A LaTeX Primer

Notes from *Guide to LaTeX*

Dan Liddell

1 Foundational Ideas

This document discusses foundational LaTeX concepts. The content is chosen to get you up and running with many commonly-used LaTeX features.

LaTeX is a document preparation system. Part of that system is the LaTeX document itself that you write or edit. That document (which we will often call the *source document*, or simply the *source*) consists of coded commands along with the content you want to communicate to your audience. The source document filename always has a *.tex* extension. The source for this document is the included *kopka_daly.tex* file.[1] You can study that file while reading this one to see how things are put together.

If you've ever worked with HTML or CSS, LaTeX code has analogous features. As with HTML and CSS, your audience does not usually see your source. They see a document generated from it (which we will call the *generated document*), often in PDF or some other format that facilitates reading.

The elementary structure of any source document's code is this:

```
\documentclass['['<options>']']{<class>}
[<global specifications>]
\begin{document}
<body text and mark-up commands of local scope>
\end{document}
```

A document's *class* determines the overall structure and the types of sections it includes. Some of the common ones are article, book, letter, and report. There are others, along with variations within these categories. This document is in the article class.

There are numerous *options* that can be specified with `\documentclass`. These occur in a comma-delimited list. Among the more common is the font size. The default is '10pt', with '11pt' and '12pt' being other choices. Page size is another common option. The default is 'letterpaper', which is US letter size. There are numerous other choices, such as 'portrait' (default) and 'landscape' orientation, 'oneside' (default) and 'twoside' printing of pages, and so on.

[1] The inspiration for this content comes from *Guide to LaTeX*, Fourth Edition, by Helmut Kopka and Patrick W. Daly.

Everything before `\begin{document}` is called the preamble, which contains *global specifications*. These are commands that apply throughout the source unless overridden locally in the main body. Among the common global specifications are packages, which are configuration files that often have their own sets of unique commands that are not available if the package isn't loaded. Packages govern page layout structure, fonts, graphics, table and caption formats, footnote style, and so forth. Complete package descriptions and documentation are available from the CTAN website.

The syntax for loading packages is this:

```
\usepackage['['<options>']']{<packages>]}
```

Multiple options and packages appear in comma-delimited lists. If you get an error when trying to load a package, you may have to add the package to your LaTeX environment, as not all packages are necessarily present in your current installation.

The main body of the source starts with `\begin{document}` and consists of the content you want the reader to see, along with markup commands that control how that content appears. The source concludes with `\end{document}`.

2 Space Characters and Punctuation

Within the main body of the source document's code, space characters are handled specially. As this behavior is important to understand and can result in some initial confusion, we will explain it first.

Here is a synopsis of space types:

Space Characters	Description
(without quotes) ''	normal space character, repetitions ignored.
(without quotes) '\ '	forced space (note trailing space character). frequently used after command names, which usually suppress trailing spaces (e.g., `\mycommand\ `).
~	non-breaking. suppresses line wrapping between words.
\,	small. roughly half a normal space.
\@	following capital letter and preceding a period to get normal spacing (e.g., `CD\@.`).

Multiple adjacent space characters count as one space. Leading spaces on a line are ignored.

Spaces terminating a command name are removed. Sometimes this requires forcing a space with '\ ' immediately following a command to get the desired effect.

The end of a line is treated as a space character, which is fine most of the time. To suppress this, use a comment character '%' at the end of the line.

A period preceded by a lowercase letter is interpreted as the end of a sentence, which gets additional spacing. To override this, use '\ ' or '~' to manage the space. A capital letter before a period is not handled as the typical end of a sentence. To get the usual spacing when a capital letter ends a sentence, you must follow the capital letter with '\@'.

For the most part, punctuation marks behave intuitively. The handling of spacing following the period can be a bit troublesome in certain cases as mentioned in the previous paragraph.

A few other marks that can cause some initial confusion are shown in the following table.

Characters	Type
`` ` ' ``	‘single quotes’ (back-tic then apostrophe)
`` `` '' ``	“double quotes” (just doubled single quotes)
`-`	hyphen -
`--`	en-dash –
`---`	em-dash —
`$-$`	minus −

Hyphenation at line breaks is handled automatically by the tool, but there can be cases where it doesn't do the correct thing. In such cases, '`\-`' can be used to induce a hyphen, but this doesn't force one. Also, this is a piecemeal solution that's only convenient when used a few times. If you have a list of words for which you want to provide hyphenations, this can be done in the source document preamble using the `\hyphenation{<word list>}` command.

A non-breaking hyphen is specified as `\mbox{-}`.

3 Commands, Declarations, and Environments

These special characters are commands in their own right: `\ ~ ^ < > { } # & _ $ %`. Some of these characters have already been mentioned and will be described further. For example, the `%` character could be thought of as a command unto itself when used for a comment. When this is done, everything from this character to the end of a line in the source document is ignored during the typesetting process. A backslash prefixing any of the characters `{ } # & _ $ %` escapes their special status, but it doesn't for the other special characters.

