LaTeX for Math and Science

Cordelia Csar
Alan Leong
Thomson Nguyen
Dan Volmar

September 16, 2009

Contents

Introduction

LaTeX dominates academic publishing in science and mathematics. If you hope for a career in any discipline that requires the production of technical documents, you must learn to love LaTeX. Fortunately, there are good reasons for this ubiquitousness. While not as friendly as the cartoon paperclip, LaTeX is not at all difficult to use. A little practice is all that is needed to produce textbook-quality mathematical documents. It will run on any modern computer system and is well supported by a community of developers. It can be extended to satisfy a wide variety of publishing needs, and a vast repository of such user-made packages are available on the Internet. Perhaps most importantly for the academic world, LaTeX is free. Before long, you might decide to junk Microsoft Word and the stupid paperclip and do all of your word processing with LaTeX.

0.1 Historical nonsense

The TeX language was written by Stanford computer scientist Donald Knuth in the late 1970s. Knuth is best known for *The Art of Computer Programming* books which are of biblical importance in academic computer science. Upset by the poor quality of the typesetting in the published editions of his books, Knuth decided that he could do better. The first version of TeX appeared in 1978 and was compiled to run on a mainframe computer. Although it began as a research project, the mathematics community became interested in TeX because it promised to be a cheap solution for academic publishing. The American Mathematical Society even sponsored their own TeX implementation.

In addition to TeX, Knuth also wrote Metafont, a language for producing document fonts from mathematical curves. This system, and its successor MetaPost, are still used for generating fonts and figures for technical documents.

The TeX language itself is rather primitive. You could typeset an entire document in pure TeX if you wanted to, but you could also drive on the wrong side of the road. Donald Knuth wrote what is called Plain TeX, a set of higher-level commands written in simpler TeX commands, to make things easier. However, even in Plain TeX, the author has to do more work setting the type than actually composing the document. Various languages derived from TeX have been developed to be more functional than Plain TeX. Of these, the most popular is LaTeX, which was written in 1984 by the research computer scientist Leslie Lamport. Thus, LaTeX is not a software package like Microsoft Office or even a stand-alone computer language. Rather, it is a macro language – a set of commands written in TeX. In practice, it functions like a typical markup language such as HTML. Development on LaTeX still continues: the current version is called LaTeX2ϵ and small updates are made to the system about every six months.

0.2 The strange name

The funny typesetting of the word TeX is supposed to be a rendition of the Greek letters $\tau\epsilon\chi$, which are transliterated as *tex*. It should be pronounced "tekh" where "kh" indicates a voiceless velar fricative (as a German would say "J. S. Bach"). However, most lazy Americans simply say "tek". The name is in honor of Caltech, where Donald Knuth did his graduate work. A popular folk etymology is that the name came from Knuth's verbal reaction to the lousy typesetting of the second volume of the *The Art of Computer Programming*: blech.

LaTeX is for Lamport TeX, after the initial author Leslie Lamport. It is usually pronounced "lay-tek(h)", although some say "lah-tek(h)".

0.3 The LaTeX system

As was painfully explained above, LaTeX itself is a document langauge. More specifically, it is a macro package for the TeX language. Just like all computer languages, it must either be compiled or interpreted in order to be of any use. The actual document preparation takes place in a text editor, such as emacs or vi which are common on UNIX and new MacOS operating systems. You could use DOS EDIT or Windows Notepad if you really wanted to. A usable document usually carries a .tex extension and is an ordinary ASCII text file.

In general terms, a compiler is a software program that translates a human-readable source file into machine code that can be executed by a computer. Similarly, the LaTeX compiler translates a .tex file into a machine-readable format. This might be a printer language such as Postscript so that a document can be printed, or into Portable Document Format (PDF) to be shared over the Internet. The output of the standard LaTeX compiler is what is called DVI format (for "device independent"). A .dvi file will appear identical on any kind of printer or computer display. Compilers have also been written to output a .tex document directly into Postscript, PDF, or even HTML. It is also possible to convert from DVI to these formats. However, it is not usually possible to recover a .tex document from a compiled document. Thus it is important to keep the source files around if further editing is anticipated.

LaTeX itself is fairly comprehensive and can accommodate a wide variety of publishing needs. It can, however, be extended through the use of user-defined packages. A package is an independent source file that adds features to the language by defining them in terms of existing LaTeX commands. An author can then refer to a package in a document and make use of these new features. Packages can greatly simplify document creation by providing additional tools that would take time and/or extensive programming knowledge to implement. The American Mathematical Society provides packages that extend on the already rich mathematical typesetting ability of LaTeX. This system is known as AMS-LaTeX. Also available are packages that add support for new types of documents, graphics, illustrations, and publishing in different languages.

BibTeX is another important extension to TeX and LaTeX. BibTeX automates the creation of bibliographies and textual citations. A separate .bib file is used to store the publication data for books, journals, and other articles that can be referred to by a TeX document. An author can create a list of commonly used citations and quickly refer to them in any document. When compiled, BibTeX automates the formatting of citations and generates the bibliography. A large number of BibTeX packages are available to handle different academic, legal, and professional citation styles.

Fonts for use with LaTeX are usually written in MetaFont or MetaPost and then compiled. Since fonts are defined by geometric curves (the curious can google "Bézier curve") they can be extremely versatile. The same font can be used on any computer system for which a MetaFont or MetaPost compiler is available. Often technical illustrations are written in MetaPost. Upon execution, the LaTeX compiler will reference the fonts that it needs and render them appropriately.

The advantage of this approach is that LaTeX is not married to any specific operating platform. A .tex document can be written on any computer system with a text editor. It is only necessary to provide the compiler for a given operating system. With a bit of work, LaTeX will function on anything that you can connect a monitor or a printer to and will produce identical output.

However, compiling and configuring a complete LaTeX system is a demanding and sometimes painful undertaking. To remedy this, some people have been kind enough to pre-package all of the necessary software and automate the process. These complete, ready-to-use LaTeX systems are called distributions. MikTeX is the most popular for Windows, TeTeX is a common distribution on UNIX and MacOS X, and TeX Live is a new effort to make LaTeX even easier to get running on Windows, UNIX, and Mac.

While there are diehards who would rather die than stop using emacs, some people feel that a text editor is not the most productive environment for composing LaTeX documents. To this end, software developers have produced a number of LaTeX front-ends. These are graphical programs that may incorporate some degree of WYSIWYG ("what you see is what you get") functionality like a typical word processor. LyX and TeXmacs are pseudo-WYSIWYG front-ends that run on most platforms. Other front-ends are simply text editors with some graphical features like syntax highlighting, toolbars for common commands, and project management. These usually integrate with an existing LaTeX distribution and link to the LaTeX compiler. WinEdt and TeXnicCenter are popular front-ends for Windows, TEXShop for Mac, and there are many implementations for UNIX systems. For those that still can't get enough of their UNIX terminal, AUCTEX adds additional TEX functionality to emacs.

0.4 Documentation and the TEX community

Because there are so many different components that make up a LATEX system, documentation is unfortunately very scattered. The information that you need may not be part of LATEX itself, but a specific package or perhaps your distribution. The various distributions are usually very good about providing documentation to the packages that they support. However, there are so many packages available that a problem might only be solved by an appeal to the larger TEX community. The Comprehensive TEX Archive Network (http://www.ctan.org) and the TEX Users Group (http://www.tug.org) are the two largest Internet communities that provide repositories of TEX packages and documentation. The benefit of the collaborative nature of LATEX is its extreme versatility, but there is a price to be paid. There is no stupid cartoon paperclip to answer your questions. Fortunately, the answers are available to those willing to make a little effort.

Chapter 1

Typesetting Text

As you'll soon find out (if you haven't already), LaTeX is different from other word processors in that you have to input both logical and semantic structures of a text. Only then will it derive a text document that looks remotely like what you intended it to be. In this chapter, you will learn the basic structure of every LaTeX document in addition to simple formatting commands in LaTeX that you can do. We begin with the syntactic structure of basic LaTeX documents.

1.1 Structure

1.1.1 Command Structure

Commands in LaTeX have a very intuititve and flexible structure, not unlike most programming languages. Most of the commands that you will deal with in LaTeX will look like this:

```
\backslash{argument}
```

Where the name of the command replaces `command`, and any argument or modifers you have replace `argument`. Some examples include `{\texttt{Text goes here}`, `\textit{Text goes here}`, and `\textbf{Text goes here}`. The first example will make text a cool typewriter font, while the second and third examples will make your text *italic* and **bold**, respectively. Note that all commands start with a forward slash.

1.1.2 Environment Structure

Environments are special commands that modify large blocks of text. Environments almost always start with `\begin{environment}`, and end with `\end{environment}`, where the name of the environment replaces `environment`. For example, here is an environment called **verbatim**:

```
\begin{texttt}

This is typewriter text

\end{texttt}


\begin{document}

Document goes here.

\end{document}
```

This will make the text in between the `\begin{texttt}` and `\end{texttt}` a typewriter font.

