Course A, Part 1
Basic Formatting in LaTeX

As you become acquainted with LaTeX, you must remember that this is *not* a piece of word processing software. Neither is it a programming language. Specifically, LaTeX is a *mark-up language*. The LaTeX compiler takes code from a raw text file and formats it to the specifications given by the source document. The file that you use to create the finished product will *not* resemble the desired output. The whole process is somewhat similar to a web browser displaying HTML code.

Before we get started, there is one important axiom of typesetting with LaTeX. This applies to every LaTeX user.

> **LaTeX User's Axiom:** LaTeX *can do whatever you want it to do, but you may have to learn how to do it on your own.*

The Typical File Format

The name of every LaTeX source file must end with the `.tex` extension. For reasons I'll explain later, you probably shouldn't use spaces in the name of the file.

Every LaTeX source file consists of two components: the *header* (or "preamble") and the *body*. The header contains formatting information that applies to the *entire* final product. This includes physical features of the document such as

- Font size
- Margins
- Line spacing
- Page numbering style
- Chapter/section heading and numbering style
- Definition/theorem/proof numbering style and appearance
- Universal user-defined shortcuts.

The header is also the place to call up any specialized *packages* that you'll need to typeset your document. As we're just getting started we do not need to know too much about headers now, but this is definitely something you will need to become acquainted with later on.

The body contains the code that LaTeX will attempt to typeset and interpret as a readable document. The start of the body is indicated with `\begin{document}` and the end of the body is indicated with (conveniently enough) `\end{document}`. This should be the final line in your LaTeX source file.

NOTE: Compiling a LaTeX document creates several auxiliary files. Most of these are useful only for advanced users, so you can safely ignore them. However, there is no way to prevent them from being created. For this reason, you are strongly advised to put every LaTeX source file *in its own folder*. This will quarantine the spreading of these extra files throughout all parts of your computer.

Comments
It is sometimes helpful to place little notes to yourself in the code. Naturally, you do not want the compiler to see these. To do this, place a percent sign to the left of your comment. For example, the line

```
% ignore this line
```

will *not* be seen by LaTeX, but *will* be seen by you.

Command Format
Every LaTeX command begins with a \ (backslash). Some commands have a *parameter*, and some do not. If a command has a parameter, it is (almost) always included in curly braces. For example, the `\vspace` command creates vertical blank space in your document. To create a half inch of vertical space, you would code `\vspace{0.5in}`. Note that the parameter (0.5 inches) is inside curly braces.

Reserved Characters
Several characters are used by LaTeX in specifying certain commands. Examples off the top of my head are

$$\$,\ \&,\ \#,\ \hat{}\ ,\ \_\ ,\ \%, \{\ ,\ \}.$$

I'm sure there are others that I am not remembering. Because these are reserved, they cannot be typeset with a single keystroke. Rather, each must be preceded with a backslash. For example, coding

```
20\% of \$50 is \$10 \& that's the truth.
```

produces

20% of $50 is $10 & that's the truth.

Controlling Physical Properties of Your Text
So how does one handle spacing, line breaks, boldfacing, and general text layout? Again, you must remember: LaTeX is *not* a word processor.

NOTE: The remarks in this section apply *only to text*, not mathematical content!

Line breaks in your source code will *not* correspond to line breaks in the finished product. LaTeX decides when to break lines (i.e. "wrap words"), and for the most part it does so correctly. If you need to force a line break, there are two ways of doing so. One is to simply code `\linebreak`. The other is to type two backslashes in succession (that is, \\). The latter is my preferred method.

LaTeX also decides when to break pages. If you do not like the way it chooses to break a particular page, just place `\pagebreak` where you would like the new page to begin.

In general, *LaTeX is better at deciding when to break lines and pages than you are!* Before forcing breaks, I usually mull it over first to make sure that LaTeX is "wrong" before proceeding.

LaTeX sets a new paragraph every time there is a blank line in your code.[1] Whether or not this indents depends on the `documentclass` declaration in the header. Most document classes indent

[1] If you put 10 blank lines between sentences, LaTeX does *not* set additional blank space. LaTeX is *not* a word processor!

by default. If you do *not* want a new paragraph to be indented, you need only code `\noindent` at the beginning of the first sentence of the new paragraph. As a matter of personal taste, I prefer block style paragraphs (no indenting at all). To prevent any and all indenting (without having to code `\noindent` every time), simply place `\setlength{\parindent}{0pt}` in the header.

Text Effects
Since LaTeX is not a word processor, you have to force any horizontal or vertical white space that you desire. For example, coding `\hspace{1.5in}` after the word "suddenly" has the following effect:

This sentence will suddenly have 1.5 inches of blank space.

Coding vertical space works the same way with the `\vspace{}` command.[2] A really clean way of getting uniform vertical space between paragraphs is to use `\medskip`.

Italicizing is easy. The command is `\emph{}` for "emphasis." For example, by coding `\emph{not}` we get

LaTeX is *not* a word processor.

You can also use `\textit{}` for the same effect, but this is inferior to `\emph{}` for subtle reasons.