3.1 Named Commands

Named commands have a `\` followed by a string that is the name.

Most named commands take arguments in braces `{}` or brackets `[]`, but some do not, like “`\LaTeX `”. The trailing space character acts as a delimiter and does not survive the typesetting process. So, as mentioned previously, the space character must be escaped with a backslash to preserve it (making it a forced space).

The simplest form of command having an argument set is `\<name>{<text>}`, where `<text>` is required. The argument could be text that is printed, or it could be a definition of a value of some kind. The braces define the scope of the command.

Some commands have more than one set of required arguments. As an example, this command controls numerous measurement settings:

```
\setlength{\<name>}{<measure>}
```

The <name> is the length parameter to set, like the page width or margin. Two common length parameters are \parskip and \parindent, which govern paragraph spacing and indentation (and also happen to be commands).

The <measure> is a value or expression. The units must be given for the measure. The units in, cm, mm, pt (point), and pc (pica) are commonly used, and these measurement units apply to spacings referred to throughout this document.

Some commands support optional arguments enclosed in brackets '[<args>]...'. There may be more than one set of these. The optional arguments precede the mandatory ones appearing in braces.

Some command names can have an asterisk '*' appended to them, thus giving a new name for a command. These commands behave somewhat differently than the ones without asterisks. This often has something to do with whitespace management, but there are other variations as well.

There are many built-in commands. The existing set in your LaTeX environment can be further expanded by loading packages, as mentioned previously.

Commands can also be user-defined in this form:

```
\newcommand{<name>}{<existing_commands>}
```

This is a simpler form of the command. The <name> is the user name for the command. The <existing_commands> are previously defined commands (either built-in or user-defined) that are executed as part of the new command. A more complicated form exists for arguments containing variables, such as this one:

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

Don't worry about understanding \ensuremath and \ldots, just concentrate on the variable calls indicated by the # symbols.

The \vect command takes three arguments that are passed as variables #1, #2, and #3: a variable name, an initial index name, and a terminal index name. With the call \vect{u}{i}{n} in the source document, this appears in the generated output: $u_i, \ldots, u_n$

The \renewcommand command takes the same syntactical form as \newcommand and redefines an existing command. With \renewcommand, we can redefine the previous command to assume a default variable name x if one isn't provided explicitly in a call to the command:

```
\renewcommand{\vect}[3][x]{\ensuremath{#1_#2,\ldots,#1_#3}}
```

So with the call \vect{1}{n}, we get this: $x_1, \ldots, x_n$

3.2 Declarations

Declarations are commands that simply have names: \<name>. Declarations affect how printed text appears and remain in effect until overridden by another declaration, or the scope of the current environment terminates the declaration's effect. More on environments and scope later.

An example declaration is \itshape, which causes italics to be used in printed text until overridden.

In order to show special characters or command names literally (without having them interpreted during typesetting), the \verb declaration is useful. The general form is this:

```
\verb<delimiter><text><delimiter>
```

where the <delimiter> is a character that is not a letter or special to the system. (This is atypical of declarations, which generally have no delimiters defining their scopes.) The <text> is what you want printed verbatim. The text appears in monospace font, and automatic line wrapping within the text block is disabled. (The *shortvrb* package allows definition of a delimiter for verbatim text using the \MakeShortVerb{\<char>} command, where the <char> is a character—often '|'. The \DeleteShortVerb command removes the status of the character.)

Declarations have one more distinguishing feature: their names can be used (without the leading \) to define environments, which are discussed next.

3.3 Environments

Environments are scoped blocks of text that are rendered according to the environment command parameters. An environment begins with \begin{<name>} and ends with \end{<name>}. The environment \begin{document} and \end{document} is an example we've seen already.

When a declaration name is used in an environment, the behavior of the declaration applies within the scope of the environment, not outside of it. Environments may be nested when it makes sense to do so.

Simply using braces {...} without a preceding name produces a nameless environment. The braces determine the environment's scope. This form has certain useful applications like limiting the scopes of declarations appearing in the environment. If using a nameless environment, then the initial \ of a declaration must be present after the { character.

There are many built-in environment commands, but you can also create your own. Custom environments are useful when you have numerous commands used together to achieve a certain effect employed often throughout a document.

The syntax for custom user environments is similar to user declarations:

```
\newenvironment{<name>}[<arg_count>][<arg1_default>]
{<begin>}
{<end>}
```

The <name> is the name of the environment. The <begin> block defines what gets printed when the \begin command of the environment is executed. The \end command in the <end> block concludes the scope of a \begin command.

