CIT 592 Online LaTeX Guide[*]

Contents

[*]Developed by the Penn CIS160 staff and used with their cooperation.

1 Introduction

TeX is a typesetting system designed by Donald Knuth in 1978 that allows for platform-independent rendering of documents including mathematical formatting. In the early 1980's, Turing Award Winner Leslie Lamport wrote LaTeX (pronounced "lay-tek" or "luh-tek," and not with an "eks" sound on the X) as a collection of TeX macros and a program that allows users to use a high-level language to render documents using TeX. If all this sounds complicated, don't worry; the gist of it is that LaTeX is a powerful tool that allows us to easily write nice-looking documents.

So why learn LaTeX? For one, it's a relatively easy way to make anything you write (especially math) look beautiful! It's also industry standard if you are considering publishing any papers in really any mathematical or science-related field (and even some others). Plus, it's just a nifty tool that makes it easy to write documents that include mathematical formatting – once you get comfortable with it, you may find that you start using it for other classes, or even other activities! Also it's required for CIT 592.

The remainder of the guide will focus on LaTeX for writing TeX documents (i.e. `.tex` files). But this isn't the only way we use LaTeX – some webpages will also allow you to write LaTeX code and render it on the page!

In particular, our discussion forum here in Coursera, accommodates LaTeX code. We highly recommend you familiarize yourself with LaTeX enough to incorporate it in your answers. Note that there are some subtle differences on Coursera – we enter mathmode by enclosing with double dollar signs (`$$...$$`) rather than single, and some packages are pre-loaded, while others can't be used. There's also a couple other differences, but for the most part, you won't encounter any besides the dollar signs.

Sometimes we will reference concepts to be covered in the course for exposition, so don't be worried if you don't understand something here (maybe if you still don't understand it after the class is over). On a similar note, just because a concept appears here doesn't mean that you can use it as justification for homework or exams (because we skip the proofs here!). There's a lot of detail here, and we'll aim to cover everything you need for CIT 592, plus a little more just in case. But we have faith that you'll pick it up quickly once you actually start writing. With a bit of practice, you'll be creating beautiful documents with well-formatted mathematical expressions in no time!

2 LaTeX Basics

First off, you're going to need to set up a LaTeX editor. We recommend following the setup guide found in Week 1. You'll also find there an interactive guide to LaTeX, which we also recommend browsing through. Once you've gotten all set up, the rest of this guide will be much more helpful!

Your LaTeX editor will probably have a text editor (for editing `.tex` files), and an output display (which generates the output from the `.tex` input, usually in the form of a PDF). The structure of a `.tex` document will generally be a preamble followed by the document body. The preamble will contain any packages used, macros defined, as well as some general information about the document (title, author, etc.). To import packages, you'll use the `\usepackage{packagename}` command; we'll cover defining macros later, but you probably won't ever need them. The document itself will be enclosed in `\begin{document} ... \end{document}` tags. This is the portion that will be rendered in the final result. In general, you won't have to mess with preambles too much (just the document body). One thing to watch out for – you should always make sure to write your name and recitation number where specified in the homework templates!

In normal text mode, most text will appear essentially how you want it to; some special symbols will give you problems. Where LaTeX really makes its mark is in mathmode (Section 3.1). There's a few special characters you should be aware of: the backslash (`\`) is used to denote commands or escape special characters; the dollar sign (`$`) is used to enter and leave inline mathmode; and the percent sign (`%`) is used to comment the remainder of the line. Additionally, curly braces (`{}`) are used to tell LaTeX that "these things belong together" and to prevent splitting them up.

3 Environments

In general, environments are used to provide a specific form of formatting. Most environments use in the following format (notable exceptions will be duly noted):

Input:

```
\begin{environmentname}
  ...
  contents
  ...
\end{environmentname}
```

Following this are some common environments which you will likely (but not necessarily) encounter and use during the course of your CIT 592 career.

3.1 Inline Math Mode

One of the primary benefits of using LaTeX is that it renders mathematical text nicely. To enter inline math mode, surround the desired text with single dollar signs $ ... $ (this is an exception to the general format described above). Consider the following example:

Input:

```
Consider f(x) = 2x + 1. If we set x = 1, we see that f(x) = 3.
```

Output: Consider f(x) = 2x + 1. If we set x = 1, we see that f(x) = 3.