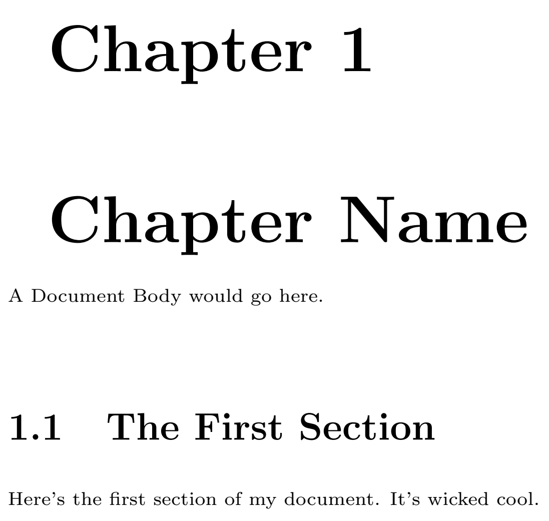
1.1.3 Document Structure

Here's a sample document that outlines the basic structure LaTeX:

```
\documentclass[12pt]{article}

\usepackage{amsmath}

\title{Some Really Important Results}

\author{My Name}


\begin{document}

\maketitle


\chapter{Chapter Name}

A Document Body would go here.


\section{The First Section}


Here's the first section of my document. It's wicked cool.
```

```
\section{The Second Section}
Here's the second section of my document. This is twice as cool as the previous one.

\subsection{A Subsection}
Sometimes things aren't important enough to get their own section, so here's a subsection.

\subsubsection{Subsubsections Exist?}
I guess they do.

\paragraph{Paragraph Title}
Why paragraph have titles, I have no idea.

\section*{The Third Section}
This is the third section of my document, but this will show without a number.

\end{document}
```

When compiled, it'll look something like this:



Chapter 1

Chapter Name

A Document Body would go here.

1.1 The First Section

Here's the first section of my document. It's wicked cool.

1.2 The Second Section

Here's the second section of my document. This is twice as cool as the previous one.

1.2.1 A Subsection

Sometimes things aren't important enough to get their own section, so here's a subsection.

Subsubsections Exist?

I guess they do.

Paragraph Title Why paragraphs have titles, I have no idea.

The Third Section

This is the third section of my document, but this will show without a number.

1.1.1 Preamble

One can see that before the actual document body, there's some weird text at the beginning of our sample document. This is called the **preamble** of the document, and tells LaTeX exactly how you want your document to be structured. Our sample document uses the `article`document class, at 12pt font. The title of your document goes in between the braces in the command `\title{}`, while the author name(s) go in `\author{}`. You'll see that one package has been declared, `amsmath`; packages and $\mathcal{AMS}$math will be covered in later chapters, so we'll just ignore it for now. The preamble is ended with the command `\begin{document}`, which begins our document. One warning: In order for our title to display in LaTeX, the command `\maketitle`must be used *after* `\begin{document}`has been declared.

1.1.2 Body

In most LaTeX documents, you'll want to split your text up into discrete sections or parts in order to make reading your document accessible and easy to your readers. LaTeX makes this easy with special commands that are placed at strategic places in your document. To reiterate, here's a table of commands available:

`\chapter{...}`	Starts a numbered chapter (Chapter 1)
`\section{...}`	Starts a numbered section (Section 1.1)
`\subsection{...}`	Starts a numbered subsection (Subsection 1.1.1)
`\paragraph{...}`	Starts a paragraph (w/o numbering)
`\subparagraph{...}`	Starts a subparagraph (w/o numbering)
`\part{...}`	Starts a part (w/o numbering)

The mechanics of these commands will be explained later, but for now, it'll suffice to say that each successive command creates a smaller title, similar to the chapters and sections you see in your usual textbook. It should be noted that the `\chapter`command is available only in the `report`or `book`class.

Note: If you want any of the numbered commands without the numbers (i.e. unmarked chapters), simply add a * at the end of the command. For example, `\section*{Section}`will yield:

Section

instead of

1.1 Section

1.2 Formatting

1.2.1 Emphasizing Words

Just like any word processor, you can <u>underline</u> and *italicize* words whenever you see fit. Underlining is done with the command `\underline{...}`, where the underlined text goes between the braces. Italicization is done with `\textit{...}`, with the desired italicized text between the braces. For example typing:

```
I like to \underline{underline} text here and  \underline{everywhere} . I like it  \textit{so} much,
I can't \textit{\underline{stop!}}
```

into your LaTeX source file will yield:

I like to <u>underline</u> text here and <u>everywhere</u> . I like it *so* much, I can't *<u>stop!</u>*

1.2.2 Left/Right/Center Justification

Justification in LaTeX is easy to do. If you want to left-align text, you must put the text you want left-justified between the commands `\begin{flushleft}`and `\end{flushleft}`. For example, the code

```
\begin{flushleft}
Left-aligned text is fun.
I like left-aligned text.
I think you should like left-aligned text too!
\end{flushleft}
```

will yield:

Left-aligned text is fun.
I like left-aligned text.
I think you should like left-aligned text too!

If you haven't guessed yet, right-aligned text is done with the `\begin{flushright}`and `\end{flushright}`commands. So typing

```
\begin{flushright}
Whoa, this text is right-aligned.
What the heck?
Who uses right-aligned text anyway?
\end{flushright}
```

will yield

Whoa, this text is right-aligned.
What the heck?
Who uses right-aligned text anyway?

Centering text is also possible in LaTeX, except instead of the `\begin{flushcenter}`command (which doesn't exist), we use the much simpler `\begin{center}`and `\end{center}`commands. Once again, typing

```
\begin{center}
This text is centered.
Centered text makes me feel at peace.
These examples make no sense, don't they?
\end{center}
```

will give us

This text is centered.
Centered text makes me feel at peace.
These examples make no sense, don't they?

1.2.3 Quotes and Verses

When using quotation marks, it's probably a good idea *not* to use the `"`key for opening and closing quotes. Rather, we use two ``` `` ```(tilde key) characters for opening quotes, and two single quotes (`''`) for closing quotes.

Bad:
I think it was Kant who said ”Let my people go.” Or perhaps that was Moses.

Good:
I think it was Kant who said “Let my people go.” Or perhaps that was Moses.

If your quotation is long (say, several paragraphs), it might be a good idea to use the `\begin{quote}`and `\end{quote}`commands, as they will automatically be indented in text. An example:

```
One of my favorite quotes from \textit{Principles of Mathematical Analysis} goes something like this:
\begin{quote}
If f is a continuous complex function on [a,b], there exists a sequence of polynomials $P_n$
such that $\lim_{n\rightarrow\infty} P_n(x)=f(x)$. uniformly on [a,b]. If f is real, the $P_n$ may be taken real.
\end{quote}

Math is very fun!
```

will yield the following result:

One of my favorite quotes from *Principles of Mathematical Analysis* goes something like this:

> If f is a continuous complex function on [a,b], there exists a sequence of polynomials P_n such that $\lim_{n\rightarrow\infty} P_n(x) = f(x)$. uniformly on [a,b]. If f is real, the P_n may be taken real.

Math is very fun!

1.2.4 Line and Page Breaks

LaTeX automatically breaks up lines and spaces between words and will auto-hyphen words when necessary. If you've noticed from this reader, every line is optimized so that the lengths are the same. Normally, the fist line of every paragraph is automatically indented, with no additional space between paragraphs. When it becomes necessary to add a line break, two commands are available: `\\`and `\newline`. Both will break an additional line. LaTeXwill automatically spill over to another page when there is too much content for one page; hence, premature page breaks are done with the command `\newpage`. If you need to break more than one line, use `\linebreak[n]`, where n is the number of lines you want to break. `\pagebreak[n]`does the same thing, except it breaks n pages.

Chapter 2

Math Mode Basics

One common usage of LaTeX is the typesetting of mathematical discourse and documents with mathematical content. LaTeX comes with a good deal of functionality in this area and has become a fairly standard tool in the math community. In addition, the math typesetting abilities of LaTeX can be further expanded by the use of the $\mathcal{AMS}$- LaTeX packages.

Packages in general will be discussed later, but for now, it suffices to add

```
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amssymb}
```

to the preamble of any document you are typesetting with math content. This places additional environments and symbols at your disposal that make life far easier.

In mathmode, all the letters on the keyboard can be used as constants and variables. They are automatically stylized to make them typographically different from their textual counterparts. For example an `a` in mathmode will be displayed as a while in normal text, it is an a. In addition to the letters available on the keyboard, there is a wide variety of other characters and symbols available through various LaTeX commands. LaTeX also offers an extensive array of hats, bars, and other useful accents.

