, with \verb+Hello+ as optional argument.
  \item Third item.
\end{listname}
```

`enumerate`		`itemize`		`description`
1.	First.	•	First item.	First item.
Hello	Second item, with `Hello` as optional argument.	Hello	Second item, with `Hello` as optional argument.	**Hello** Second item, with `Hello` as optional argument.
2.	Third item.	•	Third item.	Third item.

Table 2.1: Basic lists in LaTeX

Exercise 2.1. Typeset the following list using the `description` environment (remember that labels are automatically typeset in bold, and recall the commands `\TeX` and `\LaTeX`):

TeX a complicated typesetting system;

LaTeX a smooth, easy-to-use, completely intuitive typesetting system based on TeX that does everything you could ever want with little or no effort;

Word admittedly quite nice in its own way.

One list environment can be nested inside another, by appearing in the *text* following an `\item`. Nested `enumerate` environments are distinguished by four levels of numbering:

1, 2, 3, . . . a, b, c, . . . i, ii, iii, . . . A, B, C, . . .

Nested `itemize` environments are distinguished by four levels of bulleting:

• – $*$ ·

There can be at most four nested `enumerate` and `itemize` environments each. Nested `description` environments are distinguished by increasing indentations. You can also nest any combination of list environments, but overall there can be at most six levels of nesting.

Exercise 2.2. Typeset the following list. You should be able to do it using *only* the `itemize` and `enumerate` environments, and *without* optional arguments for the `\item` command.

1. First point, with
    - A dot point;
    - Another dotpoint, although

        (a) i. This

          – says

          – not much

         ii. so there

         iii. A. point taken

          B. indeed

        (b) not very interesting

        (c) nor insightful

         – Some more dots

          ∗ with subdots

           · and a subsubdot

           · or two

          ∗ which really do

         – say very little
    - Back to this one.
2. And the final point.

In Module 3 we will discuss how to modify the labelling of `enumerate`, `itemize` and `description` environments. There are two other list making environments: `list`, which we will meet in Module 4, and `trivlist`. The first takes two mandatory arguments which allow you to completely customise the labelling and spacing of the list; the second is mostly used for formatting in the definition of new commands and environments.

Tabstops and tables

Often you will want to line your text up in columns. There are two main ways to do this in LaTeX: with the `tabbing` environment which emulates a typewriter's tabstops, and the `tabular` environment which produces tables.

In the `tabbing` environment you are responsible for ending each line, except the last one, with `\\` (otherwise the output might go beyond the right margin). Tabstops are set at any point on a line with `\=`, and you advance to the next tab stop (which was set in a previous line) with `\>`. You can add new tabstops or override previous ones at any time.

```
\begin{tabbing}
 If \= raining\\
    \> then \= frown\\
    \>      \> get wet\\
    \> otherwise \= smile\\
    \>           \> stay dry
\end{tabbing}
```

```
If raining
   then frown
          get wet
   otherwise smile
                stay dry
```

Notice how the second tabstop which was set in line 2 is overridden by the new tabstop in line 4. Some other things to note:

- all white space after a `\=` or `\>` is ignored, but a single white space before a `\=` will affect the position of the tabstop;
- the number of `\>`'s in a line may be less than but cannot exceed the number of distinct[1] `\=`'s defined in previous lines;
- `\>` advances to the next *unused* tabstop, so this might actually be on the *left* of preceding text;
- The commands `\=`, `\'` and `` \` `` all have special meanings in a `tabbing` environment, so they cannot be used for accents; instead use `\a=`, `\a'` and `` \a` ``;
- a `\kill` command at the end of a line (instead of `\\`) can be used to define tabstops without producing any output;
- unlike list environments, you *cannot* nest one `tabbing` environment inside another.

The next example illustrates some of these points:

```
\begin{tabbing}
 My fianc\a'{e} says I\= shouldn't abuse\\
 tabstops like I seem \>to have here\\
 No tabstops here\\
 One \=Two \=Three \=Four \kill
 1   \>2   \>3     \>4\\
 One \>Two \>Three \>Four
\end{tabbing}
```