The user-defined environment call may optionally have arguments, and the count of the arguments is <arg_count>. No arguments may appear in the <end> block. Like user declarations, the first argument can have a default value <arg1_default> if the first

argument is omitted. The `\renewenvironment` command has the same arguments and redefines an existing environment.

Here is an extended example that uses some commands that haven't been discussed. Just concentrate on the basic form of the `\newenvironment` command:

```
\newsavebox{\comname}
\newcounter{com}
\newenvironment{newcomment}[1]
{\rule{2cm}{1pt}\stepcounter{com}\begin{sloppypar}\noindent
\slshape
Comment (\arabic{com}): #1
\begin{quote}\small\itshape}
{\end{quote}\end{sloppypar}}
```

This call in the source code:

```
\begin{newcomment}{A reviewer}
A comment.
\end{newcomment}
```

renders this after typesetting:

Comment (1): A reviewer

> *A comment.*

4 Sectioning Commands

Certain commands support division of content into logical sections. The types of commands that are available depend upon the document class. The following are common sectioning commands.

Command	Description
`\chapter{<name>}`	chapter division.
`\section{<name>}`	major section.
`\subsection{<name>}`	secondary section.
`\subsubsection{<name>}`	minor section.
`\paragraph{<name>}`	numbered paragraph.
`\subparagraph{<name>}`	minor numbered paragraph.

Other sectioning commands govern things like front matter tables (contents, figures, tables), appendices, bibliographies, and indexes.

All sections receive a header and sequential number by default. The font and leading have defaults. All of these characteristics are configurable.

5 Whitespace and Pagination Controls

Much of the generated document's horizontal whitespace is managed by the source preamble. In certain cases, it may be desirable to have some custom spacing at certain locations in the generated document. The following commands govern horizontal spacing.

Command	Description
`\frenchspacing`	normal space after periods until overridden by `\nonfrenchspacing`.
`\nonfrenchspacing`	extra space after periods not preceded by a capital letter; this is the default.
`\hspace{<spacing>}`	horizontal space of <spacing> width to the right; ignored at start of an output line.
`\hspace*{<spacing>}`	like `\hspace` but often more useful; spacing is always applied.
`\quad`	horizontal space equal to current typeface size.
`\qquad`	double `\quad`.
`\hfill`	forces text on either side of command to the margins.

Vertical spacing is also largely managed by the preamble. Blank lines in the code cause paragraph breaks in the generated document. Consecutive blank lines in the code are treated as a single blank line.

The following commands manage vertical whitespace:

Command	Description
`\\[<space>]`	inserts a newline within a paragraph with optional additional <space> gap; permits page breaking to occur, in which case vertical spacing isn't applied.
`\\*[<space>]`	like `\\` but spacing never ignored.
`\newline`	alias for `\\`; both are ignored between paragraphs.
`\par`	new paragraph, analogous to blank line; ignored on its own line.
`\vspace{<spacing>}`	similar to `\hspace` but applied vertically.
`\vspace*{<spacing>}`	similar to `\hspace*` but applied vertically.
`\vfill`	similar to `\hfill` but applied vertically.
`\linebreak[0-4]`	similar to the `\\` commands except it justifies the current line; default is 4, which forces a newline, while lesser values decrease the likelihood.
`\nolinebreak[0-4]`	similar syntax and opposite semantics to `\linebreak`.
`\newpage`	similar to `\newline` but applies to page breaking; fills remainder of current page with whitespace.
`\clearpage`	similar to `\newpage` but forces floating figures and tables onto subsequent pages before starting the new page.
`\pagebreak[0-4]`	similar to `\linebreak` but applies to page breaking; default is 4; takes effect at end of current line.

Command	Description
`\nopagebreak[0-4]`	complement to `\pagebreak`; similar to `\nopagebreak`.

By default, paragraphs are set off by indentation with no spacing between them. This can be changed to the inverse of both settings by using this command in the preamble:

```
\usepackage[parfill]{parskip}
```

It is also possible to use the `\parskip` and `\parindent` arguments to `\setlength` discussed earlier.

The overall styles of pages are governed by these commands, which have some effect on whitespace:

```
\pagestyle{<style>}
\thispagestyle{<style>}
```

The first command configures the global page style and is often used in the preamble. The second command configures a local page. Common styles are fancy (used with the *fancyheader* package), plain, and empty. Use empty if you want a plain page without numbers.

To keep paragraphs together so that they cannot appear on separate pages, the `\parbox[<pos>]{<width>}{<text>}` command may be used. The `\vbox` command can also work, but it has some touchy behaviors like disabling the automatic insertion of a blank line before a paragraph, so it should be used with care. Both of these commands interfere with printing of footnotes within the paragraphs in the scope of the commands. In that case, `\pagebreak` is a better choice to force a paragraph to the next page.