The math mode chapters do not have many examples. However, there is a selection of examples following the chapters.

2.1 Environments

LaTeX is extremely powerful, not least in its ability to typeset math. There are three math environments, `math`, `displaymath` and `equation`. The inline environment allows the inclusion of math in a normal line of text, while the displayed environment enables one to set off lines of math. The equation environment is essentially the same as the displayed environment, but each instance is numbered sequentially. $x + y = z$ is an example of the inline environment, while

$$x + y = z$$

is the same thing in the displayed environment.

2.1.1 Inline Math

Like other LaTeX environments, inline math, or more accurately, the `math` environment is delimited by `\begin` and `\end` commands. In TeX, inline math was set off by dollar signs. This convention was carried over to LaTeX, which allows us a shortcut, as all those `\begin` and `\end`s would get very tedious very quickly. Using the `$` convention, the above example would appear in one's source as

```
$x+y=z$ is an example of the inline environment.
```

2.1.2 Displayed Math

The displayed math environment is delimited by in the strictest sense by `\begin{displaymath}` and `\end{displaymath}`. However one can also use \[at the beginning and \] at the end of the statement. Thus, using square brackets as the delimiters,

$$x + y = z$$

would appear in one's source as

```
\[x+y=z\]
```

2.1.3 Equations

The equation environment is delimited, not surprisingly, by `\begin{equation}` and `\end{equation}`. Our example equation in the equation environment would appear as

$$x + y = z \tag{2.1}$$

2.1.4 Stars (Well, asterisks really)

Various environments and structures in LaTeX will desire the number themselves, like the `equation` environment above. However, unlike `equation`, not everything has an unnumbered counterpart. To avoid the numbering, an asterisk, *, is used to disable numbering. This fact is of little concern now, but it is a fact to file away, as it will come up again later in the chapter.

2.2 Commands

If you take a look at the keyboard, you'll see a number of symbols that look fairly useful for typing math, including +, -, *, /, =, (and), essentially the same things you find on a small calculator. Which is all well and good, up to a point, but what about typing homework? Or textbooks for that matter, as your math textbooks were most likely done with LaTeX. It is possible to include symbols in math environments that are not available on one's keyboard. This is achieved through various commands. With two important exceptions, math commands, like many other LaTeX commands, begin with a backslash, \.

2.2.1 Subscripts and Superscripts

The two exceptions to the commands beginning with a backslash rule are ˆ and _, the commands for superscript and subscript, respectively. Thus, x^2 is entered as `x^2` and x_2 as `x_2`. Note that ˆ and _ only include the character immediately following, unless one encloses the subscript or superscript in brackets. Therefore x^{12} is entered as `x^{12}` not as `x^12`, which is x^12.

2.2.2 Math Symbols

If you think back to the last homework assignment you did or the last textbook you read, you'll probably remember that math uses all sorts of symbols such as ∞, $\sum$, $\int$ and a whole host of non-Roman letters. Below is a table of various math symbols. The commands work in essentially the same way as other LaTeX commands, just remember that they only work in math mode.
TABLE OF MATH SYMBOLS Frequently, it seems like a lot of bother to go sifting through appendices looking for a command. Fortunately, many of the commands have fairly intuitive names so, if you're willing to risk errors, you can be lazy and guess. At this point, you might be thinking "You said think about my last homework. I didn't use half that stuff. I had Greek letters, hats and transposes coming out of my ears!" We will get there in a few short paragraphs, so don't worry.

2.2.3 Delimiters

In a very basic sense, delimiters are things like parentheses and brackets. Parentheses and square brackets can be typeset with the matching key on the keyboard. Other delimiters have specific commands. Beyond the commands for different delimiters, it is important to know how to control the size of delimiters. Let's say you wanted to put parentheses around a summation for some reason.

$$(\sum_{k=0}^{10} k)$$

looks pretty silly because the parentheses are so small. We can use `\left` and `\right` to correct this problem. When immediately followed by the appropriate delimiter (i.e. one uses (with `\left`), LaTeX will size the delimiter to what it thinks is the right size. By writing `\left(\sum_{k=0}^{10}k\right)`, we get

$$\left(\sum_{k=0}^{10} k\right)$$

which looks a lot better. It is important to note that the left and right hand delimiters must be paired. For example, attempting to compile `$\left($` will result in an error. It is possible to have a single left or right delimiter by pairing it with a `\left.` or `\right.` as appropriate. If you only want a delimiter on one side, you can use `\left` or `\right` without a delimiter to complete the pair. You also have to option of choosing the size of a delimiter with `\big`, `\Big`, `\bigg` and `\Bigg`, which are used like `\left` and `\right` and give progressively larger delimiters. The existence of these commands may seem a little puzzling, given that LaTeX will size delimiters on its own, if you ask it to. However, there are times when you will want to do it yourself.

Delimiters (those with commands)

$\{$	`\{`
$\}$	`\}`
$\backslash$	`\backslash`
$\langle$	`\langle`
$\rangle$	`\rangle`
$\Vert$	`\Vert` or `|`
$\lfloor$	`\lfloor`
$\rfloor$	`\rfloor`
$\lceil$	`\lceil`
$\rceil$	`\rceil`

2.2.4 Fractions, Roots and Combinations(?)

Yet another group of commands are ones somewhat like ˆ and _ in that they take arguments. Included in this group are the rather useful `\frac` and `\sqrt`, fractions and roots, respectively. One writes a root by `\sqrt`{*expression*}, where *expression* is replaced by what goes under the root. If one wants the nth root, the full command is actually `\sqrt`[n]{*expression*}, so $\sqrt[3]{x}$ is written `\sqrt[3]`{*expression*}. Likewise, fractions take the form of
`\frac`{*numerator*}{*denominator*}. Combinations, such as $\binom{a}{b}$ are written as {*combinations*}`\choose`{*objects*}.

2.2.5 Operators

Occasionally you will want to type things like sin and det, things that are both words and symbols. LaTeX provides commands for such operators both so you don't have to exit math mode and to allow for appealing spacing.

Operators

sin	`\sin`	exp	`\exp`
cos	`\cos`	gcd	`\gcd`
tan	`\tan`	hom	`\hom`
csc	`\csc`	inf	`\inf`
sec	`\sec`	ker	`\ker`
cot	`\cot`	lg	`\lg`
cosh	`\cosh`	lim	`\lim`
sinh	`\sinh`	lim inf	`\liminf`
tanh	`\tanh`	lim sup	`\limsup`
coth	`\coth`	ln	`\ln`
arcsin	`\arcsin`	log	`\log`
arccos	`\arccos`	max	`\max`
arctan	`\arctan`	min	`\min`
arg	`\arg`	Pr	`\Pr`
deg	`\deg`	sup	`\sup`
det	`\det`	dim	`\dim`

2.2.6 Accents

Another group of math commands are what one might describe as modifiers for letters. This would include bars and hats. It might not come as a surprise that the command for $\hat{x}$ is `\hat{x}`. There is also `\widehat`, which looks a bit better on capital letters. Compare $\hat{T}$ to $\widehat{T}$. This happens because `\widehat` extends over everything enclosed in the brackets, whereas `\hat` simply centers a hat over what's enclosed in the brackets. As a result, one can have $\widehat{xyz}$ (`\widehat{xyz}`) rather than $x\hat{y}z$ (`\hat{xyz}`). `\tilde` works the same way as `\hat`. There is also `\widetilde`. $\bar{x}$ is obtained by `\bar{x}`. It is worth noting that `\widebar` does not exist. Instead, one can use `\overline` to accomplish the same effect.

2.3 Greek and Other Fancy Letters

2.3.1 Greek Letters

For the Greek alphabet, the commands are simply a backslash followed by the name of the letter. For example, β is `\beta`. As always, commands are case sensitive, so `\Gamma` is Γ rather than γ. Additionally, some letters don't have capitalised versions in LaTeX. The only capital Greek letter commands are `\Gamma`, `\Delta`, `\Theta`, `\Lambda`, `\Xi`, `\Pi`, `Sigma`, `\Upsilon`, `\Phi`, `\Psi` and `\Omega`.

Greek Letters (CURRENTLY A LOUSY PAGE BREAK, FIXABLE IN FINAL DRAFT)

α	`\alpha`	ν	`\nu`
β	`\beta`	ξ	`\xi`
γ	`\gamma`	π	`\pi`
δ	`\delta`	ρ	`\rho`
ϵ	`\epsilon`	σ	`\sigma`
ζ	`\zeta`	τ	`\tau`
η	`\eta`	υ	`\upsilon`
θ	`\theta`	ϕ	`\phi`
ι	`\iota`	χ	`\chi`
κ	`\kappa`	ψ	`\psi`
λ	`\lambda`	ω	`\omega`
μ	`\mu`		

Γ	`\Gamma`	Σ	`\Sigma`
Δ	`\Delta`	Υ	`\Upsilon`
Θ	`\Theta`	Φ	`\Phi`
Λ	`\Lambda`	Ψ	`\Psi`
Ξ	`\Xi`	Ω	`\Omega`
Π	`\Pi`		

2.3.2 Fancy Fonts

The $\mathcal{AMS}$ packages provide several fonts that allow for some rather useful letter that are not avaialble through previously discussed commands. These fonts are selected by `\[font command]{text in this font}`. This is most easily illustrated by an example such as `$\mathbb{R}$`. This command typesets a capital `R` in the math board bold font. Compiled, it gives $\mathbb{R}$, the familiar `R` for the reals.