Boldfacing and underlining work in a similar fashion. The commands are `\textbf{}` and `\underline{}` respectively. For example, we use `\textbf{How many times}` and `\underline{not}` to achieve

How many times do I have to tell you that LaTeX is not a word processor?

Centering is also easy. The centering process begins with `\begin{center}` and ends with `\end{center}`. Everything in between these lines will be centered. For example, in order to get

Headings look good when centered.
You can also *italicize* inside a centered piece.
Pretty neat, huh?

I coded the following:

```
\begin{center}
Headings look good when centered.\\
You can also \emph{italicize} inside a centered piece.\\
Pretty neat, huh?
\end{center}
```

(Note that I had to force some line breaks.)

So, have you been noticing my little footnotes?[3] Looks pretty nice, right? Of course, it's easy to do: just type `\footnote{my footnote text}` at the exact location where you want the footnote citation to occur. LaTeX does the rest, including numbering them automatically!

[2] You don't have to use inches. You can use almost any measurement, including points! (1 inch $\approx$ 72 pt)
[3] Like this one.

Making Lists
This can be incredibly useful, and it's also incredibly easy. There are basically two kinds of lists: enumerated and bulleted.

To make a bulleted list, use `\begin{itemize}` and declare new items with the `\item` declaration. In order to get

- Step 1: Take the derivative.
- Step 2: Set it equal to zero.

I coded

```
\begin{itemize}
\item  Step 1:  Take the derivative.
\item  Step 2:  Set it equal to zero.
\end{itemize}
```

If you want an enumerated list instead, just use `\begin{enumerate}`. So the code

```
\begin{enumerate}
\item  Take the derivative.
\item  Set it equal to zero.
\end{enumerate}
```

produces

1. Take the derivative.
2. Set it equal to zero.

(The enumeration is done automatically.) You can also enumerate manually by placing your item tags in closed brackets. For example, to enumerate with lower case letters, one might code

```
\begin{enumerate}
\item[(a)]  Take the derivative.
\item[(b)]  Set it equal to zero.
\end{enumerate}
```

producing

(a) Take the derivative.

(b) Set it equal to zero.

You can also do lists within lists within lists... and LaTeX keeps track of it all. This is handy for producing, say, outlines.

Basic Mathematics Formatting

When you need to code mathematics, you must be in the *mathematics environment*. Math-oriented commands will not compile correctly if you are not in the math environment. There are two ways that LaTeX recognizes this mode. The most common way is to begin and end a mathematical expression with a dollar sign ($). This is what you do if you are placing some mathematics in a sentence (this is called "in-line"). If you wish to have a mathematical expression on a line by itself, you should start it with `\[` and end it with `\]` (this is called "display style"). This will have the bonus effect of centering the mathematical expression.

If you ever need some extra space between symbols in the math environment, you need only code a single backslash. This will act as a buffer between symbols, as long as it is not "touching" anything else.

The commands that generate mathematical symbols are fairly user-friendly and sensible. Any good LaTeX editor will have tool bars that display the code for commonly used symbols. It is impossible for me to give you a list of *all* of the symbols that you will ever need. Below, I have provided you with some of the more common symbols, but even this list will be incomplete. To discover how to code other symbols, you should apply the LaTeX user's axiom.

To form an exponent, use `^`. We code `$x^2$` to get x^2. However, `$x^23$` produces x^23. To obtain x^{23} we must use `$x^{23}$`. Thus, without curly braces, LaTeX expects that an exponent will consist of *only one* character.

Subscripts work under same rules, but you use an underscore `_` to create them. Thus `$x_23$` produces x_23 while `$x_{23}$` produces x_{23}.

To create fractions, we code `$\frac{}{}$`, where the first parameter is the numerator and the second is the denominator. So `$\frac{x^2}{y}$` produces $\frac{x^2}{y}$. LaTeX goes through great efforts to make everything fit "in-line," so it makes that fraction small. If you thought that made things a bit cluttered, you can use `\displaystyle` to enlarge it. For example, coding

`$\displaystyle \frac{x^2}{y}$` produces $\displaystyle \frac{x^2}{y}$. Notice that now there is a bit of uneven space between this and the previous two lines. You can't get something for nothing.

`$\leq$` produces $\leq$. Similarly, `$\geq$` produces $\geq$. If you want $<$ or $>$, you can find them on your keyboard!

Type `$\neq$` to create $\neq$.

Creating roots is easy. The string `$\sqrt{5}$` creates $\sqrt{5}$. What if you wanted the cube root of 5? This is done with `$\sqrt[3]{5}$`, producing $\sqrt[3]{5}$. Other roots are handled in the same way.

Common functions have obvious codes. For example, `$\sin$` creates sin. So `$\sin x$` produces $\sin x$.

Greek letters are coded by their names. Uppercase codes create uppercase Greek letters, while lowercase codes create lowercase Greek letters. For example, `$\Delta$` creates Δ while `$\delta$` creates δ.