The `\enlargethispage[*]{<size>}` command can adjust the page height so that you can accommodate more text lines on a page. This allows a certain amount of page break control, which implies widow and orphan control, but it's not global. The non-* version of the command makes the printed area bigger (allows more lines without adjusting spacing). The * version does the same thing and shrinks the interline spacing as needed to accommodate more text.

Page numbering is controlled by this command:

```
\pagenumbering{<style>}
```

The standard style is "arabic". "roman" is also a common option. There are others.

This command controls the starting index of pages:

```
\setcounter{page}{<number>}
```

5.1 Indented and Offset Sections

Quotations can appear in indented sections and are specified with the `\begin{quote}` and `\begin{quotation}` environments. The former has no paragraph indentation but

has separation. The latter has just the opposite. As with all environments, these are concluded with an \end{<env>} command. These environments may be nested to a depth of 6 ply, but 3 is a practical maximum.

Lists are constructed in three different ways: bulleted, sequential, and descriptive. All lists are specified in an environment block similar to quotations mentioned in the preceding paragraph. The three environment names are: \itemize, \enumerate, and \description. For the \itemize and \enumerate environments, each member of the list begins with \item as the first command on a line. In the case of the \description environment, each member begins with \item[<string>] where the string is an optional word or phrase that is typically followed by its definition. If you leave the string out, the brackets are also omitted. Like quotations, lists may be nested, but are limited to 4 ply rather than 6. Here is an example of single-item lists.

- a bullet

1. a trivial sequence

   trivial a nested trivial description

Here is a description block that isn't nested:

trivial a trivial description

Both the \itemize and \enumerate environments have default characters that are used for the lists, and these characters change with the number of ply in the list. If you want to choose your own list labels, the \item command supports an optional label argument \item[<label>]. Labels can be changed globally in the document for the two environments with this command:

```
\label{item|enum}{i|ii|iii|iv}'{'<label>'}'
```

The unquoted braces are not literal. The roman numerals stand for the list ply level for which the specified label is used throughout the document. This command is usually nested in a \renewcommand declaration.

5.2 Boxes

Boxed text acts as a unit that can't broken up and can affect on how whitespace appears.

The \makebox and \framebox commands, and their abbreviated versions, create horizontal blocks of text. The \mbox command is useful to prevent line breaking between words. \framebox puts a wire box around a block of text in addition to what \makebox does.

The \raisebox command creates a vertically adjusted block of text and can be used to create superscripts and subscripts.

The \parbox command creates a vertical block of text with of a specified width. In essence, it creates a custom paragraph. A drawback is, it doesn't accept verbatim text. For that, you need a *minipage*.

Box Commands	Description
`\mbox{<text>}`	centered text in a borderless box.
`\makebox[<width>][<pos>]{<text>}`	like `\mbox` but with width and text alignment controls.
`\fbox{<text>}`	centered, bordered box.
`\framebox[<width>][<pos>]{<text>}`	like `\makebox` but with border.
`\raisebox{<lift>}{<text>}`	raises or lowers text versus the baseline.
`\parbox[<pos>]{<width>}{<text>}`	creates a custom paragraph block.

The `<pos>` controls justification, which is centered by default. The values are l, r, and s, for left, right, and stretch to fit (fill). The `<width>` is a non-negative measurement value. The `<lift>` is positive to raise the box and negative to lower it.

If you need more control over paragraph-like blocks of text, try the `\minipage` environment.

5.3 Footnotes

Footnotes are a type of box. The `\footnote[<num>]{<text>}` command works in the majority of cases. The index of the footnote is automatically incremented. The index counter can be manually adjusted with a number of commands discussed later (a common method is briefly discussed in the next section), and the index character can be as well.

The main limitation of the `\footnote` command is, it can't be used in the box environments discussed earlier or in tables which are discussed later. If used inside a minipage, the footnote appears at the end of the minipage, not at the bottom of the page.

For environments that don't support the standard `\footnote` command, the `\footnotemark` and `\footnotetext` commands can be used instead. These can achieve a number of customized footnote effects. Here is an example (see the code)[2]: this[3] that[4]

6 Fonts

Fonts have these attributes: family, series, shape, size, and encoding. The first four attributes have some convenient high-level commands that are useful. These are the declarations and corresponding inline commands (defaults are out of the box defaults and can be overridden in the preamble):

```
% family
\rmfamily \textrm{<text>}   % roman; default
\sffamily \textsf{<text>}   % sans serif; (why not ss or ssf?)
\ttfamily \texttt{<text>}   % monospace typewriter
```