Other fonts available from LaTeX and the $\mathcal{AMS}$ packages are

```
\mathbb{}
\textrm{}
\mathrm{}
\mathbf{}
\mathsf{}
\mathit{}
\mathcal{}
\mathfrak{}
```

The board bold font is the most commonly used one. Also of note are `\textrm{}` and `mathrm`. Both of these fonts override the default italicization in math mode. `\textrm{}` typesets text in a style identical to normal text mode and `\mathrm` typesets text in normal upright letters, but slighly different for math. A more detailed discussion of these differences can be found in Oetiker `a reference` Some experimentation with a LaTeX compilier or consultation of documentation for the $\mathcal{AMS}$ packages will give you a good idea of what the other fonts look like.

2.3.3 Dots

An ellipsis (three dots) is often handy. LaTeXprovides a wide variety of ellipsises. They are as follows.

`\ldots`	$\ldots$
`\cdots`	$\cdots$
`\vdots`	$\vdots$
`\ddots`	$\ddots$

Chapter 3

More Math Mode

3.1 Matrices

LaTeX provides several different environments for making matrix and table-like structures. Common to all of these environments is the use of the ampersand, &, to delineate columns and the line break \\ to separate rows. Each row much be on its own line, though columns need not line up. Although it is not necessary to maintain any manner of alignment, I find it helpful, especially in larger tables, to maintain some alignment of the columns between rows.

Let us start by making a simply matrix using the array environment. Note that to use the array environment, one must be in math mode first. Suppose we want to type the following matrix:

$$\left(\begin{array}{rr} 1 & 0 \\ 0 & 1 \end{array}\right)$$

Then we want an array with a 1 and a 0 in the first row, and a 0 and a 1 in the second row. Thus, the code is

```
\begin{array}{rr}
1 & 0 \\
0 & 1
\end{array}
```

However, this will not enclose the numbers in parentheses. As previously noted, the large left and right parentheses can be typeset using the commands `\left` and `\right`. Also, not that `{rr}` has been appended to `\begin{array}`. This indicates the vertical alignment of the columns. The first `r` indicates the first column should be right-aligned, while the second `r` indicates the same for the second column. Text in the columns can be aligned left, right or center using `l, r,` and `c`, respectively. Thus

```
\left(\begin{array}{rr}
1 & 0 \\
0 & 1 \\
\end{array} \right)
```

gives the completed matrix.

The $\mathcal{AMS}$ packages provide various environments for matrices beyond `array`. `matrix` is essentially the same as `array`, as it creates a matrix without delimiters. However, `matrix` automatically centers the entries within the columns. `pmatrix` and `vmatrix` are two other matrix environments, and they come with delimiters. Consider our previous example matrix written using `pmatrix` and `vmatrix` respectively

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

Additionally, `matrix` and like environments can only handle matrices of up to 10 columns. Should you need a matrix with more columns, precede the start of the matrix environment with `\setcounter{MaxMatrixCols}`{*columns*}.

3.2 Commutative Diagrams

While extensive graphics work in LaTeX becomes something of a computer geek black art, commutative diagrams can be drawn easily using the `\xymatrix{}` command. This command is part of the xy package. The `\xymatrix{}` command is used inside math mode, and behaves much like the array environment. For example,

```
\[
\xymatrix{G & H \\
J & K}
\]
```

will give

$$\begin{tikzcd} G & H \\ J & K \end{tikzcd}$$

These can all be connected with arrows using the `\ar[]` command. The argument of the command indicates the direction of the arrow, `l`,`r`,`u`,`d` for left, right, up, and down respectively. These directions can be combined, repeatively if necessary. For example, `\ar[drrr]` points down three columns to the right. The source code for a basic commutative diagram woud look like

```
\[
\xymatrix{G \ar[r] \ar[d] & H \ar[d] \\
J \ar[r] & K}
\]
```

This gives the diagram

$$\begin{tikzcd} G \arrow[r] \arrow[d] & H \arrow[d] \\ J \arrow[r] & K \end{tikzcd}$$

Furthermore, these arrows can be labeled using subscripts and sueprscripts. For vertical arrows, subscripts go the the left, and sueprscripts go to the right.

```
\[
\xymatrix{G \ar[r]_f \ar[d]^\tau & H \ar[d]_\sigma \\
J \ar[r]_g & K}
\]
```

$$\begin{tikzcd} G \arrow[r, "f"'] \arrow[d, "\tau"'] & H \arrow[d, "\sigma"'] \\ J \arrow[r, "g"'] & K \end{tikzcd}$$

3.3 Piecewise Functions

The typesetting of piecewise functions is easily accomplished by combining our knowledge of delimiters and arrays. Personally, I recommend that piecewise functions be typeset almost exclusively in displayed math environments. It is possible to place a piecewise function in an inline environment, but the large size of a piecewise function will make a mess of your page.

For this discussion, let us understand that the left hand side of the equation is only one line, and the various pieces of the function are on the right hand side. To begin, first, you enter the left hand side of the equation as you would normally do with any other function. Thus, input something similar to

```
\[
y=
\]
```

Next, we need a large curly brace. As noted in the prior material on delimiters, we can get an automatically sized brace by using the command `\left\{`. Now we are ready to add in the right hand side. This is easily accomplished using an array. Our example source becomes

```
\[
y= \left\{ \begin{array}{ll}
\frac{x^2}{x} & x\neq 0 \\
0 & x=0
\end{array} \right.
\]
```

Which gives the output

$$y= \left\{ \begin{array}{ll} \frac{x^2}{x} & x\neq 0 \\ 0 & x=0 \end{array} \right.$$

The one possibly unexpected part of this is the command `\right.`. This command is necessary to pair with the command `\left\{`. The dot tells LaTeX to print a blank delimiter to pair with the left brace that we used in our piecewise function. If you add a piecewise function to a document, and LaTeX begins a lot of cryptic complaining about things being unpaired check for a missing `\right.`.

If you would like to add some space between the right hand side of the equation and the range of the variable, there are a variety of spacing commands that can be used that will be discussed later. A simple and rough way to accomplish this is with the command `\quad`. This gives us the followsing.

```
\[
y= \left\{ \begin{array}{ll}
\frac{x^2}{x} & \quad x\neq 0 \\
0 & \quad x=0
\end{array} \right.
\]
```

$$y = \left\{ \begin{array}{ll} \frac{x^2}{x} & \quad x \neq 0 \\ 0 & \quad x = 0 \end{array} \right.$$

3.4 Better Looking Math

There are several commands that are very useful for improving the appearance of your mathematical content.

3.4.1 `eqnarray`

The `eqnarray` envirnoment is similar to the `array` or `tabular` envirnoment, but is at most three columns wide, and no alignment needs to be declared for the columns. Also, `eqnarray` is a math environment, meaning no dollar signs or other mathmode envirnoments are necessary. The envirnoment comes in two flavors; `eqnarray` will number the rows while `eqnarray*` will not. The environment is entered and exited like any other with `\begin{eqnarray*}` and `\end{eqnarray*}` .

3.4.2 `displaystyle`

The `\displaystyle` command is the best friend of inline math. For the most part, `\displaystyle` will make things bigger. It will make inline fractions more readable, and for inline integrals, and general sums and products, it will place the indicies above and below, rather than to the side of the operator. The use of `\displaystyle` is best described with a few examples. In the following, the compiled code will immediately follow the source.

```
Some inline math $\frac{\sqrt{2}}{\sqrt{3}}$.
```

Some inline math $\frac{\sqrt{2}}{\sqrt{3}}$.

```
Now with \texttt{displaystyle}
$\displaystyle\frac{\sqrt{2}}{\sqrt{3}}$.
```

Now with `displaystyle` $\displaystyle\frac{\sqrt{2}}{\sqrt{3}}$.

```
A bad looking sum $\sum_{n=1}^\infty \frac{1}{n}$.
```

A bad looking sum $\sum_{n=1}^\infty \frac{1}{n}$.

```
\texttt{displaystyle} to the rescue!
$\displaystyle \sum_{n=1}^\infty \frac{1}{n}$.
```

`displaystyle` to the rescue! $\displaystyle \sum_{n=1}^\infty \frac{1}{n}$.

