Although \LaTeX{} is best known for its mathematical typesetting, it is also very good at typesetting plain text. It offers a variety of features to simplify the task of writing complex documents.

\section{Formatting commands}

\subsection{Typefaces}
\LaTeX{} will output all text in roman (i.e. upright) type by default, but of course it is able to change this to suit your needs. This is done using formatting commands, which tell \LaTeX{} to set specific text in specific types. For example, to produce the following text:

\texttt{\LaTeX{} is my \textit{favorite} document processor!}

you could use the following code:

\begin{verbatim}
\LaTeX{} is my \textit{favorite} document processor!
\end{verbatim}

The \textit{} command used in the previous example is \textit{semantic}; it indicates that certain text should be \textit{emphasized}, and then allows \LaTeX{} to decide how to handle this. Whenever possible, you should leave these kinds of decisions to \LaTeX{}'s discretion; it will simplify your life and make your documents easier to modify later.

However, there are many formatting commands available to you which explicitly tell \LaTeX{} how to format text.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
Command & Effect \\
\hline
\textnormal{} & default text \\
\textrm{} & roman text \\
\textit{} & italic text \\
\textbf{} & bold text \\
\textsc{} & smallcaps text \\
\textsf{} & sans-serif text \\
\texttt{} & monospace text \\
\hline
\end{tabular}
\caption{Text formatting commands in \LaTeX{}}
\end{table}

(Note: roman text is usually the default, so \textrm{} and \textdefault{} will have the same effect.)

By default, \LaTeX{} does not support underlined text. This is a deliberate omission; underline is generally not used in professional typesetting, as it is basically just a paper-and-pencil substitute for italic and bold type. If you \textit{really} need to underline something, there are packages that provide this.

\subsection{Quotes}

\LaTeX{} supports typesetting quotes in a variety of ways. Of course, the simplest of these is the “inline quote”, which is simply surrounded by quotation marks.
1.2.1 Inline quotes

Due to some complications with parsing, \LaTeX{} does \textit{not} support using the ordinary straight quote characters " and ’ on your keyboard. Instead, to achieve this output:

\LaTeX{} supports “double and ‘single’ quotes”.

you must type:

\LaTeX{} supports ``double and ‘single’ quotes’’.

The opening quote symbols are the “backtick” key on your keyboard, which is usually found above TAB.

But there is another way! Using the \texttt{csquote} package (which is loaded automatically by all the Carleton styles), you can describe quotes semantically, using the \texttt{\enquote{}} command. Thus, you can achieve the same output as before:

\LaTeX{} supports “double and ‘single’ quotes”.

by typing:

\LaTeX{} supports \texttt{\enquote{double and \enquote{single} quotes}}.

This even supports alternative languages! For example, Magritte’s “The treachery of images” famously notes (in French) that « Ceci n’est pas une pipe ». To achieve this effect, I simply wrote

\foreignquote{french}{Ceci n’est pas une pipe}

and let \LaTeX{} and \texttt{csquote} do the rest.

1.2.2 Block quotes

\LaTeX{} also supports block quotes. I have used these throughout this document to illustrate command output. For example, to achieve this output:

This is a block quote.

I used this input:

\begin{quote}
This is a block quote.
\end{quote}

\texttt{csquote} also provides facilities for integrating citations into block quotes; check out its manual if you need this functionality.

1.3 Special symbols

Quotation marks aren’t the only symbol that \LaTeX{} treats a bit strangely. Several other examples are listed in Table\textsuperscript{2}. For each, we list a symbol, the effect of typing it directly, and the code that should be used to produce the given symbol in the final document. (For example, to obtain an ampersand (&), per the table, we need to type \texttt{\&}.)

There are also a few important commands to produce symbols which don’t appear on a standard keyboard. These are listed in Table\textsuperscript{3}.