[2] this is a standard footnote
[3] first box note
[4] second box

```
% series
\mdseries \textmd{<text>}   % normal weight and expansion;
                            % default
\bfseries \textbf{<text>}   % bold

%%% shape
% upright; \textnormal defaults to this
\upshape \textup{<text>} \textnormal{<text>}

% italics
\itshape \textit{<text>} \emph{<text>} {\em ...}
% \emph and \em toggle back and forth when nested
% \itshape and \emph use italic space correction,
% \textit does not

% slanted
\slshape \textsl{<text>}

% small caps
\scshape \textsc{<text>}

% size declarations
\normalsize % 10 pt by default; can be set to 11 or 12
\large % 12 pt
\Large % 14.4 pt
\small % 9 pt
\footnotesize % 8 pt
```

The following text is an example of an unnamed environment. See the source code to understand how the commands are scoped:

this text is Large *and now slanted*

The preceding commands are actually built upon other lower level commands. These other commands have the form `\font<attr>`, where `<attr>` is one of the font attributes.[5]

The encoding selects the lookup table for the font. In essence, this sets the fundamental character forms that are printed.

The family sets the font properties such as serif (r for roman), sans serif (s/ss), or equal spacing (tt for typewriter). There are various names for the families, with the letters r, s, and t often giving hints as to what the family looks like.

The series sets the weight and the width. The letter 'l' for a weight means lighter than normal. The letter 'b' means heavier than normal. The letter 'm' means normal (think median). For 'l' and 'b', these prefixes further modify the weight: s = semi (means less than 'l' or 'b' by themselves), e = extra, u = ultra. Width uses similar conventions, with 'm'

[5] Pages 362-4 of Kopka and Daly cover these in detail.

meaning normal, 'c' meaning compressed, and 'x' meaning expanded. Same modifiers as for weight apply.

The shape sets the angle or small caps. The letter 'n' is normal, 'sl' is slanted, 'it' is italic, and 'sc' is small caps.[6]

The size sets the height of the letter 'x' in points, along with the vertical separation between lines in points. There are 12 preset values the pitch can take out of the box.[7]

If a font with all the specified attributes cannot be found, the tool issues a warning and says which attributes are used.

As an example:

```
% defaults are OT1, cmr, mm, n, 10, and the default spacing
% OT1 is a font class. cmr is Computer Modern Roman,
% mm is probably median, n is upright, 10 is 10 pt.
% use Cork encoding; this expands on Knuth's original table
\fontencoding{T1}
\fontfamily{cmss} % computer modern sans serif
\fontseries{sbm}  % semibold normal width
\fontshape{n}     % upright
\fontsize{14.4}{16} % 14.4 pt height, 16 pt space between lines
\selectfont % activates everything in preceding scope
```

A more abbreviated command set is this:

```
\usefont{T1}{cmss}{sbm}{n}
\fontsize{14.4}{16}
\selectfont % remains in effect until next \selectfont command
```

[6] The book also shows a 'u' which possibly means upright.

[7] Other values may be added, but the book doesn't mention how.

7 Numbering

Numerous elements have counters associated with them, such as sections, paragraphs, pages, footnotes, and so on. These counters can be manipulated. Custom counters can also be created. The following commands control counters.

Command	Description
`\newcounter{<name>}[<in_counter>]`	defines a new counter and sets its value to 0. `<in_counter>` associates the incrementing of the new counter with an existing one.
`\addtocounter{<name>}{<value>}`	adds a numeric value to a counter.
`\stepcounter{<name>}`	increments a counter and resets all counters associated with it to 0.
`\refstepcounter{<name>}`	works like `\stepcounter` but also makes the referenced counter the current counter for the `\label` command.

Here is some source code that shows how `\refstepcounter` works:

```
\newcounter{test}
\addtocounter{test}{99}
\refstepcounter{test}\label{foo}
Counter value: \thetest
```

Counter value: 100

(Notice the counter is referenced as `\thetest` rather than simply `\test`.)

The `\ref{foo}` value of the label is 100.

If you place the label inside some special context like a caption within a table or figure, then the referenced counter value may not be that of the custom counter, but that of the special context environment.

8 Tables

There is a rich set of commands and options that support sophisticated tables. For most tables, however, a relatively simple environment suffices:

```
\begin{tabular}{<cols>}
  [\hline]
  <text> & <text> & <text> \\
  [\hline]
  ...
\end{tabular}
```

The <cols> block is comprised of the letters l, c, and r, which stand for left, center, or right justification of a column. Each column of the table must have its justification defined by one of these letters. Additionally, the "|" character may appear in the <cols> block each time a vertical line is desired to set off a column in the table. The <text> blocks are where cell text appears. Each "&" represents a column delimiter. The \\ characters delimit rows. The \hline command can be used to insert horizontal lines that form row boundaries.

The \cline{<m>-<n>} command may be used instead of \hline to insert a horizontal line that spans columns 'm' through 'n'. The \vline command draws a vertical line within a row at the location the command appears.

Here is an example that uses all the basic table constructs:

leftward	center		rightward
left	center	center	right
left	center		right

Commonly, tables are made to float so whitespace usage can be optimized. (The same is true of figures. Collectively, tables and figures are often referred to as *floats*.) This allows tables to appear on other pages than where they appear to be defined on in the source code.