Note that `\displaystyle` need only be used once in each math environment.

3.5 Theorems and Proofs

Back in the dark ages of TEX[1], the command to begin a theorem was called proclaim. However, LATEX isn't as overjoyed about its theorems, so proclaim is gone and replaced by the theorem environment. If you open a math book, you'll notice that the theorems and theorem-like objects are all labeled differently. Some are called theorems, others lemmas and still others propositions, among other things. In the preamble of the document, you need to set up each of these different names separately using the `\newtheorem` command.

First, let's look at a theorem that just has a number, rather than a specific name.

Theorem 1. *Let V be a finite-dimensional vector space, and and define $\psi : V \rightarrow V^{**}$ by $\psi(x) = \hat{x}$. Then ψ is an isomorphism.*

The first thing we had to do to typeset this fine theorem was define an environment for it via `newtheorem`. The command has two arguments, taking the form `\newtheorem`{*name*}{*Name*}. The first argument names the environment, and the second argument determines what things will be labeled. (The two arguments do not have to be the same, but it would be bit weird for the `chicken` environment to be producing theorems.) In our example, the full command looked like `\newtheorem{theorem}{Theorem}`. The remainder of the code was

```
\begin{theorem}
Let $V$ be a finite-dimensional vector space, and and define $\psi:V\rightarrow V^{**}$
by $\psi(x)=\hat{x}$.  Then $\psi$ is an isomorphism.
\end{theorem}
```

But, what happens if your theorem has a name?

Theorem 2 (Cayley-Hamilton)**.** *Let T be a linear operator on a finite-dimensional vector space V, and left $f(t)$ be the characteristic polynomial of T. Then $f(T) = T_0$, the zero transformation. That is, T "satisfies" its characteristic equation.*

The only difference between this and the previous example is the addition of the name. We didn't have to add another `\newtheorem` because we had already defined a theorem environment, which is also why this is numbered as Theorem 2. So how did we add the name? When we opened the `theorem` environment, we did it with `\begin{theorem}[Cayley-Hamilton]`. This is all well and good, but what if you have a theorem that is so fantastically cool giving it a number would be cheapening the moment?

Morse Lemma. *Let p_0 be a non-degenerate critical point of a function f of two variables. Then we can choose appropriate local coordinates (X, Y) in such a way that the function f expressed with respect to (X, Y) takes one of the following three standard forms:*

(i). $f = X^2 + Y^2 + c$

(ii). $f = X^2 - Y^2 + c$

(iii). $f = -X^2 - Y^2 + c$

[1]One should not advertise that one actually used TEX, especially, if LATEX was written before one was born, as one will get strange looks.

If you recall, the way we got rid of numbering on other things was to add a *. The question is, do you add it to `\newtheorem` or the start of the environment? From experience, it seems as if adding it to `\newtheorem` would ensure that every instance of that environment will be unnumbered, and adding it to the start of the environment would just eliminate the number from that one instance. As it happens, the answer is `\newtheorem`, sort of. If you just tack on the asterisk, you'll get an error as `\newtheorem*` is not defined in LaTeX. However, it does exist with the inclusion of the `amsthm` package. To typeset the Morse Lemma we declared the environment with `\newtheorem*{morse}{Morse Lemma}`.

Each different proclamation environment, by default, has its own counter, so you could have a Theorem 1, a Definition 1, a Lemma 1 and who knows what else numbered 1. But what if you wanted your theorems and lemma numbered in succession so Theorem 1 was followed by Lemma 2 rather than Lemma 1? Covering the entire scope of counters would be unmanageable at this juncture, so we'll just cover the very basics and leave the rest for another time and place (quite possibly another book). As it turns out, achieving this end leads us back to the `\newtheorem` command. So how do you declare the lemma environment?

```
\newtheorem{lemma}[theorem]{Lemma}
```

There's one potential pitfall with this system. What happens if `theorem` hasn't been defined yet? Bad, bad things happen, so make sure that you tie the counting to something that already exists.

There another counting option ties numbering to some subdivision of the document, creating Theorem 1.1, etc. This option is of the form

`\newtheorem{theorem}{Theorem}`[*section*]

where *section* can be any subsection of the document such as `section`, `chapter` or `subsection`. Using this numbering scheme and tying the numbering to the subsection, we can create:

Theorem 3.5.0.1. *Every bounded, monotonic sequence is convergent.*

The Proof Environment

In addition to allowing for unnumbered theorems, the `amsthm` package provides the `proof` environment. Using the environment is fairly straightforward, just follow the `\begin{proof}` with what you want the proof labeled in square brackets. For example, if you want to include the proof of the Morse Lemma, your code would look something like:

`\begin{proof}[Proof of Morse Lemma]`
the proof goes here
`\end{proof}`

The `proof` environment automatically places the QED symbol, □, at the end, leaving an appropriate amount of space in front. However, should it place the symbol in the wrong place, you can force the placement with `\qedhere`.

Chapter 4

Math Errors

4.1 Errors

Errors are quite probably everyone's least favorite part of using LaTeX. LaTeX is considerate enough to give error messages rather than simply not compiling, but the error messages are still fairly opaque. Sometimes the easiest way to fix an error is to rewrite the offending line.

4.2 Common Errors

4.3 Hard to Fix Errors

Chapter 5

Math Examples

The following is a collection of LaTeX examples. First, the source code is displayed, this is immediately followed by the compiled code. Try not to pay attention to the mathematical content as it is culled from old homeworksets and may or may not be corect.

5.1 Radicals and Fractions

This is an example of usages of radicals and fractions. No packages are necessary.

```
\paragraph*{4.4}
Show that $1,\sqrt{2},\sqrt{3},\sqrt{6}$ are
linearly independent over $\mathbb{Q}$.
\subparagraph*{}
Suppose $1,\sqrt{2},\sqrt{3},\sqrt{6}$ are linearly
dependent over $\mathbb{Q}$. Then there exist
$p,q,r,s\in\mathbb{Q}$ such that
$p+q\sqrt{2}+r\sqrt{3}+s\sqrt{6}=0$ and
$p,q,r,s$ are not all zero. Then either $r$ or $s$
is nonzero, otherwise, $p=0$, or
$q\sqrt{2}\in\mathbb{Q}$, neither of which can be
the case if $p,q,r,s$ are not all zero. Then we have
\begin{displaymath}
\sqrt{3}=\frac{-p-q\sqrt{2}}{r+s\sqrt{2}}
\end{displaymath}
and this is well defined because at least one of
$r$ and $s$ is not zero. Thus, there exist rational
numbers $e,f$ such that $\sqrt{3}=e+f\sqrt{2}$.
Squaring this, we have
\begin{eqnarray*}
3 & = & e^2+ef\sqrt{2}+2f^2 \\
\frac{3-e^2-2f^2}{ef}=\sqrt{2}.
\end{eqnarray*}
```

```
Which implies that $\sqrt{2}$ is rational.
Thus, we have a contradiction. Therefore, we can
conclude that $1,\sqrt{2},\sqrt{3},\sqrt{6}$ are
linearly independent over $\mathbb{Q}$.
```

4.4 Show that $1, \sqrt{2}, \sqrt{3}, \sqrt{6}$ are linearly independent over $\mathbb{Q}$.

Suppose $1, \sqrt{2}, \sqrt{3}, \sqrt{6}$ are linearly dependent over $\mathbb{Q}$. Then there exist $p, q, r, s \in \mathbb{Q}$ such that $p + q\sqrt{2} + r\sqrt{3} + s\sqrt{6} = 0$ and p, q, r, s are not all zero. Then either r or s is nonzero, otherwise, $p = 0$, or $q\sqrt{2} \in \mathbb{Q}$, neither of which can be the case if p, q, r, s are not all zero. Then we have

$$\sqrt{3} = \frac{-p - q\sqrt{2}}{r + s\sqrt{2}}$$

and this is well defined because at least one of r and s is not zero. Thus, there exist rational numbers e, f such that $\sqrt{3} = e + f\sqrt{2}$. Squaring this, we have

$$\begin{aligned} 3 &= e^2 + ef\sqrt{2} + 2f^2 \\ \frac{3 - e^2 - 2f^2}{ef} = \sqrt{2}. & \end{aligned}$$

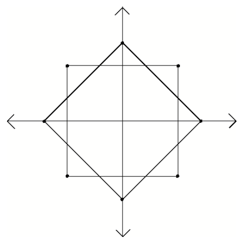
Which implies that $\sqrt{2}$ is rational. Thus, we have a contradiction. Therefore, we can conclude that $1, \sqrt{2}, \sqrt{3}, \sqrt{6}$ are linearly independent over $\mathbb{Q}$.

Note that the inline franctions can be made larger by using `\displaystyle`.