If we were to instead use inline math mode, we would input:

Input:

```
Consider $f(x) = 2x + 1$. If we set $x = 1$, we see that $f(x) = 3$.
```

Output: Consider $f(x) = 2x + 1$. If we set $x = 1$, we see that $f(x) = 3$.

Doesn't that look nicer than the first? In particular, all of the math mode text is stylized differently, so that the reader can tell where "the math" occurs. This is helpful, for example, when naming variables; italicized variables will stand out more obviously. Consider the (somewhat ridiculous) short blurb:

> Let a denote the number of variables used, and the denote the number of words. We note that we used a variables, where a single variable would suffice. Indeed, the variable can be used to represent the words simultaneously, making it unnecessary to use a variables.
>
> Let a denote the number of variables used, and the denote the number of words. We note that we used a variables, where a single variable would suffice. Indeed, the variable can be used to represent the words simultaneously, making it unnecessary to use a variables.

While neither necessarily makes sense, the second is significantly easier to parse. (Note that this also hints at another important style consideration – name your variables appropriately!)

In math mode, LaTeX will ignore any whitespace present in the input. That means that given the input:

Input:

```
$Here is
a
  sentence   .$
```

what would actually appear in the document would be:

Output: $Hereisasentence.$

In general, you should avoid writing text in mathmode, as this will make it virtually unreadable. One solution to this issue is the `\text{...}` command, discussed below (5.4), although if you're using inline math mode, it's generally easier to just split the math into two separate sections.

Additionally, math mode allows you to use a variety of different commands for symbols you might find helpful (or even necessary) in your work. See Section 5 for some examples and more information.

3.2 Display Math Mode

In some cases, inline math mode will look unappealing – consider $\sum_{i=1}^n i$. How can we solve this? We use display math mode! For single-line math mode statements, you can enclose the desired text in `\[...\]` opening and closing tags (note that this is another exception to the aforementioned format). This will put the contents in their own line in display math mode (which is generally nicer than inline). It's typically a good idea to use this when you have a longer expression you want to put in math mode, to avoid cluttering up paragraphs with excessive scary-looking notation. For example, the text:

Input:

```
We recall the equation $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.
In this case, we have $a = 1, b = -5, c = -24$. Plugging these
values in, we find $x = -3$ and $x = 8$ as our solutions.
```

Output: We recall the equation $x = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$. In this case, we have $a = 1, b = -5, c = -24$. Plugging these values in, we find $x = -3$ and $x = 8$ as our solutions.

Using the brackets for math mode for the first bit of math will result in the following:

Input:

```
We recall the equation \[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \] .
In this case, we have $a = 1, b = -5, c = -24$. Plugging these
values in, we find $x = -3$ and $x = 8$ as our solutions.
```

Output: We recall the equation

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In this case, we have $a = 1, b = -5, c = -24$. Plugging these values in, we find $x = -3$ and $x = 8$ as our solutions.

The fully expanded expression draws emphasis and is much easier to read (the fraction, in particular, is much larger; it's also easier to distinguish the $\pm$). We often use display-style for this very purpose, as well as to break up large chunks of text. If you want to use inline math mode, but with display-style formatting, you can use the command `\displaystyle` immediately before the argument. For example:

`$\sum_{i=1}^n i$` will render as $\sum_{i=1}^n i$.

`$\displaystyle\sum_{i=1}^n i$` will render as $\displaystyle\sum_{i=1}^n i$.

Another common application of `\displaystyle` is for fractions (although there is a separate shortcut for this). Note that using `\displaystyle` might mess up your line spacing – this is okay!