Additionally, \LaTeX{} supports a wide array of text accents, which are used for typesetting non-English languages. A full list of these would fill a page and is beyond the scope of this document, but we’ll mention one important example in passing: “Erdős” is typeset as \texttt{Erd\h{o}s}, using a double acute accent (and \textit{not} a diacritical, as in “ö”!)

Finally, \LaTeX{} has support for several different kinds of dashes.

\begin{itemize}
\item The ordinary hyphen is obtained using -. This should be used to hyphenate words (but not at line breaks, since this is done automatically). (If used in math mode, it automatically becomes the minus sign − as appropriate.)
\end{itemize}
Table 2: Special characters in \LaTeX

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Use in \LaTeX</th>
<th>Code to print</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;</code></td>
<td>Alignment tab</td>
<td><code>\&amp;</code></td>
</tr>
<tr>
<td><code>%</code></td>
<td>Comment lines</td>
<td><code>\%</code></td>
</tr>
<tr>
<td><code>$</code></td>
<td>Math mode</td>
<td><code>\$</code></td>
</tr>
<tr>
<td><code>{</code></td>
<td>Commands</td>
<td><code>\{</code></td>
</tr>
<tr>
<td><code>\</code></td>
<td>Subscript</td>
<td><code>\_</code></td>
</tr>
<tr>
<td><code>^</code></td>
<td>Superscript</td>
<td><code>\textasciicircum{}</code></td>
</tr>
<tr>
<td><code>#</code></td>
<td>\LaTeX{} internals</td>
<td><code>\#</code></td>
</tr>
<tr>
<td><code>~</code></td>
<td>Non-breaking space</td>
<td><code>\textsc{tilde}</code></td>
</tr>
</tbody>
</table>

Table 3: Non-keyboard symbols in \LaTeX

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code to print</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>§</code></td>
<td><code>\$</code></td>
</tr>
<tr>
<td><code>¶</code></td>
<td><code>\p</code></td>
</tr>
<tr>
<td><code>—</code></td>
<td><code>\texttrademark{}</code></td>
</tr>
<tr>
<td><code>©</code></td>
<td><code>\copyright{}</code></td>
</tr>
<tr>
<td><code>°</code></td>
<td><code>\textdegree{}</code></td>
</tr>
</tbody>
</table>

• The en-dash is obtained using `--`. This should be used for ranges of numbers and dates (such as “1861–1865”) and in certain compound hyphenated adjectives.

• The em-dash is obtained using `---`. This should be used to set off parentheticals—which can also be marked using actual parentheses (naturally)—and to set off the sources of quotations. (Important: when em-dashes are used for parentheticals, they should not be surrounded by spaces!)

2 Document sectioning

\LaTeX{} provides great support for rich document structure. Your document can be divided into sections, subsections, and subsubsections, which may contain paragraphs and subparagraphs; in addition, the \texttt{memoir} class (which is the basis for the \texttt{comps} template) supports parts and chapters. (Paragraphs and subparagraphs are usually only used in legal documents like the U.S. Code, where it is important to be able to refer to something as specific as a single sentence or clause.)

To start a new document section, simply use the type of section as a command. For example, to start this section of the guide, I wrote the following:

\begin{verbatim}
\section{document sectioning}
\end{verbatim}

\LaTeX{} took care of assigning the number 2 to this section. (In Section 5, we’ll talk about how to get \LaTeX{} to take care of references like these.)

Nested layers of sections will be numbered and styled appropriately. For example, Section 1.2.2 was set up simply by typing

\begin{verbatim}
\subsection{block quotes}
\end{verbatim}

and letting \LaTeX{} sort out how to handle a subsubsection.

Unnumbered sections

If you want to section your document without assigning numbers to the sections, you should use the \textit{starred} versions of the sectioning commands. For example, to start this subsection, I wrote the following:
3 Lists

\LaTeX{} also supports several kinds of lists.