5.2 Graphics

This is an example of an eps grphic and a table. The package `graphicx` was used to include the image.

```
\includegraphics[width=0.25\textwidth]{9-13-15e.png}
Using the notation from Chapter 13, the action of
the Galois group as as follows:
```



Using the notation from Chapter 13, the action of the Galois group as as follows:

5.3 `eqnarray*`

The following is an example of the `\eqnarray*` environment. No packages are necessary

```
Conjugating the elements of $\mathbb{V}$ by the
transposition of $\mathbb{S}_4$, we have
\begin{eqnarray*}
(12)(12)(34)(12) & = & (12)(34) \\
(12)(13)(24)(12) & = & (14)(23) \\
(12)(14)(23)(12) & = & (13)(24) \\
(13)(12)(34)(13) & = & (14)(23) \\
(13)(13)(24)(13) & = & (13)(24) \\
(13)(14)(23)(13) & = & (12)(34) \\
(14)(12)(34)(14) & = & (13)(24) \\
(14)(13)(24)(14) & = & (12)(34) \\
(14)(14)(23)(14) & = & (14)(23) \\
(23)(12)(34)(23) & = & (13)(24) \\
(23)(13)(24)(23) & = & (12)(34) \\
(23)(14)(23)(23) & = & (14)(23) \\
(24)(12)(34)(24) & = & (14)(23) \\
(24)(13)(24)(24) & = & (13)(24) \\
(24)(14)(23)(24) & = & (12)(34) \\
(34)(12)(34)(34) & = & (12)(34) \\
(34)(13)(24)(34) & = & (14)(23) \\
(34)(14)(23)(34) & = & (13)(24)
\end{eqnarray*}
thus, $\mathbb{V}$ is closed under conjugation by
transpositions in $\mathbb{S}_4$ as $\mathbb{S}_4$
is generated by alltranspositions on 4 letters,
we can conclude that $\mathbb{V} \lhd \mathbb{S}_4$.
```

Conjugating the elements of $\mathbb{V}$ by the transposition of $\mathbb{S}_4$, we have

$$\begin{array}{rcl}
(12)(12)(34)(12) & = & (12)(34) \\
(12)(13)(24)(12) & = & (14)(23) \\
(12)(14)(23)(12) & = & (13)(24) \\
(13)(12)(34)(13) & = & (14)(23) \\
(13)(13)(24)(13) & = & (13)(24) \\
(13)(14)(23)(13) & = & (12)(34) \\
(14)(12)(34)(14) & = & (13)(24) \\
(14)(13)(24)(14) & = & (12)(34) \\
(14)(14)(23)(14) & = & (14)(23) \\
(23)(12)(34)(23) & = & (13)(24) \\
(23)(13)(24)(23) & = & (12)(34) \\
(23)(14)(23)(23) & = & (14)(23) \\
(24)(12)(34)(24) & = & (14)(23) \\
(24)(13)(24)(24) & = & (13)(24) \\
(24)(14)(23)(24) & = & (12)(34) \\
(34)(12)(34)(34) & = & (12)(34) \\
(34)(13)(24)(34) & = & (14)(23) \\
(34)(14)(23)(34) & = & (13)(24)
\end{array}$$

thus, $\mathbb{V}$ is closed under conjugation by transpositions in $\mathbb{S}_4$ as $\mathbb{S}_4$ is generated by all transpositions on 4 letters, we can conclude that $\mathbb{V} \lhd \mathbb{S}_4$.

5.4 Picewise Functions

The following is an example of a piecewise function.

```
\paragraph*{3}
Let $m$ be a cardinal number such that $m+\aleph_0=C$.
Without the Axiom of Choice, show that $m=C$. Since
$m+\aleph_0=C$, we have $A\cup \omega \sim \lbrace
0,1 \rbrace^\omega$ where $A$ and $\omega$ are disjoint.
Let $f:\lbrace0,1\rbrace^\omega \to A \cup \omega$ be
an equivalence. Consider the families of functions
$F_n$ and $G_n$ in $\lbrace0,1\rbrace^\omega$, for all
$n\in \omega$ defined as follows.

\begin{displaymath}
F_n(x)=\left\{\begin{array}{ll}
0 & x\neq n \\
1 & x=n
\end{array}\right.
G_n(x)=\left\{\begin{array}{ll}
```

```
0 & x=n \\
1 & x\neq n
\end{array}\right.
\end{displaymath}

Define a one-to-one and onto function
$g:\lbrace0,1\rbrace^\omega \to \lbrace0,1\rbrace^\omega$
such that $f\circ g(F_n)\in \omega$ for all $n\in \omega$
and $f\circ g(G_n)\in A$ for all $n\in \omega$.
Since $g$ is one-to-one and onto, $f\circ g$ is an
equivalence from $\lbrace0,1\rbrace^\omega$ to $A \cup \omega$.
Furthermore, since the set of all
$F_n\in \lbrace0,1\rbrace^\omega$ and the set of all
$G_n\in \lbrace0,1\rbrace^\omega$ are both clearly
equivalanet to $\omega$, we can conclude that there is a
subset $B$ of $A$ that is equivalent to $\omega$.
Thus, $A=(A-B) \cup B$. Hence, there is a cardinal number $n$
such that $m=n+\aleph_0$. Therefore,
$m+\aleph_0=n+\aleph_0+\aleph_0=n+\aleph_0=m$. Thus, $m=C$.
```

3 Let m be a cardinal number such that $m + \aleph_0 = C$. Without the Axiom of Choice, show that $m = C$. Since $m + \aleph_0 = C$, we have $A \cup \omega \sim \{0,1\}^\omega$ where A and ω are disjoint. Let $f : \{0,1\}^\omega \to A \cup \omega$ be an equivalence. Consider the families of functions F_n and G_n in $\{0,1\}^\omega$, for all $n \in \omega$ defined as follows.

$$F_n(x) = \left\{ \begin{array}{ll} 0 & x \neq n \\ 1 & x = n \end{array}\right. \quad G_n(x) = \left\{ \begin{array}{ll} 0 & x = n \\ 1 & x \neq n \end{array}\right.$$

Define a one-to-one and onto function $g : \{0,1\}^\omega \to \{0,1\}^\omega$ such that $f \circ g(F_n) \in \omega$ for all $n \in \omega$ and $f \circ g(G_n) \in A$ for all $n \in \omega$. Since g is one-to-one and onto, $f \circ g$ is an equivalence from $\{0,1\}^\omega$ to $A \cup \omega$. Furthermore, since the set of all $F_n \in \{0,1\}^\omega$ and the set of all $G_n \in \{0,1\}^\omega$ are both clearly equivalanet to ω, we can conclude that there is a subset B of A that is equivalent to ω. Thus, $A = (A - B) \cup B$. Hence, there is a cardinal number n such that $m = n + \aleph_0$. Therefore, $m + \aleph_0 = n + \aleph_0 + \aleph_0 = n + \aleph_0 = m$. Thus, $m = C$.

5.5 XY

The following is an example of drawing commutative diagrams with the xy package. For the following code to function, the xy package must be declared in the preamble of the document.

```
Suppose $\phi:A\to B$ and $\psi:C\to D$ are isomorphic
field extensions. Then the following diagram commutes.
\begin{displaymath}
\xymatrix{
A \ar[r]^\phi \ar[d]_\alpha & B \ar[d]_\beta \\
C \ar[r]^\psi & D}.
\end{displaymath}
Thus, $\beta\circ\phi=\psi\circ\alpha$. Therefore the diagram
\begin{displaymath}
```

```
\xymatrix{
C \ar[r]^\psi \ar[d]_\alpha^{-1} & D \ar[d]_\beta^{-1} \\
A \ar[r]^\phi & B}.
\end{displaymath}
also commutes. Thus, $\psi$ is isomorphic to $\phi$.
Therefore, $\phi~\psi$ implies $\psi~\phi$.
\subparagraph*{}
Suppose $\phi:A\to B$ and $\psi:C\to D$ are isomorphic field
extensions. Furthermore, let $\psi:C\to D$ and $\mu:E\to F$
be isomorphic field extensions. The the following diagrams commute.
\begin{displaymath}
\xymatrix{
A \ar[r]^\phi \ar[d]_\alpha & B \ar[d]_\beta \\
C \ar[r]^\psi & D}
\xymatrix{
C \ar[r]^\psi \ar[d]_\delta & D \ar[d]_\varepsilon \\
E \ar[r]^\mu & F}
\end{displaymath}
```

Suppose $\phi : A \to B$ and $\psi : C \to D$ are isomorphic field extensions. Then the following diagram commutes.

$$\begin{tikzcd} A \arrow[r, "\phi"] \arrow[d, "\alpha"'] & B \arrow[d, "\beta"'] \\ C \arrow[r, "\psi"] & D \end{tikzcd}.$$

Thus, $\beta \circ \phi = \psi \circ \alpha$. Therefore the diagram

$$\begin{tikzcd} C \arrow[r, "\psi"] \arrow[d, "\alpha"', "-1"] & D \arrow[d, "\beta"', "-1"] \\ A \arrow[r, "\phi"] & B \end{tikzcd}.$$

also commutes. Thus, ψ is isomorphic to ϕ. Therefore, $\phi\ \psi$ implies $\psi\ \phi$.