Symbols like $\mathbb{R}$ are in what is called *blackboard font*. Such symbols are created with `$\mathbb{}$`. Thus `$\mathbb{Q}$` produces $\mathbb{Q}$, `$\mathbb{C}$` produces $\mathbb{C}$, etc.

The symbol for set membership is coded `$\in$`. Thus, to get $x \in A$ you would type `$x \in A$`.

The symbol for subset containment is coded `$\subseteq$`. So `$\mathbb{N} \subseteq \mathbb{Z}$` typesets the true statement $\mathbb{N} \subseteq \mathbb{Z}$. To make this a *proper* containment, code `$\mathbb{N} \subset \mathbb{Z}$`, yielding $\mathbb{N} \subset \mathbb{Z}$.

As a quick summary, the (long) string
`$e^{i \theta} = \cos \theta + i \sin \theta$ for all $\theta \in \mathbb{R}$`
creates "$e^{i\theta} = \cos\theta + i\sin\theta$ for all $\theta \in \mathbb{R}$" (without the quotes of course). If you wanted to set off a portion of this, as in

$$e^{i\theta} = \cos\theta + i\sin\theta$$

you would code the following:

```
\[
e^{i \theta} = \cos \theta + i \sin \theta
\]
```

Note that when you set something off this way, you do *not* need dollar signs. Of course, another way to achieve the above is to use the `\begin{center}` command and then use dollar signs inside of this. This is usually a bad way of doing things: the spacing will often be off. Use the above method instead.

A word of warning about curly braces: these symbols are reserved by LaTeX to be used as a grouping device for commands. So what do you do if you actually need to typeset some braces, as in $\{1, 2, 3\}$? The open brace is coded as `\{` and the closed brace is coded as `\}`. So to obtain $\{1, 2, 3\}$ we must type `$\{1, 2, 3\}$`.

As a final review, we produce

$$T = \{a + b\sqrt{5} \mid a, b \in \mathbb{Z}\}$$

by coding

```
\[
T = \{a + b\sqrt{5} \ | \ a,b \in \mathbb{Z}\}
\]
```

Note that I used some backslashes to add some extra padding around the vertical "such that" bar. If I didn't do this, the output would have been $T = \{a + b\sqrt{5}|a, b \in \mathbb{Z}\}$, which I feel is a bit cluttered.

Final Remarks

This is certainly not all that you need to know. Next time, we'll talk about deeper issues concerning formatting mathematical expressions. We'll also examine headers in more detail so that you have greater freedom in customizing your documents.

If you want a good reference, George Grätzer's book *Math Into LaTeX* is probably the best I've ever seen. Plenty of online resources exist too, most notably the TeX user's group (TUG) at `www.tug.org` or CTAN (the Comprehensive TeX Archive Network) at `www.ctan.org`.

Learning LaTeX requires some patience, but it will pay off tenfold. You have to trust me on this. Yes, you will certainly struggle at first. Frustration will occur. This happens to everyone, but everyone overcomes it (and rather quickly at that). If you put in the time to become comfortable with LaTeX, you will find that you have gained the ability to create professional-grade documents with relatively little effort.

Bonus Tips!

Managing Packages

There will be numerous instances when LaTeX will need additional macros to process your code. How does one acquire these additional packages? If you are connected to the internet while compiling, much of this is automatic. Often, MiKTeX will realize that you need something and prompt you to allow it to install the package for you.

Other times you may need to install a package *before* you attempt to assemble the code. MiKTeX handles this very nicely as well. Here's how:

- Open the MiKTeX package manager by following

  Start ⇒ Programs ⇒ MiKTeX ⇒ MiKTeX Package Manager.

- The manager will figure out which packages are available and which ones you already possess. It even gives the dates on which the packages were installed.
- Find the package that you want and click the plus sign in the upper left of the window. Follow the instructions.
- Finally, you need to tell your system that you now have a new package that it can freely use. Do this by following

  Start ⇒ Programs ⇒ MiKTeX ⇒ MiKTeX Options ⇒ Refresh file name database.

Backwards Searching

The ability to perform a backwards search is incredibly handy. I only know how to do this through the yap previewer. Sorry, Mac users.

In order to do this, you first need the package that gives you this capability. This package is called `srcltx` (for "source LaTeX"). Use the MiKTeX package manager to see if you already have this. If not, follow the steps above to install it. You are now *almost* ready to perform backwards searches.

You have to inform the yap previewer that you want to go backwards into your WinEdt file. This is easy. Open your yap previewer and follow

View ⇒ Options ⇒ Inverse Search ⇒ Program

and then select WinEdt as the go-to program. You're done!

Managing Margins

Because it is so customizable, setting margins in LaTeX is notoriously intricate. A package has been developed to circumvent all the associated difficulties and subtleties. It is called `geometry` and I highly recommend it. If you'd like to use it, find it and install it using the MiKTeX package manager described above. To set standard margins, just put the line

```
\usepackage[margin=1in, letterpaper]{geometry}
```

in your header. Of course, you should first remove any previous margin-related settings in your header to avoid conflicts with the `geometry` package.