This uses the table environment:

```
\begin{table}'['[h][t][b][p]']'
  <header_text>\\'['<spacing>']'
  \begin{tabular}{<cols>}
  ...
  \end{tabular}\\'['<spacing>']'
  <footer_text>
\end{table}
```

There are many options that can be used to configure floats, and most of them are rather specialized.

The 'h t b p' options control preferences for where the float should appear. These are short for here, top, bottom, and page. The 'h' option means prefer the current location. The 't' and 'b' options mean prefer the top and bottom of the current page, respectively. The 'p' option means prefer a special page dedicated to floats. The default is [tbp].

A numbered caption can be included using the following command:

```
\caption'['<short_title>']'{[\label{<target>}]<caption_text>}
```

The <short_title> does not appear in the caption; it appears as the table title in a generated list of tables. The <caption_text> is the main caption text and may be rather long. It functions as the short title if <short_title> is omitted. The optional, but useful, \label command has a <target> string that can be used in \ref and \pageref commands to cross-reference the float's index number and page, respectively.

Caption text is center justified if it is on one line. Otherwise, it is set as a normal paragraph. The width can be controlled by placing the caption command in a \parbox or \minipage command.

Table 1: Leading caption.

Floating Table Header

leftward	center		rightward
left	center	center	right
left	center		right

Table Footer

The caption appears before the table if the command is placed at the start of the float environment. It is placed after the table if it is placed at the end of the float environment.

The float index number for captions is incremented every time a float is used, not when a caption is used. This means caption numbers may not be sequential if you don't consistently use captions. Interestingly, this behavior holds if you show a float environment as verbatim text (that is, no intent to show an actual table, but simply to show the syntax). To overcome this problem, you can use this command:

```
\addtocounter{table}{<number>}
```

where the number may be any integer. The number is added to the current index value to give the desired index value.

The floating table code appears in the source code here, but the table in a generated file appears either at the top or bottom of the page.

9 Importing Graphics

Graphics importation has some complexities that are driven by the desired output format, which affects which compiler to use for generating the output. For this document, we assume the output is PDF and PdfLaTeX is used for the compiler.

There are two packages to choose from to enable graphics importation: *graphics* and *graphicx*. Each one supports a different syntax, but both have the same capabilities. Which one you use is a matter of choice.

These packages support a number of options. Perhaps the most useful option is 'draft', which causes graphics not to be imported. This can help performance, but more importantly, it shows the bounding box that the imported figure will occupy. This can assist in fine-tuning the position and scaling of the imported figure.

For the *graphicx* package, the command to import figures is this:

```
\includegraphics'['<key>=<value>[,<key>=<value>...]']'<file>
```

There are many supported key-value pairs. The more frequently used ones are 'draft=true' and 'scale=*factor*'. The draft option is the same as discussed previously. The scale option takes a positive factor by which to magnify or shrink the imported graphic.

PdfLaTeX supports JPEG, PNG, and PDF imported images. Interestingly, it does not support postscript formats.

To float a figure, a similar structure is used as was discussed under the Tables section:

```
\begin{figure}'['[h][t][b][p]']'
  \includegraphics'['<key>=<value>[,<key>=<value>...]']'<file>
  [\caption[<short_title>]{<caption_text>}]
\end{figure}
```

Here is an imported PNG file:

10 Defining and Using Color

For all LaTeX compilers, the colors black, blue, cyan, green, magenta, red, white, and yellow are pre-defined and may be called by name. Other colors may be declared with this command:

```
\definecolor'{'<name>'}{'rgb | cmyk'}{'<spec>'}'
```

The `<name>` is the name of the color. The `\<spec>` is a comma-delimited list of floating-point numbers on the interval [0, 1]. If the the rgb model is used, three numbers stand for the percentage of red, green, and blue in the color, respectively. If the cmyk model is used, then four numbers stand for the percentage of cyan, magenta, yellow, and black in the color. (For the rgb model, a spec value of 0.004 is roughly equivalent to 1 unit of color on the 255 scale.)

The following commands use color. The presentation here assumes a color name is used, but the specification `{rgb | cmyk}{<spec>}` may be used instead.

Command	Description
`\color{<color>}`	sets current text color.
`\colorbox{<color>}{<text>}`	sets text in a box with the color as background.
`\fcolorbox{<clr1>}{<clr2>}{<text>}`	sets text in a box with a frame of color clr1 and a background of color clr2.
`\normalcolor`	sets color to the one active at the end of the preamble.
`\pagecolor{<color>}`	sets page background color.
`\textcolor{<color>}{<text>}`	prints the text using the color.

An example `\colorbox`.

11 Drawing

LaTeX offers a *picture* environment for basic drawings, many of which we will describe here.