3.1 Unordered lists

An “unordered list” is one in which the ordering of items is not important. These can be typeset in \LaTeX{} using the \texttt{itemize} environment. For example, this list of operating systems:

- Windows
- OSX
- Linux

is produced using the following code:

\begin{verbatim}
\begin{itemize}
\item windows
\item osx
\item linux
\end{itemize}
\end{verbatim}

Note the use of the \texttt{item} command to introduce each item.

3.2 Ordered lists

An “ordered list” is one in which the ordering of items is important. These can be typeset in \LaTeX{} using the \texttt{enumerate} environment. For example, the following recipe for eating canned soup:

1. Open can.
2. Pour soup into bowl.
3. Use spoon to eat soup.

is produced using the following code:

\begin{verbatim}
\begin{enumerate}
\item open can.
\item pour soup into bowl.
\item use spoon to eat soup.
\end{enumerate}
\end{verbatim}

Note that we still use \texttt{item} to produce the items; the only difference between this and the unordered list in Section 3.1 is that we used the \texttt{enumerate} environment instead of \texttt{itemize}.

Using the \texttt{enumerate} package (which is included by default in all the Carleton templates), the formatting of these ordered lists can be redesigned to meet your needs. For example, we can number a list using lowercase roman numerals in parentheses:

(i) An item
(ii) Another item

with the following code:

\begin{verbatim}
\begin{enumerate}[i)]
\item an item
\item another item
\end{enumerate}
\end{verbatim}
3.3 Description lists

A third option is the “description list”, which is unordered but has a label for each item. This may be used, for example, to give a list of definitions or descriptions. For example, we can produce the following list of mobile operating systems:

**iOS** Produced by Apple.

**Windows Mobile** Produced by Microsoft.

**Android** Produced by Google and the Open Handset Alliance.

with the following code:

```
\begin{description}
\item{ios} produced by apple.
\item{windows mobile} produced by microsoft.
\item{android} produced by google and the open handset alliance.
\end{description}
```

3.4 Nested lists

Lists may be nested inside lists, simply by including one of the list environments as part of an `item` of another. \LaTeX{} will automatically use different numbering schemes or types of bullet points for different levels. For example, we can produce the following nested list:

- An item.
- Another item, with subitems.
  1. First item.
  2. Second item, with subitems.
     (a) First deep item.
     (b) Second deep item.
- Yet another item.

with the following code:

```
\begin{itemize}
\item An item.
\item Another item, with subitems.
  \begin{enumerate}
  \item First item.
  \item Second item, with subitems.
  \end{enumerate}
\item Yet another item.
\end{itemize}
```
4 Theorem-like environments

\LaTeX was designed from the ground up to be very good at typesetting mathematical documents. Other than typesetting equations (discussed in the Math Typesetting Guide), perhaps the most important feature of mathematical documents is that they have lots of theorems! Naturally, \LaTeX handles these nicely (with the help of the `amsthm` package, included in the Carleton styles by default).

To typeset a theorem in \LaTeX, just use the `theorem` environment. For example, we can typeset the following famous theorem of number theory:

**Theorem 4.1** (Euclid). *There are infinitely many prime numbers.*

using the following code:

\begin{theorem}{euclid}
There are infinitely many prime numbers.
\end{theorem}

(The argument `[Euclid]` is optional; if you don’t want to have such a citation on your theorem, just omit the square braces entirely.)

You can create many kinds of “theorem-like” environments using the `\newtheorem{}` command. However, for convenience, the Carleton styles set up several common ones for you: `theorem`, `corollary`, `lemma`, `proposition`, `observation`, `definition`, `remark`, `example`, and `note`.

4.1 Starred theorems

All of the theorem-like environments above also have starred equivalents (such as `theorem*`), which are not numbered.

4.2 Proofs

\LaTeX and `amsthm` also provide the `proof` environment for typesetting proofs.

For example, we can produce the following staple of math textbooks:

**Proof.** Obvious.

using the following code:

\begin{proof}
Obvious.
\end{proof}

Note that the QED symbol is automatically added by the `proof` environment—you don’t need to do anything to get it! However, sometimes things will go wrong with the placement of this symbol, especially if the last line of your `proof` is an equation. To fix this, just put the `\qedsymbol` command at the end of the equation.