```
My fiancé says Ishouldn't abuse
tabstops like I seemto have here
No tabstops here
1     2     3       4
One Two Three Four
```

There are a number of other features available in a `tabbing` environment — see Lamport's *LaTeX: A Document Preparation System.*

Exercise 2.3. Typeset the following using the `tabbing` environment (you should be able to do it with at most two tabstops in each line):

```
On Sunday I went to the market,
             and bought a pig.
It never occurred to me
                   that I should instead
have bought              a cow.
```

[1]by "distinct" I mean tabstops that haven't (yet) been overridden by a later tabstop

The `tabular` environment takes one mandatory argument which is a string of characters that governs the layout of the table — I'll call this the **layout argument**. The simplest layout argument consists of the letters `l`, `c` and `r` which tell LaTeX that the corresponding column will be left-, centre- or right-justified respectively. Thus `\begin{tabular}{llr}` says that there are three columns, the first two are left-justified and the third is right-justified. Each row is a list of entries separated by an `&` character and (with the possible exception of the last row) ended with a `\\` command:

```
\begin{tabular}{rcl}
 one   & two   & three \\
 four  & five  & six \\
 seven & eight & nine
\end{tabular}
```

one	two	three
four	five	six
seven	eight	nine

The spacing of columns and rows is set by default, but we will see in Module 4 how to alter these.

The `\multicolumn` command is used to stretch an entry over more than one column, or to override for a single entry the justification that was set in the layout argument. It has three mandatory arguments: the first is the number of columns the entry will take up; the second is a single letter `l`, `c` or `r` which specifies how to typeset the entry; the third is the entry itself:

```
\begin{tabular}{rcl}
 one   & \multicolumn{2}{r}{twothree} \\
 \multicolumn{3}{c}{fourfivesix} \\
 seven & eight & nine
\end{tabular}
```

one	twothree	
fourfivesix		
seven	eight	nine

Care must always be taken that the number of columns expected in a row by `&` and `\multicolumn` commands does not exceed the number specified in the layout argument (it may be less, though).

Vertical lines are specified with the `|` character in the layout argument:

```
 \begin{tabular}{||rc|l}
 one   & two   & three \\
 four  & five  & six \\
 seven & eight & nine
\end{tabular}
```

one	two	three
four	five	six
seven	eight	nine

The `|` character can also appear in the second argument of a `\multicolumn` command; this can be used to create partial vertical lines either by adding a new vertical line to a row, or overriding an existing line from the layout argument:

```
\begin{tabular}{||rc|l}
 one   & two   & three \\
 four  & five  & \multicolumn{1}{l|}{six} \\
 seven & \multicolumn{1}{c}{eight} & nine
\end{tabular}
```

one	two	three
four	five	six
seven	eight	nine

(Sometimes LaTeX won't do quite what you expect because some lines take precedence over others; you may need to do a bit of fiddling around with this.)

Horizontal lines are produced with the `\hline` command; it should appear at the beginning of the row above which the line is required. Double lines are produced with two `\hline`'s and so on. To draw lines at the bottom of the table, end the last row with a `\\` and put the required number of `\hline`'s on the next line by themselves:

```
\begin{tabular}{rcl}
 \hline       one   & two   & three \\
 \hline\hline four  & five  & six \\
              seven & eight & nine \\
 \hline
\end{tabular}
```

one	two	three
four	five	six
seven	eight	nine

Partial horizontal lines are produced in a similar way with the `\cline` command. It takes one mandatory argument which is a range *i*-*j* which tells LaTeX to put the line only above columns *i* to *j*:

```
\begin{tabular}{rcl}
 \cline{2-3}  one   & two   & three \\
 \hline\hline four  & five  & six \\
              seven & eight & nine \\
 \cline{1-1}\cline{3-3}
\end{tabular}
```

one	two	three
four	five	six
seven	eight	nine

You should now be able to. . .

Exercise 2.4. Typeset the following using the `tabular` environment (recall the command `\$` for $):

Item	Price	
	Aus $	US $
TeX	0.00	0.00
LaTeX	0.00	0.00
Word	174.95	89.99

Finally, the `@` special character is a very useful command which can be used in the layout argument. It takes one mandatory argument which is a piece of text that will be printed in the corresponding position in every row. The default space between columns is removed from either side of a `@` command:

```
\begin{tabular}{rc@{\$}l@{huh?}}
 one   & two   & three \\
 four  & five  & six \\
 seven & eight & nine
\end{tabular}
```

```
  one    two $threehuh?
 four   five $six   huh?
seven  eight$nine  huh?
```

This can be overridden for an individual row with the `\multicolumn` command. The next exercise gives one example of how useful the `@` command can be.

Exercise 2.5. Use `@{.}` in the layout argument to typeset the following table:

Name	Value
one hundred	100.000
pi	3.14159
cos(2)	–0.416147
inches per metre	39.3701

Importing pictures

There are a number of ways of creating *simple* pictures in LaTeX, for example with the `picture` environment or the `xypic` package. The flow diagrams on pages 1·2 and 1·3 of the Module 1 notes were created with the `picture` environment. Further information on these can be found in *LaTeX: A Document Preparation System* or *The LaTeX Companion*.

However, for complex diagrams and images you are better off creating them in another application and then importing them into your document. The types of files that will be supported depends on the individual installation of the LaTeX program you are using. In my experience you're safest using `.eps` (encapsulated postscript) files, or possibly `.bmp` (bitmap) files, although the latter can be a bit temperamental. Some installations will accept some of the other usual suspects: `.jpg`, `.gif`, `.tif`, ...

The first step is to load the `graphics` package: type `\usepackage{graphics}` in the preamble.

Next use the `\includegraphics` command where you would like the picture inserted. This command takes one mandatory argument, the name of the image file, and two (four?) optional arguments which specify the dimensions:

```
\includegraphics[88mm,242mm][125mm,266mm]{supernova.eps}
```

This tells LaTeX to import the file `supernova.eps`, but only leave enough room to include the rectangle defined by the co-ordinates given: the bottom left corner of the rectangle is 88mm in from the left and 242mm up from the bottom of the picture's normal border, and the top right corner is 125mm in from the left and 266mm up — see Figure 2.1 (in this case `supernova.eps` is the size of an A4 page).

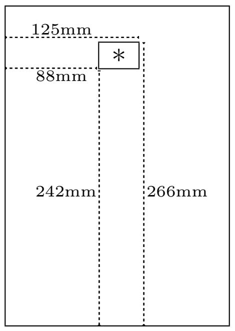

Figure 2.1: Specified dimensions of `supernova.eps`

With the `\includegraphics` command the entire picture will still be printed, but only enough room for the specified rectangle will be allowed for; the starred variation `\includegraphics*` only outputs what's inside the specified rectangle.

Note that LaTeX may complain that it can't work out the dimensions of an `.eps` picture if the optional argument is left out, but it can usually handle a `.bmp` okay in this respect. Dimensions can be specified by using any of the units `mm`, `cm`, `in`, `pt` or `pc`.

Most DVI viewers will *not* be able to show imported pictures, so you will need to convert your file to a postscript file, and then maybe even to a PDF file — refer to Table 1.1 from Module 1.

The `graphics` package has an optional argument `draft`. This tells LaTeX not to import any picture files, but instead draw a rectangular box of the required dimensions which contains the name of that file. This is especially useful if you want to use DVI viewers (which are generally much more efficient than postscript or PDF viewers) but would still like to see the exact position of pictures. It will also save compilation time if there are many pictures in your document. To use this feature type `\usepackage[draft]{graphics}` in the preamble.

Exercise 2.6. Download the encapsulated postscript file `supernova.eps` from

```
http://wwwmaths.anu.edu.au/~chrisw/LaTeX
```

and then try to include it in your file. See what happens when you change dimensions or use the `draft` option.

Displaying tables and figures

Here we will discuss two rather unfortunately named environments: `table` and `figure`. Warning: they do *not* produce tables and figures — we've already seen ways of doing this above. Instead they

- create a **floating object** which LaTeX will try to position as best it can;
- allow you to give a numbered caption to an actual table or figure with the `\caption` command.

What you put inside the `table` and `figure` environments is completely up to you. However the caption (if provided) will be labelled "Table #:" or "Figure #:" respectively, so you would usually use them for actual tables or figures. Because the caption is centered it is usual to begin these environments with the `\centering` declaration:

```
\begin{figure}
 \centering
 \includegraphics[88mm,242mm]
   [125mm,266mm]{supernova.eps}
 \caption{A supernova}
\end{figure}
```

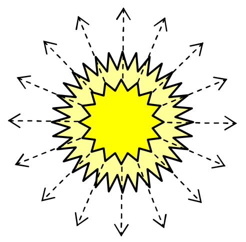
Figure 2.2: A supernova

Note that tables and figures are numbered independently. The `\caption` command can contain any text or symbols you like; for long captions you may need to use `\\` to break a line.

LaTeX will try to position floating objects to give the neatest looking output — the main criterion is to avoid large vertical spaces in a page. This means that a table or figure may not appear exactly where you expect, but it will *not* appear on a page earlier than the corresponding position in the source code.

There is an optional argument to both environments which allows you to give LaTeX some hints about where the floating object is allowed to go. The argument is a string containing one or more of the following characters:

`h`	*here*, where it appears in the source code;
`b`	at the *bottom* of a page;
`t`	at the *top* of a page;
`p`	on a *page* containing only floating objects;
`!`	try even harder to put it where I want.

The most common option is `\begin{table}[ht!]` or `\begin{figure}[ht!]`. There is a complicated set of rules about which choices have precedence when more than one is given, and there are ways to gain even more control over positioning, but *mostly* the position LaTeX chooses for you will be satisfactory.

Exercise 2.7. Using the `table` environment, centre the table in Exercise 2.4 and give it the caption "Price of typesetting packages".

A relatively brief introduction to mathematics

LaTeX typesets mathematics beautifully — afterall, this is precisely what its predecessor TeX was designed to do. The resources available are so vast that we will really only be able to scratch the surface, but hopefully you will get a feel for the kind of things that are possible.

The definitive guide to mathematical typesetting is Chapter 8 of *The LaTeX Companion*; it is available from the web at

`http://www.ctan.org/tex-archive/info/companion-rev/ch8.pdf`

In addition to the array of commands and environments designed for the specific needs of modern mathematicians, this document catalogues literally hundreds of symbols which have far broader applications.

Mathematics is entered in **maths-mode**, a special kind of source code that LaTeX interprets in a different way to normal text. The three most common ways of entering maths-mode are with the `math`, `displaymath` and `equation` environments; the first creates an **in-text formula** such as $E = mc^2$, the second creates a **displayed formula** such as

$$E = mc^2$$

and the third creates a numbered displayed formula such as

$$E = mc^2 \tag{2.1}$$

These examples are produced with

```
\begin{math} E=mc^{2} \end{math}

\begin{displaymath} E=mc^{2} \end{displaymath}
```

and

```
\begin{equation} E=mc^{2} \end{equation}
```

respectively. (There is also a starred variation `equation*` which supresses the number, so this is equivalent to `displaymath`.)

Thankfully, because the `math` and `displaymath` environments are used so often, they each have (two) convenient shortcuts as shown in Table 2.2.

`\begin{math}`	`\end{math}`	`\begin{displaymath}`	`\end{displaymath}`
`\(`	`\)`	`\[`	`\]`
`$`	`$`	`$$`	`$$`

Table 2.2: Shortcuts for maths-mode

Note that the shortcuts you use for `\begin{...}` and `\end{...}` must match each other. Thus I might type `$E=mc^{2}$` and `\[E=mc^{2}\]` respectively, but I could not use `$E=mc^{2}\)` or `\begin{displaymath}E=mc^{2}$$` for example.

Standard symbols like =, +, −, /, <, >, (,), [, and] can all be typed normally in maths-mode; the braces { and } are again produced with `\{` and `\}` respectively.

Some commands, like the special character `^` which produces superscripts, are only available in maths-mode; others like the `\rmfamily` declaration can only be used in **text-mode** (the opposite of maths-mode which we've used to date); some are available in both, like the ellipsis command `\ldots`.

Otherwise maths-mode is different from text-mode in three important respects:

- letters (which are not part of a command name) are typeset in italics;
- spacing is quite different to normal text, and in fact *all* white space in the source code is ignored;
- blank lines are not allowed.

You should *never* use maths-mode as a shortcut for italics or emphasised text, even though they use the same fonts: compare