Drawings in the *picture* environment are based upon a coordinate system. The unit of length is given by this command:

```
\setlength{\unitlength}{<unit>}
```

The `<unit>` is a number followed by some unit like mm, cm, in, and so on. The default unit is 1pt. Changing units of an existing figure changes the scale of how it is rendered. It is not permitted to use more than one unit length for a picture, but the unit length can be different for different pictures.

The *picture* environment syntax is this:

```
\begin{picture}{<length>,<width>}
<picture_commands>
\end{picture}
```

The length and width parameters are in units of the `\unitlength` and define the extent of the drawing. So it's good to know the dimensions of the sheets you plan to use. The extent of the picture is centered on the page by default (this can be overridden by giving an offset). The lower left corner of the picture extent is the origin of the picture's coordinate system. Negative coordinate values are permitted, but it tends to be helpful to work in first quadrant coordinates. In general, the system does not prevent you from crossing the boundaries of the frame or the page margins.

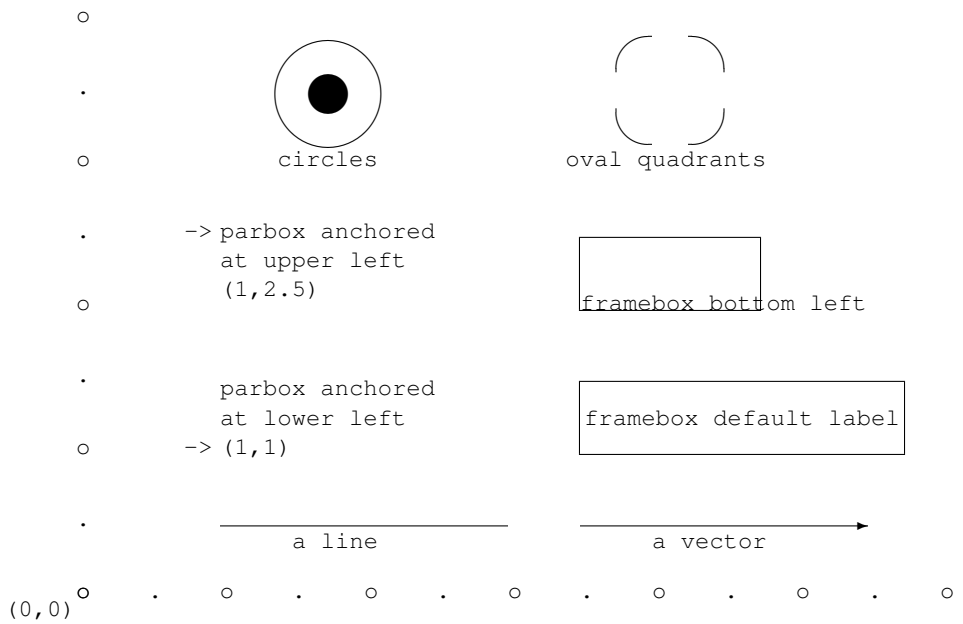
Picture, font style, size, `\thicklines`, and `\thinlines` commands are the only ones allowed in the picture environment. `\thinlines` is the default for weight of drawn lines.

Picture commands come in two forms:

```
\put(<x>,<y>){<element>}
\multiput(<x>,<y>)(<dx>,<dy>){<n>}{<element>}
```

These are required to place the specified picture element at the stated coordinates. In the case of `\multiput`, the element is repeated an integer 'n' number of times (the first placement is included in the count) and placed with the specified `<dx><dy>` offsets from the last location at each iteration.

The following are graphics elements examples. See *Inputs/graphics.tex* for the code.



Notice in the figure that the system doesn't prevent the labels from overflowing a box or the page margins (it doesn't even ensure that an object is on the page).

The following commands all make boxes with text in them and can be used in the `\put`-type commands:

```
\dashbox{<dash_length>}(<width>,<height>)[<loc>]{<text>}
\framebox(<width>,<height>)[<loc>]{<text>}
\makebox(<width>,<height>)[<loc>]{<text>}
```

The first command creates a dashed border. The second a continuous border. The third has no border. The figure shows some examples of the second. For greater control, the `\makebox` command is frequently used with (0,0) width and height to place text centered at a location.

Text is all on one line and is centered by default. The `<loc>` parameter has these possibilities for changing text location in the box:

b – bottom

t – top

l – left

r – right

s – stretch the text horizontally to fill the box

These can be combined in pairs (that don't contradict each other) to combine effects.

There also is a set of commands that allow saving and re-using boxes:

```
\newsavebox{\<boxname>} % initiates a box definition.
\sbox{\<boxname>}{<text in box>} % simple form
\savebox{\<boxname>}[<width>][l|r|s]{<text>} % fancy form;
                                % default is centered text
```

```
\usebox{\boxname} % place a defined box whenever you like
```

The syntax of `\savebox` might be extended in the same way as the previously mentioned box commands.