5 References

As discussed in Sections 2 to 4, \LaTeX can automatically number your sections, lists, and theorems. Fortunately, it can also automatically number your cross-references, so you don’t have to worry about things getting out of sync.

The first step in setting up a cross-reference is to give the thing in question a name using the `\label{}` command. For example, at the start of this section, I typed the following:

\section{References}
\label{s:refs}
The first line, of course, started the section, so \LaTeX assigned it a number. The second line then tells \LaTeX that I want to refer to this section with the code \texttt{s:refs}. The choice of label is completely up to you; I like to use prefixes that indicate what kind of thing the label points to (such as the \texttt{s:} before).

Once you’ve labeled something, it’s easy to set up a reference. To print the number, you merely need to use the \texttt{\ref{}} command. For example, the following code:

\begin{verbatim}
\texttt{See Section \ref{s:refs} for more.}
\end{verbatim}

produces the following output:

\begin{verbatim}
\texttt{See Section 5 for more.}
\end{verbatim}

This will continue to work even if you reorder your sections, change the numbering scheme (to use roman numerals, for example), or even change Section 5 to a subsection!

### 5.1 Automatic references with \texttt{cleveref}

Of course, if you were to change Section 5 to a subsection, the word “Section” in the previous example wouldn’t change. This may sound like a small caveat, but when you’re restructuring a large, complicated document like a comps paper, this can turn into quite a pain. Fortunately, there is a solution!

The \texttt{cleveref} package (included in the Carleton styles by default) adds facilities for automatic \texttt{naming} of references, as well as some other handy features. Using \texttt{cleveref}, we could obtain the same output as before using this code, instead:

\begin{verbatim}
\texttt{See \cref{s:refs} for more.}
\end{verbatim}

cleveref automatically adds the word “Section” for you.

Additionally, if you want to refer to several things in your document at once, \texttt{cleveref} can automatically handle aggregating them into a nice form. For example, the following code:

\begin{verbatim}
\texttt{As discussed in \cref{s:unordlist,s:ordlist,s:desclists,thm:euclid},}
\end{verbatim}

produces the following output:

\begin{verbatim}
\texttt{As discussed in Sections 3.1 to 3.3 and Theorem 4.1,}
\end{verbatim}

which is much simpler and more maintainable than the equivalent reference written out by hand.

### 6 Tables and floats

#### 6.1 Tables

\LaTeX also has support for producing aligned tables (like Tables 1 to 3 in this documents). With the help of the \texttt{booktabs} package (which the Carleton styles include by default), it can produce very high-quality output.

To create a table, use the \texttt{tabular} environment. It takes one mandatory argument, which specifies the number and layout of the columns of the table. Each column may be either left-aligned (l), centered (c), or right-aligned (r); to describe the layout of your table, just give the appropriate letter for each column in order. You can then give the content of the table line-by-line, separating columns with ampersands (\&) and rows with line breaks (\\). To give the table a professional appearance, use the horizontal “rules” (lines) provided by \texttt{booktabs}: a \texttt{\toprule} before the first line, a \texttt{\midrule} after the last line of the header, and a \texttt{\bottomrule} after the last line of the table.

For example, the following code:

\begin{verbatim}
\begin{tabular}{c l r}
\toprule
center & Left & Right \\
\midrule
1 & 2 & 3 \\
\bottomrule
\end{tabular}
\end{verbatim}

\begin{verbatim}
\texttt{produces the following output:}
\end{verbatim}

\begin{verbatim}
\texttt{| Center | Left | Right |
\midrule
| 1 | 2 | 3 |
\bottomrule}
\end{verbatim}
produces the following output:

<table>
<thead>
<tr>
<th>Center</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

\(\text{T}_{\text{e}}\text{X} \) does provide mechanisms for including vertical rules, but these are not commonly used by professional typographers and should usually be omitted. (They make tables look very busy, and don’t actually help with readability!)