Suppose $\phi : A \to B$ and $\psi : C \to D$ are isomorphic field extensions. Furthermore, let $\psi : C \to D$ and $\mu : E \to F$ be isomorphic field extensions. The the following diagrams commute.

$$\begin{tikzcd} A \arrow[r, "\phi"] \arrow[d, "\alpha"'] & B \arrow[d, "\beta"'] \\ C \arrow[r, "\psi"] & D \end{tikzcd} \quad \begin{tikzcd} C \arrow[r, "\psi"] \arrow[d, "\delta"'] & D \arrow[d, "\varepsilon"'] \\ E \arrow[r, "\mu"] & F \end{tikzcd}$$

Chapter 6

Tables

Matrices are really wonderful is you want to make, well, matrices. But what if you wanted to but words in your matrix? Well, if you really wanted, you could probably use `mathrm` and make a giant ugly mess. But, in reality, your scary mutant matrix would be a table. To avoid this untimely end to your sanity, the `tabular` environment exists. Much like matrices, ampersands are use to separate entries in a row and double backslashes are used to end lines. Additionally, one can control the justification of each of the individual columns and add vertical lines between the columns. In order to avoid continuing to describe tables in hazy terms, we're just going to cut to the chase and have an example.

```
\begin{tabular}{|c|c|}
a & b \\
c & d \\
\end{tabular}
```

makes the following table:

a	b
c	d

Now that was cool, but you've probably noticed that this example table has no horizontal lines and, as a result, is rather ugly. That's where `\hline` comes in. `\hline` draws a horizontal line. So if you want horizontal lines in table above, you'd change things to:

```
\begin{tabular}{|c|c|}
a & b \\ \hline
c & d \\ \hline
\end{tabular}
```

to get

a	b
c	d

Note that you need a line break before the last `\hline` (if you didn't use `\hline`, you wouldn't need a line break after the last set of entries). So now that we have a pretty table, how did we get there?

6.1 Anatomy of the `table` Environment

To start, we see that things are laid out similarly to a matrix, with each cell in a row separated by an & and rows being ended with linebreaks. Notice that we put some arguments after the `\begin{tabular}`. The vertical bars, —, draw the lines down each column. The `c`s center the contents of each cell. The other options are `l` and `r` for left and right justification, respectively. But what if you don't want vertical lines? You simply leave out the bars and continue as usual. By default, entries are centered vertically within the cell, but, not surprisingly, you have the option of changing this to either the top, `t`, or bottom, `b`. This option, should you desire it, goes in square brackets immediately following the `\begin{tabular}`. So, if we wanted a table with the entries at the bottom of the cells, we'd have:

```
\begin{tabular}[b]{|c|c|}
a & b \\ \hline
c & d \\ \hline
\end{tabular}
```

a	b
c	d

6.1.1 Fiddling with Columns

LaTeX automatically decides how wide to make each column, but, as with many things in LaTeX you can override it, should you need to do so. Likewise, you can make a single cell span more than one column. To set the width of a column manually, you, somewhat counterintuitively, change the alignment. Instead of `l, r` or `c`, you use `p{}` and put the width of the column inside the curly braces. `\multicolumn` and `\cline` are two other commands that override the layout of a table. `\multicolumn` lets you have a cell span more than one column. It takes three arguments: the number of columns spanned, the alignment within the cell and the actual contents. `\cline{a-b}` is a variation on `\hline` which draws a horizontal line from column `a` to column `b`. There are other, far more complicated, things that can be done with table alignment, but you'll have to look in a bigger, far more complicated, book for that.

6.2 The `table` Environment and Captions

It would look awfully silly if tables were split in two at page breaks. As you might imagine, tables don't do that. If a table falls on a pagebreak, LaTeX will move it entirely onto the next page. But what if you want to caption your table? You'd want the caption to ride along with the table over the page breaks, wouldn't you? Of course, you could always align the caption yourself, but why do extra work when you could use the `table` environment? `table` doesn't replace `tabular`, rather it encloses `tabular`. Within the `table` environment, you can use `\caption` to caption your table (either above or below the `tabular` environment). Additionally, the `\begin{table}` command takes an optional (although strongly suggested) argument that controls the placement of the table. The options are `h` to place the table where it appears in the source, `t` to place the table at the top of a page, `b` to place the table at the bottom of a page and `p` to place the table on its own page.

6.3 Example

Up to this point, We've had fairly silly, uninteresting examples. This section is just going to be one big example.

Table 6.1: Career Statistics: Brant Brown

Year	Tm	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	BA	OBP	SLG
1996	CHC	29	69	11	21	1	0	5	9	3	3	2	17	.304	.329	.536
1997	CHC	46	137	15	32	7	1	5	15	2	1	7	28	.234	.286	.409
1998	CHC	124	347	56	101	17	7	14	48	4	5	30	95	.291	.348	.501
1999	PIT	130	341	49	79	20	3	16	58	3	4	22	114	.232	.283	.449
2000	FLA	41	73	4	14	6	0	2	6	1	0	3	33	.192	.224	.356
	CHC	54	89	7	14	1	0	3	10	2	1	10	29	.157	.248	.270
	TOT	95	162	11	28	7	0	5	16	3	1	13	62	.173	.237	.309
5 Seasons		424	1056	142	261	52	11	45	146	15	14	74	316	.247	.301	.445

```
\begin{table}[h]
\caption{Career Statistics:  Brant Brown}
\begin{tabular}{r r r r r r r r r r r r r r r r r}
Year & Tm & G & AB & R & H & 2B &3B & HR & RBI& SB& CS& BB & SO & BA & OBP & SLG \\\ hline
1996& CHC& 29 & 69 & 11 & 21 & 1 & 0 & 5 & 9 & 3& 3& 2& 17& .304 & .329& .536 \\
1997& CHC & 46 & 137 & 15 & 32 & 7 & 1& 5 & 15 & 2 & 1 & 7 & 28 & .234 & .286 & .409 \\
1998 & CHC & 124 & 347& 56 & 101 & 17& 7& 14 & 48& 4& 5& 30& 95& .291 & .348 & .501 \\
1999 & PIT & 130 & 341 & 49 & 79 & 20& 3& 16& 58 & 3& 4 & 22 &114 & .232 & .283& .449\\
2000 & FLA & 41 & 73& 4 & 14 & 6& 0 & 2 & 6 & 1 & 0 & 3 & 33 & .192 & .224 & .356\\
\multicolumn{2}{r}{CHC} & 54 & 89 & 7 & 14 & 1 & 0& 3 & 10 & 2 & 1 & 10& 29& .157& .248 & .270
\\
\multicolumn{2}{r}{TOT} & 95 & 162 & 11 & 28 & 7 & 0 & 5 & 16 & 3 & 1 & 13 &62 & .173 & .237 &
.309 \\ \hline
\multicolumn{2}{l}{5 Seasons} & 424 & 1056 & 142 & 261 & 52 & 11 & 45 & 146 & 15 & 14 & 74 & 316
& .247 & .301 & .445
\end{tabular}
\end{table}
```

Chapter 7

Document Classes

To this point, we've been talking pretty much just about how to get LaTeX to make text look the way you want, and we've always included the line `\documentclass{article}`. This is all well and good, and we get what we want, but you might be wondering, why only `article`? Does LaTeX do other things? And what's an article anyway?

LaTeX has several document classes: `article, report, book, slides` and `letter`, each of which are used pretty much for precisely what the name implies. `article` is also used, as you have learned, in more general situations as the generic document class. These are not the only document classes in existence, as individuals and organizations can write there own, but they are the standard ones.

7.1 Article

`article` is used for, as the name implies, journal articles. It is the most basic LaTeX document class, as it has few special commands and quirks. (This may be less because `article` is somehow inherently superior and more because it's the document class everyone thinks of first. But maybe that makes it inherently superior.) The title and associated information is centered by the `\maketitle` command. If one wants to include an abstract, this can be done with `\begin{abstract}...\end{abstract}`.

7.2 Report

`report` is very similar to `article`. However, it is designed to be used for longer documents (like, maybe, reports?) and, as such, has two additional sectioning commands: `\chapter` and `\part`. Additionally, as one might expect, the title information appears on a title page rather than simply at the top of the first page and the abstract gets its own page as well.

7.3 Slides

`slides`, not very surprisingly, makes slides. The slides are meant to be printed and projected, rather than projected from the computer as you might with a PowerPoint. This isn't to say you can't do it and that people don't do it, but people have written other document classes to achieve this end. Additionally, `slide` documents are a bit trickier in their construction that other types of documents. As such, we're going to look at the general outline first and then make some more general comments.