```
\emph{\ldots and they're off!}
```
. . . and they're off!

with

```
$\ldots and they're off!$
```
$\ldots and they're off!$

Subscripts and superscripts are created with the special characters `_` and `^` respectively, and when both are needed they are applied successively (but in no particular order): $a_0 = b^2 = c_1^3$ can be produced with `$a_{0}=b^{2}=c_{1}^{3}$`.

Exercise 2.8. Typeset the in-text formula: $a^3_{i^2+1} = b_j^{4+c_k^{-1}}$

These commands can be particularly useful for adding limits to a sum (`\sum`), product (`\prod`) or integral (`\int`).

Occasionally you might need to have some plain text appearing in maths-mode, and this can be achieved with the `\mbox` command.

Exercise 2.9. Typeset the following displayed formulas:

$$\sum_{i=1}^{n} a_i x^i = \prod_{0<j<m} (1 - x^j)$$

$$\int (2x+1) dx = x^2 + x + \mbox{constant}$$

There are many commands already defined in LaTeX which produce plain looking text in maths-mode; for example `\sin`, `\cos`, `\log`, `\exp` and `\lim`. These are usually referred to as **log-like functions**. If a LaTeX command exists for such a thing then its name will be exactly what you expect, as in the examples just mentioned. On the other hand, if there is no such command then you could easily create your own with `\mbox`.

Fractions in an in-text formula are often simply written 1/2 (`$1/2$`). For displayed formulas you might instead use the `\frac` command which takes two mandatory arguments, the numerator and the denominator:

```
\[ \frac{1}{2}=\frac{a^{2}+1}{\log c} \]
```
$$\frac{1}{2}=\frac{a^{2}+1}{\log c}$$

Square roots are produced with the `\sqrt` command on one mandatory argument, and other roots are possible if an optional argument is also used:

```
$\sqrt{16}=\sqrt[3]{64}$
```
$\sqrt{16}=\sqrt[3]{64}$

`(`	$($	`)`	$)$
`[`	$[$	`]`	$]$
`\{`	$\{$	`\}`	$\}$
`\langle`	$\langle$	`\rangle`	$\rangle$
`/`	$/$	`\backslash`	$\backslash$
`|`	$\vert$	`\\|`	$\Vert$
`.`	*no output*		

Table 2.3: Common delimiters in maths-mode

As we have already seen, some objects created in maths-mode are much taller than normal characters in text-mode, therefore we can't expect the standard mathematical delimiters (,), {, } etc. to always be large enough. To overcome this use `\left` and `\right`: compare

```
( \prod_{n}a_{n} )
```

$$(\prod_{n} a_n)$$

with

```
\left( \prod_{n}a_{n} \right)
```

$$\left(\prod_{n} a_n\right)$$

A `\left` command must always be paired with a `\right`, but they needn't be applied to matching delimiters, or even to what we would normally think of as "left" and "right" delimiters. For example, something like `\left]`. . . `\right\{` would be perfectly okay — it's what's between the `\left` and `\right` which determines how big the] and { would need to be. Table 2.3 lists some common delimiters that can be used with `\left` and `\right`. Take particular note of `.` — this is used as a "dummy" delimiter in the event that only one large symbol is actually wanted:

```
\[ \int_{a}^{b} x^{n-1} =
   \left. \frac{x^{n}}{n} \right|_{a}^{b} \]
```

$$\int_{a}^{b} x^{n-1} = \left. \frac{x^{n}}{n} \right|_{a}^{b}$$

Exercise 2.10. Typeset the following displayed formula:

$$\frac{dy}{dx} = \frac{1}{\sqrt{1+\frac{1}{x^2}}} - \sqrt[4]{\sin\left(\frac{e^x}{x(1+x)}\right)+1}$$

All Greek letters are available in maths-mode; the command is simply a \ followed by the name of that letter:

`$\delta, \lambda, \mu$` δ, λ, μ

(In addition there are variations for the six letters epsilon, phi, pi, rho, sigma and theta, namely `\varepsilon`, `\varphi`, `\varpi`, `\varrho`, `\varsigma` and `\vartheta`.) Capital letters are produced by starting the letter's name with a capital, *unless* the symbol is the same as a standard Roman letter:

`$\Delta, \Lambda, M$` Δ, Λ, M

(That is, there is no `\Mu` command.)

Table 2.4 lists some other commonly used mathematical symbols, and a few non-mathematical ones. There are hundreds more — see *The LaTeX Companion.*

$\times$	`\times`	$\div$	`\div`	$\pm$	`\pm`	$\mp$	`\mp`
$\cap$	`\cap`	$\cup$	`\cup`	$\leq$	`\leq`	$\geq$	`\geq`
$\approx$	`\approx`	$\propto$	`\propto`	$\in$	`\in`	$\ni$	`\ni`
$\subset$	`\subset`	$\subseteq$	`\subseteq`	$\supset$	`\supset`	$\supseteq$	`\supseteq`
$\rightarrow$	`\rightarrow`	$\leftarrow$	`\leftarrow`	$\Rightarrow$	`\Rightarrow`	$\Leftarrow$	`\Leftarrow`
$\leftrightarrow$	`\leftrightarrow`	$\Leftrightarrow$	`\Leftrightarrow`	$\mapsto$	`\mapsto`	∞	`\infty`
ℓ	`\ell`	$\Re$	`\Re`	$\Im$	`\Im`	$\emptyset$	`\emptyset`
∇	`\nabla`	$\forall$	`\forall`	$\exists$	`\exists`	∂	`\partial`
$\ldots$	`\ldots`	$\cdots$	`\cdots`	$\vdots$	`\vdots`	$\ddots$	`\ddots`
$\star$	`\star`	$\natural$	`\natural`	$\sharp$	`\sharp`	$\flat$	`\flat`
$\heartsuit$	`\heartsuit`	$\clubsuit$	`\clubsuit`	$\diamondsuit$	`\diamondsuit`	$\spadesuit$	`\spadesuit`

Table 2.4: Some common maths-mode symbols

Any binary relation (such as $=$, $\subset$ or $\geq$) can be negated with a preceding `\not` command: `$1 \not< 0$` gives $1 \not< 0$.

Exercise 2.11. Use Table 2.4 to typeset the following displayed formula:

$$A \neq \pi r^2 \Rightarrow \exists \delta_1 \leq \delta_2 \leq \cdots : \frac{\partial A}{\partial r} \approx \prod_{i=1}^{\infty} \delta_i$$

Accents in maths-mode are *not* produced with the text-mode commands of Table 1.6, because they behave a little differently. Table 2.5 shows the available maths-mode accents. Note that the `\wide`... varieties are intended for arguments which are not a single character. The commands `\imath` and `\jmath` are the maths-mode equivalents of `\i` and `\j` — they remove the dot so accents can be added.

$\hat{x}$	`\hat{x}`	$\tilde{x}$	`\tilde{x}`	$\widehat{xy}$	`\widehat{xy}`	$\widetilde{xy}$	`\widetilde{xy}`
$\acute{x}$	`\acute{x}`	$\bar{x}$	`\bar{x}`	$\dot{x}$	`\dot{x}`	$\check{x}$	`\check{x}`
$\grave{x}$	`\grave{x}`	$\vec{x}$	`\vec{x}`	$\ddot{x}$	`\ddot{x}`	$\breve{x}$	`\breve{x}`

Table 2.5: Accents in maths-mode

Arrays and matrices are produced in maths-mode with the `array` environment. This is very similar to the text-mode `tabular` environment, *except* that all entries are typeset in maths-mode, `\multicolumn`'s are not allowed, and there are no lines (so no `|`'s in the layout argument, or `\hline`'s and `\cline`'s). For example

```
\[ \begin{array}{cr}
    a & b \\
    0 & -1
   \end{array} \]
```

$$\begin{array}{cr} a & b \\ 0 & -1 \end{array}$$

(In the unlikely event you need a table with multicolumns or lines in maths-mode, use a `tabular` environment in the argument of an `\mbox` command.)

Exercise 2.12. Use a log-like function, the `\left` and `\right` commands, symbols from Table 2.4 and an `array` environment to typeset the following displayed formula:

$$\det \left[\begin{array}{cccc} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \end{array} \right] = a_1 a_2 \cdots a_n$$

Multi-line formulas can be typeset with the `eqnarray` environment — this is also similar to the `tabular` environment, except there is no mandatory argument because there are always exactly three columns which are typeset with the default layout argument `{rcl}`:

```
\begin{eqnarray}
a & =    & b + c \\
  & \leq & d
\end{eqnarray}
```

$$\begin{array}{rcll} a & = & b+c & (2.2) \\ & \leq & d & (2.3) \end{array}$$

The starred variation `eqnarray*` suppresses all numbering; to suppress the number on an individual line, use `\nonumber` instead:

```
\begin{eqnarray}
a & =    & b + c \nonumber \\
  & \leq & d
\end{eqnarray}
```

$$\begin{array}{rcll} a & = & b+c & \\ & \leq & d & (2.4) \end{array}$$

Exercise 2.13. Typeset the following using an `eqnarray` environment, symbols from Table 2.4 and accents from Table 2.5 (but don't worry if the numbering is different in your output):

$$\begin{array}{rcll} c^2 & = & a^2 + b^2 - 2ab\cos\theta & (2.5) \\ \bar{y} & \rightarrow & e + f + g + h + i + & \\ & & j + k + l + m & (2.6) \\ \tilde{x} & \in & Y \cap Z & (2.7) \\ & \subseteq & Y & \end{array}$$

Some maths-mode objects are typeset differently by LaTeX depending on whether they appear in an in-text formula or a displayed formula. For example $\sum_{n=1}^{\infty}\frac{1}{n}$ and

$$\sum_{n=1}^{\infty}\frac{1}{n}$$

are both produced with `\sum_{n=1}^{\infty}\frac{1}{n}`, between a pair of `$`'s or `$$`'s respectively. LaTeX makes these decision for you, but you can reverse them with the `\displaystyle` and `\textstyle` declarations: the in-text formula $\displaystyle\sum_{n=1}^{\infty}\frac{1}{n}$ and displayed formula

$$\textstyle\sum_{n=1}^{\infty}\frac{1}{n}$$

are produced with

```
$ \displaystyle \sum_{n=1}^{\infty}\frac{1}{n} $
```

and

```
$$ \textstyle \sum_{n=1}^{\infty}\frac{1}{n} $$
```

respectively.

Finally, Table 2.6 shows how to fine-tune horizontal spacing in maths-mode (you can also use `\hspace` and `\vspace` as you would in text-mode). Note that `\!` is a negative space.

`\!`	$|\!|$ (negative)		
`\,`	$|\,|$	`\␣`	$|\ |$
`\:`	$|\:|$	`\quad`	$|\quad|$
`\;`	$|\;|$	`\qquad`	$|\qquad|$

Table 2.6: Spaces in maths-mode

Compare

`$\int\int f(x,y) dy dz = \sqrt{2}x$`	$\int\int f(x,y) dy dz = \sqrt{2}x$

and

`$\int\!\!\int f(x,y)\,dy\,dz = \sqrt{2}\,x$`	$\int\!\!\int f(x,y)\,dy\,dz = \sqrt{2}\,x$

Quotations and so forth

Quotations can be handled by one of two environments: `quote` and `quotation`. Both environments display a passage of text with margins indented on both the left and right of the page.

The `quote` environment is intended for small passages; there is no indenting at the beginning of a paragraph and blank lines in the source code produce a small vertical space between lines. The `quotation` environment is intended for longer passages; it is typeset in much the same way as paragraphs normally are.

All of the sample source code in these notes is displayed with a `quote` environment.

A variation on the quotation theme is the `verse` environment, intended for poetry. Lines are ended with `\\` commands, or with `\\*` if LaTeX should not break a page at that point. Blank lines in the source code start a new stanza, and stanzas are separated in the output by a small vertical space. Any lines which are longer than the allowed width will continue on the next line with a small indentation.

Exercise 2.14. Typeset the following limerick using the `verse` environment:

> There was a young man from Japan,
> Whose limericks never would scan.
>
> When told this was so,
> He replied "Yes, I know...
> But I always like to try to fit as many words into the last line as I possibly
>   can."

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.

(2.1)
```
\begin{description}
  \item[\TeX] a complicated typesetting system;
  \item[\LaTeX] a smooth, easy-to-use, completely intuitive
       typesetting system based on \TeX\ that does everything
       you could ever want with little or no effort;
  \item[Word] admittedly quite nice in its own way.
\end{description}
```

(2.2)
```
\begin{enumerate}
 \item First point, with
   \begin{itemize}
    \item A dot point;
    \item Another dotpoint, although
      \begin{enumerate}
       \item \begin{enumerate}
              \item This
                \begin{itemize}
                 \item says
                \item not much
                \end{itemize}
                \item so there
                \item \begin{enumerate}
                       \item point taken
                       \item indeed
                      \end{enumerate}
               \end{enumerate}
        \item not very interesting
         \item nor insightful
           \begin{itemize}
            \item Some more dots
              \begin{itemize}
               \item with subdots
                \begin{itemize}
                  \item and a subsubdot
                  \item or two
                 \end{itemize}
               \item which really do
              \end{itemize}
            \item say very little
           \end{itemize}
        \end{enumerate}
      \item Back to this one.
     \end{itemize}
 \item And the final point.
\end{enumerate}
```

(2.3)