6.2 Floats

Once a table (or figure, or code listing) has been produced, \(\text{T}_{\text{e}}\text{X} \) has a variety of facilities for controlling their behavior as “floats”, which provide captions, centering, and placement.

By the end of this section, we’ll have turned the table from Section 6.1 into Table 4. Pretty slick, eh?

Table 4: An example table

<table>
<thead>
<tr>
<th>Center</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

6.2.1 Float placement

To place a table in a float, enclose it in the \texttt{table} environment. (There are other float environments for other purposes; we’ll talk about them later, but they work the same way.) This takes one optional argument, a “float specifier”, which tells \(\text{T}_{\text{e}}\text{X} \) how you would like it to place the float. This is a sequence, in order, of characters, each of which corresponds to something \(\text{T}_{\text{e}}\text{X} \) is allowed to try; it will select the first of these which it can follow without messing things up too badly. The available specifiers are described in Table 5.

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Place the float near the place it appears in the text.</td>
</tr>
<tr>
<td>t</td>
<td>Place the float at the top of a page.</td>
</tr>
<tr>
<td>b</td>
<td>Place the float at the bottom of a page.</td>
</tr>
<tr>
<td>p</td>
<td>Place the float on a special page just for floats.</td>
</tr>
</tbody>
</table>

Table 5: Float specifiers in \(\text{T}_{\text{e}}\text{X} \)

If you’re used to working with a WYSIWIG word processor, this may seem strange—why can’t you just make \(\text{T}_{\text{e}}\text{X} \) put that table where you want? This is one of the cases where you’ll be much happier if you just let \(\text{T}_{\text{e}}\text{X} \) do its thing; because of the way \(\text{T}_{\text{e}}\text{X} \) handles page formatting, it just isn’t realistic to hard-code all the table locations. \(\text{T}_{\text{e}}\text{X} \) will do a good job of finding placements that make sense if you give it reasonable options; the specifier \texttt{htb} is a reasonable default that will work well in most cases.

6.2.2 Captions

Since floating tables may appear some distance from the text where they are introduced, they should have captions that explain their contents. This is easy to do in \(\text{T}_{\text{e}}\text{X} \); just use the \texttt{\caption{}} command inside the \texttt{table} environment.
6.2.3 Centering

By default, the contents of the floating table will still be left-justified. This probably isn’t what you want! To fix this, use the \centering{} command at the beginning of the table environment.

6.2.4 Labels and references

To label your float for cross-references, use the \label{} command, just as for sections and theorems.

6.2.5 Bringing it together

Now, let’s combine all of this to produce a full, working float table. To produce Table 4, we use the following code:

\begin{table}{htb}
  \centering{}
  \begin{tabular}{c l r}
  \toprule
  center & Left & Right \\
  \midrule
  1 & 2 & 3 \\
  a & b & c \\
  \bottomrule
  \end{tabular}
  \caption{An example table}
  \label{tab:example}
\end{table}

Some important features of this code:

- It uses the htb float specifier.
- It uses the \centering{} command to center the table on the page.
- It uses the c l r column specifier. Thus, the first column is centered, the second column is left-aligned, and the third column is right-aligned.
- It uses \toprule, \midrule, and \bottomrule to produce the horizontal rules.
- It has a \caption{}.
- It has a \label{}.

6.3 Images

$\LaTeX$ also has facilities for including graphics in your documents. These are provided by the graphicx package, which is included by default in the Carleton styles.

The core command for using graphics in a $\LaTeX$ document is \includegraphics{}. It takes one mandatory argument, which is the name of an image file to be included without its extension. Important: graphicx only supports the JPEG, PNG, and PDF formats for included graphics. You must convert your image to one of these formats before using it!