Format of a Slides Document

```
\documentclass{slides}
preamble
\begin{document}
\title{title}
\author{author}
\date{date}
\maketitle \begin{slide}
slide content
\end{slide}
\end{document}
```

The things that one should take from this skeleton of an example is that the document class is called `slides` not `slide` (this is a seemingly small detail, but when your document won't compile because you forgot what the document class was called, that's going to have become an important point); the title information does not go within a slide, it goes before the slides start; lastly, the contents of each slide are set off within a `slide` environment.

There's some information you probably didn't glean from our little skeleton (good job if you figured this out, because the information's not in the example to be observed). First, a fact that you may want to know or just not care about: `slides` uses a different default font that the other LaTeX document classes. Additionally, it is your responsibility to make sure that what you want to go on one slide fits on one slide. If the stuff between the `\begin{slide}` and `\end{slide}` is going to spill over onto another page, LaTeX lets it and simply doesn't number the overflow slide(s).

7.3.1 Overlays and Notes

`slide` documents can have two other environments besides `slide`: `overlay` and `note`. `overlay` is used for making a slide designed to be placed over the slide environment preceding it. As a result, overlays aren't numbered in the same way as slides; the first overlay corresponding to the fourth slide would be numbered `4-a`. In order to get the text on the overlay to line up correctly, it's best to include the text from the slide with the color set to white. This can be accomplished with the `\textcolor{white}`{*text*}.

The `note` environment allows one to make notes corresponding to each slide. Each note is numbered according to its corresponding slide, so the first note corresponding to the fourth slide would be numbered 4-1.

7.4 Letters

The `letter` document class has several commands specific to it. It bears some similarity to the `slide` class because you can have multiple letters in one document. The return address is controlled by the `\address` command in the preamble. Each line of the return address (and the other command arguments in letters, if necessary) is separated by a \\. The start of the `letter` environment is structured as `\begin{letter}`{*addressee*}, where *addressee* is both the name and address of the addressee. Next comes the `\opening{}` command, which has as its argument the opening of the letter. The letter concludes with the `\closing{}` and `\signature{}` commands, where `\closing` taking the closing of the letter and `\signature` the name and/or title appearing under the signature as their arguments, respectively. The `\cc` and `\encl` commands are optional and take the names of those receiving copies and the list of enclosures.

```
\documentclass{letter}
\address{Thomson \\ 2 Carnaby Ave. \\ Berkeley, CA 4709 \\ USA}
\signature{Thomson}
\begin{document}
\begin{letter}{Mr. Santa Claus \\ The Cottage \\ North Pole}
\opening{Dear Santa,}
Perhaps you did not receieve my letter dated December 15, 2004. In it,
I specifically asked for a Jaguar X-5, and an island in the East
Maldives. As you may know, I was saddened this past Christmas when I
received neither car nor island, but instead, a cat. In case you were
wondering Santa, my cat has bit me 6 times in the last hour. I hate
you, Santa.
\closing{Sincerely,}
\end{letter}
\end{document}
```

Thomson
2 Carnaby Ave.
Berkeley, CA 94709
USA

Mr. Santa Claus
The Cottage
The North Pole
Dear Santa,

Perhaps you did not receieve my letter dated December 15, 2005. In it, I specifically asked for a Jaguar X-5, and an island in the East Maldives. As you may know, I was saddened this past Christmas when I received neither car nor island, but instead, a cat. In case you were wondering Santa, my cat has bit me 6 times in the last hour. I hate you, Santa.

Sincerely,
Thomson Nguyen

7.5 Books

While most of you are probably not going to writing any books in the near future it is important that you have the option of typing up your manifesto in LaTeX . LaTeX has been set up so that making professionally looking books is fairly easy. This reader is an example of a book made in LaTeX.

7.5.1 The Basics

In order to make a book, you need to tell the compiler that you are making a book. This is done in the preamble by saying `\documentclass{book}`

7.5.2 Front Matter

The front matter of a book usually includes a cover or title page as well as a table of contents and maybe other sections such as a preface or a short note of thanks. Usually what defines the front matter is that its pages are not numbered with the body of the book and they are usually numbered in roman numerals. The chapters in the front matter (the preface or note of thanks for example) are also not numbered like chapters in the body of the book. LaTeX makes all of these things happen with the simple command `\frontmatter` that should be used right after the `\begin{document}` command.

Title Page

The title page is made in the usual way as described in the Section **??**. The command `\maketitle` should be placed right after the `\frontmatter` command.

Table of Contents and Other Such Things

One of the great things about LaTeX is its wonderful way of assigning references each time it compiles. This is especially useful when you want to make a table of contents. The command `\fronmatter` automatically makes a table of contents including all of the titles of chapters, sections, subsections, and subsubsections with their page numbers. Sometimes the names of these parts are very long so it is best to have a shortened name show up in the table of contents. This can be achieved by the following commmand when declaring one of these parts `\section{The Full Section Name}[The TOC Name]` The table of contents will not appear after the first time you compile your document; it is usually necessary to compile your document 2 or 3 times to make it appear. Similar to the table of contents is a list of figures and a list of tables that are created by `\listoffigures` and `\listoftables`

7.5.3 Main Matter

The main matter of the book you are writing, or the body, is the meat of what you are writing. In this part of the book we want the chapters to be numbered and the page numbering to be in arabic numbers. This is accomplished with the command `\mainmatter`

7.5.4 Appedices

Appendices are really just chapters that are labeled with letters instead of numbers. LaTeX has a nice command `\appendix` that makes these changes happen.

7.5.5 Back Matter

The back matter of your book should contain the index and bibliography. The command for this part of the book is `\backmatter`. This command has no visual affect in the standard document classes but I hear it serves a purpose on occation. I would use it just in case it makes a difference.

7.5.6 Some Tips for Dealing With Big Documents

A book can be very large so here are some recommendations to make your life easier when dealing with large files. The tips work for any size LaTeX file but are especially useful when dealing with large files.

Including Other Files

You can split up your LaTeX document into seperate files to organize yourself better. When including files you use two seperate commands, one that is more basic and the other with more features. The basic command is `\input{filename}`. This just inputs the other file into the LaTeX file that you have now. The command with more options is `\include{filename}`. When using this command, LaTeX will automatically start the new file name on a new page. You can use the `\includeonly{filename1,filename2, . . .}` command that will only include the filenames named. It is important to note that there can't be any spaces between filenames and the commas are not optional. When using `\includeonly` the page breaks made by the `\include` commands will not be moved even when some files are omitted.

Checking Syntax Only

The syntonly package is a very useful package that allows you to check to see if the syntax in your LaTeX document is correct with out making a DVI file. This will help you find errors more quickly in big documents because it is executed much quicker than the standard compiling of a file. In order to use the package you need to have `\usepackage{syntonly}` and `\syntaxonly` in the preamble of your document. When you want to compile your document all you need to do is comment out the second line.

Chapter 8

New Commands

LaTeX also gives you the ability to define your own commands. This ability is seldom necessary as a great variety of packages are distributed with the major LaTeX distributions and many others are available on the internet from LaTeX using organizations and individuals. Chances are, most things you want to do are converged either in LaTeX or an available package.

Should you find yourself in a situation where you need something that is not available in LaTeX or a package, or you have a large pile of commands being used over and over, you can definite a new command in the preamble of your document.

8.1 New Commands

The `\newcommand` command allows you do define a command with a name that is not already in use by LaTeX or any of the packages you are using with your document. Usage is as follows.

```
\newcommand{name}{definition}
```

Additionally, you can also add a third argument as follows to specify the number of arguments for your new command.

```
\newcommand{name}[number of arguments]{definition}
```

The name argument is the name you want for your command. For example `\card` might be the name you want. The definition argument is a set of commands that describes what the command should do. For example, if you want `\card` to double overline it's argument, you would use `\overline{\overline{}}`. Thus, you would add `\newcommand{\card}{\overline{\overline}}}` to the preamble of your document. Alternatively, you could use `\newcommand{\card}[1]{\overline{\overline{#1}}}`. This formation is essential if you want to define a command that you wish to define has more than one argument.

For example, if you have the following document

```
\documentclass[letterpaper]{article}
\newcommand{\card}[1]{\overline{\overline{}}}
```

```
\begin{document}

\[
\card{A}=\aleph_0
\[

\end{document}
```

you get an output that looks like

$$\overline{\overline{A}} = \aleph_0$$

8.2 Redefining commands

If a command already exists, you cannot use it as a name for a new command. However, if you want to redefine how a command acts, you can do this with the `\renewcommand` command. Other than modifying existing commands, rather than defining new commands, `\renewcommand` behaves identically to `\newcommand`.