```
\begin{tabbing}
 On Sunday \= I went to the market,\\
           \> and bought \= a pig.\\
 It never occurred \= to me\\
                   \> that I should instead\\
 have bought       \>    \> a cow.
\end{tabbing}
```

(2.4)

```
\begin{tabular}{|l|r|r|}
 \hline       Item   & \multicolumn{2}{c|}{Price}\\
 \cline{2-3}         & Aus \$ & US \$ \\
 \hline\hline \TeX   & 0.00   & 0.00  \\
              \LaTeX & 0.00   & 0.00  \\
              Word   & 174.95 & 89.99 \\
 \hline
\end{tabular}
```

(2.5)

```
\begin{tabular}{l|r@{.}l}
 Name & \multicolumn{2}{c}{Value}\\ \hline
 one hundred      & 100&000 \\
 pi               &   3&14159 \\
 cos(2)           & --0&416147\\
 inches per metre &  39&3701
\end{tabular}
```

(2.7)

```
\begin{table}
 \centering
 \begin{tabular}{|l|r|r|}
        ⋮
 \end{tabular}
 \caption{Price of typesetting packages}
\end{table}
```

(2.8)

```
$a_{i^{2}+1}^{3} = b_{j}^{4+c_{k}^{-1}}$
```

(2.9)

```
\[ \sum_{i=1}^{n} a_{i} x^{i} = \prod_{0<j<m} (1-x^{j}) \]
$$ \int (2x+1) dx = x^{2}+x+\mbox{constant} $$
```

(2.10)

```
\[ \frac{dy}{dx} = \frac{1}{\sqrt{1+\frac{1}{x^{2}}}}
   - \sqrt[4]{\sin \left( \frac{e^{x}}{x(1+x)} \right) +1} \]
```

(2.11)

```
$$ A \not= \pi r^{2} \Rightarrow \exists \delta_{1} \leq
   \delta_{2} \leq \cdots : \frac{\partial A}{\partial r}
   \approx \prod_{i=1}^{\infty} \delta_{i} $$
```

(2.12)

```
$$
 \det \left[
  a_{1}  & 0     & \cdots & 0     \\
  0      & a_{2} &        & 0     \\
  \vdots &       & \ddots & \vdots\\
  0      & 0     & \cdots & a_{n}
 \end{array}
 \right] = a_{1} a_{2} \cdots a_{n}
$$
```

(2.13)

```
\begin{eqnarray}
c^{2}     & =           & a^{2} + b^{2} - 2ab\cos\theta \\
\bar{y}   & \rightarrow & e+f+g+h+i+ \nonumber\\
          &             & j+k+l+m\\
\tilde{x} & \in         & Y \cap Z\\
          & \subseteq   & Y \nonumber
\end{eqnarray}
```

(2.14)

```
\begin{verse}
 There was a young man from Japan,\\
 Whose limericks never would scan.

 When told this was so,\\
 He replied ``Yes, I know\ldots\\
 But I always like to try to fit as many words into the last line
 as I possibly can.''
\end{verse}
```