The \includegraphics{} command also takes a variety of optional arguments. Perhaps most importantly, these can be used to specify the size of the image on the page. Setting either height=x or width=x will scale the image while keeping its shape (aspect ratio); setting both will force the aspect ratio.

For example, if there is a file named logo.png in the same directory as this $\LaTeX$ file, the following command:

\includegraphics[width=.5cm]{logo}
results in the following image:  

In practice, of course, most included images should be in floats. To do this, use the `figure` environment, which works just like the `table` environment from Section 6.2.

For example, the following code:

```latex
\begin{figure}{htb}
  \centering
  \includegraphics[width=2in]{logo}
  \caption{The logo of the SVG file format}
  \label{fig:svg}
\end{figure}
```

produces Fig. 1.

![SVG logo](logo)

Figure 1: The logo of the SVG file format

### 6.4 Inserting space

Occasionally, you’ll need to force \LaTeX{} to include extra vertical space in your document. The most common use case for this is to include a blank area to hand-draw a diagram. This is easily done with the \texttt{vspace*{}} command. For example, to add a vertical blank space one inch high, we simply call

\texttt{\vspace*{1in}}

Which results in the desired space:

![Vertical space example](vspace.png)

### 7 Defining commands

Often, when working with a document, you’ll find yourself typing the same code over and over again to instruct \LaTeX{} how to format some important bit of text. For example, in a problem set for a complex analysis course, you will use the symbol $\mathbb{C}$ quite a lot. Similarly, in this document, we frequently typeset bits of code using the monospace font. We could just type \texttt{} over and over, but this is un-semantic—it tells \LaTeX{} what to print, rather than describing the content—and fragile—it’s a huge pain if you decide to change how these things are formatted, because you’ll need to replace many different instances.

To overcome these difficulties, \LaTeX{} has facilities for defining your own commands.
7.1 Commands without arguments

The simplest kind of command to define in \LaTeX{} is one that does not take any arguments. For example, you might want to define a new command \texttt{\textbackslash cc} that automatically typesets \$\mathbb{C}\$ for you. This will make it easier to produce the symbol \$\mathbb{C}\$ where you need it, and will allow you to instantly change all instances of it to \texttt{C} if you suddenly decide you prefer that.

To do this, go to the “preamble” of your document (i.e. the part after \texttt{\documentclass{}} and before \texttt{\begin{document}}). The basic command you’ll need is \texttt{\newcommand{}} which, naturally, tells \LaTeX{} about a new command. To create a new command which defines \texttt{\textbackslash cc} to produce \$\mathbb{C}\$, add the following code:

\begin{verbatim}
\newcommand{\textbackslash cc}{\ensuremath{\mathbb{c}}}\end{verbatim}

The first argument, \texttt{\textbackslash cc}, is the name of the new command. The second argument is the code that will be substituted whenever \texttt{\textbackslash cc} appears in the document. (The \texttt{\ensuremath{}} command tells \LaTeX{} to be sure that the \texttt{\mathbb{c}} is set in math mode; this allows you to use the command in either text or math mode.)

7.2 Commands with arguments

You can also define your own commands that take arguments. For example, for all the code snippets in this document, we’d like to define a \texttt{\textbackslash code{}} command which typesets its argument as code (using the monospace typeface). To do this, add the following code to your preamble:

\begin{verbatim}
\newcommand{\textbackslash code}[1]{\texttt{#1}}\end{verbatim}

Just like before, the first argument (\texttt{\textbackslash code}) is the name of the new command. Immediately after this, though, comes something new—an \textit{optional argument} \texttt{[1]} which tells \LaTeX{} that the command will accept one argument. The command definition \texttt{\textbacktt{#1}} will then be filled in by substituting the argument of \texttt{\textbackslash code{}} for the # in the definition.

For example, if we type \texttt{\textbackslash code{test}}, \LaTeX{} transforms this into \texttt{\textbacktt{test}}, which is then rendered in the document as “test”.

You can define commands that take up to nine arguments, which will be given names \#1, \#2, \ldots, \#9.