Line segments and directed segments (or vectors) are specified with these commands:

```
\line(<dx>,<dy>){<length>}
\vector(<dx>,<dy>){<length>}
```

The `<dx><dy>` parameters give the slope. The dx and dy values must conform to these criteria: integers on the interval [-6, 6], and they must be relatively prime. The length must be at least 10 pt (3.5 mm) for a segment or vector that isn't horizontal or vertical. For horizontal or vertical objects, the length is self-explanatory. For oblique objects, the length is that of the projection of the object on the x-axis.

Circles are specified with these commands:

```
\circle{<diameter>}
\circle*{<diameter>}
```

The diameter is self-explanatory. The * version of the command creates a solid circle rather as opposed to a perimeter. The placement of the circle is based on its center.

Ovals are specified with this command:

```
\oval(<length>,<width>)[<quadrant>]
```

The `<length>` and `<width>` parameters give the dimensions of the horizontal and vertical axes of the oval. The optional `<quadrant>` takes these values:

b – bottom

t – top

l – left

r – right

and controls which part of the oval is printed. These can be combined in pairs (that don't contradict each other) to combine effects.

12 Cross-References and Links

Internal cross-references to locations and pages are specified using these commands:

Command	Description
`\label{<target>}`	a location in the text to reference by the `<target>` name.
`\pageref{<target>}`	prints the page number of the `<target>` label.
`\ref{<target>}`	prints the number or letter of environment context the `<target>` label appears within.

A \label may be placed in any active portion of the source code. It references the location counter of that portion using the immediate context, such as a section, subsection, caption, and so forth. The <target> may contain any string, including one containing special characters. The \ref and \pageref commands can appear before or after the <target> they reference.

As an example, at this location in the source there is a reference to section 10 on page 16.

The *hyperref* package enables hypertext cross-references in the generated document. Enabling this package causes all cross-references to have red boxes drawn around them by default. This includes footnotes, \ref, \pageref, and any commands in the *hyper* family, among others. While this behavior is useful in draft documents to detect where links exist, it is ugly. For a production document, this command in the preamble makes for a less obtrusive presentation:

```
usepackage[colorlinks=true,hyperfootnotes=false,linkcolor=blue
{hyperref}
```

The *hyperref* package has a large number of options and enables numerous commands. The following are a subset of the complete group of commands.

Command	Description
\href{<url>}{<text>}	creates a <text> hyperlink to the <url>. # and & characters may be used in the URL.
\hyperref[<target>]{<text>}	creates a <text> hyperlink to the <target> label.
\hypertarget{<anchor>}{<text>}	defines <anchor> as an internal hyperlink target with the <text> printed in the location of the target. the <text> may be empty.
\hyperlink{<anchor>}{<text>}	creates an internal link to the <anchor> with the <text> used for the link.
\hypersetup{<option>=<value> [,<option>=<value>...]}	sets any of the options of the *hyperref* package locally.

As an example, at this location in the source, there is a \hyperref command cross-referencing the same target label pointed to by the preceding \ref and \pageref commands.

The *hyperref* usepackage option list offers limited selectivity over which types of cross-references create active links and the appearance of links by type. To work around these limitations, the \hypersetup command is useful for local overrides. In particular the *linkcolor* and *hidelinks* options can change the appearance of links on a case-by-case basis.

For example, \hypersetup{hidelinks} is set in the source file here, and this is how the preceding link text appears: “the same target label”.

13 Conditional Text

It can be helpful to have text in the source document rendered only when certain conditions are true. A common example is when a generated document is intended for multiple audiences, but not all text is of interest to each audience. So selective rendering of the text in the generated document is desirable.

To employ conditional text, the *ifthen* package can be loaded. This package enables these commands to be used:

```
\ifthenelse{<test>}{<then_block>}{<else_block>}
\whiledo{<test>}{<do_block>}
```

These behave similarly to if-then-else conditional blocks and while loops in programming languages. For either command, there is a test condition. For `\ifthenelse`, the <then_block> is executed if the test is true and the <else_block> otherwise. For `\whiledo`, the <do_block> is executed so long as the test is true. Within the body of the while script, one must alter a value that is tested so the loop will end.

The Boolean commands are useful for setting up conditions to test:

```
\newboolean{<name>}
\setboolean{<name>}{<value>}
\boolean{<name>}
```

The first two commands initialize a Boolean flag and give it a truth value. The final command returns the value of a Boolean flag.

The *equal* command can be used in conjunction with a Boolean flag:

```
\equal{<string1>}{<string2>}
```

Here is an example:

```
\newboolean{cond1}
\setboolean{cond1}{true}
\ifthenelse{\equal{\boolean{cond1}}{true}}
  {This is printed.}
  {This isn't.}
```