3.3 Align

The align environment is a powerful tool for grouping together series of equations, and perhaps the most common environment you will encounter in CIT 592. The align environment essentially takes all the lines given as inputs as individual mathmode lines, and aligns them based on the positions of the special align character – your friendly neighborhood ampersand (`&`). To see how we can use this environment, consider the following problem. Suppose we want to show that $(x+1)(x+2)(x+6) = x(x+4)(x+5)+12$. So our first attempt might look something like this:

Input:

```
$(x + 1)(x + 2)(x + 6) = (x^2 + 3x + 2)(x + 6) = x^3 + 9x^2 + 20x + 12
= x(x^2 + 9x + 20) + 12 = x(x^2 + 4x + 5x + 20) + 12
= x(x(x + 4) + 5(x + 4)) + 12 = x(x + 4)(x + 5) + 12$
```

Output: $(x+1)(x+2)(x+6) = (x^2+3x+2)(x+6) = x^3+9x^2+20x+12 = x(x^2+9x+20)+12 = x(x^2+4x+5x+20)+12 = x(x(x+4)+5(x+4))+12 = x(x+4)(x+5)+12$

Wow, that is... kind of a mess. There's a lot of math happening in not much space. To make this readable, we now try the following snippet, where each line is in display math mode:

Input:

```
\[(x + 1)(x + 2)(x + 6) = (x^2 + 3x + 2)(x + 6)\]
\[= x^3 + 9x^2 + 20x + 12\]
\[= x(x^2 + 9x + 20) + 12\]
\[= x(x^2 + 4x + 5x + 20) + 12\]
\[= x(x(x + 4) + 5(x + 4)) + 12\]
\[= x(x + 4)(x + 5) + 12\]
```

Output:

$$(x+1)(x+2)(x+6) = (x^2+3x+2)(x+6)$$

$$= x^3+9x^2+20x+12$$

$$= x(x^2+9x+20)+12$$

$$= x(x^2+4x+5x+20)+12$$

$$= x(x(x+4)+5(x+4))+12$$

$$= x(x+4)(x+5)+12$$

This is a bit better, as there's more room – each line indicates a single logical step. But note that each of the lines is centered. Since they have different lengths, this results in a wobbly and somewhat disorienting display (especially if the difference is greater than in our example). To remedy this, we use the align environment!

Input:

```
\begin{align*}
    (x + 1)(x + 2)(x + 6) &= (x^2 + 3x + 2)(x + 6) \\
    &= x^3 + 9x^2 + 20x + 12 \\
    &= x(x^2 + 9x + 20) + 12 \\
    &= x(x^2 + 4x + 5x + 20) + 12 \\
    &= x(x(x + 4) + 5(x + 4)) + 12 \\
    &= x(x + 4)(x + 5) + 12
\end{align*}
```

Output:

$$\begin{aligned}(x+1)(x+2)(x+6) &= (x^2+3x+2)(x+6)\\ &= x^3+9x^2+20x+12\\ &= x(x^2+9x+20)+12\\ &= x(x^2+4x+5x+20)+12\\ &= x(x(x+4)+5(x+4))+12\\ &= x(x+4)(x+5)+12\end{aligned}$$

Much neater! We use `\begin{align*} ... \end{align*}` here to avoid equation numbering; if we leave off the asterisks, then each equation will be numbered. Note that the equations are all lined up so that the `&`'s are all in the same vertical location (though they don't actually appear in the text). Also note that we have to manually include a line break (`\\`) after each line – otherwise, the the environment won't split them automatically (remember, LaTeX ignores whitespace in math mode). But if you have extra lines of just whitespace, LaTeX won't know how to align them and it'll throw an error.

You can also use this tool to split a single equation over multiple lines (instead of aligning on =, you might assign on something like +, with a bit of horizontal spacing (7.4), say a `\quad`, thrown in there). Finally, you can align multiple segments of each row by adding the appropriate ampersands.

Input:

```
\begin{align*}
    x + 2y &= 1      &  a + 2b - c &= 0  \\
    2x - y &= 7      &  ab + 2c &= y  \\
    &  &  a + 2bc &= 3
\end{align*}
```

Output:

$$\begin{aligned} x+2y &= 1 & a+2b-c &= 2\\ 2x-y &= 7 & ab+2c &= y\\ & & a+2bc &= 3\end{aligned}$$

But be careful that each line contains the same number of `&`'s; otherwise, you might end up with some unexpected results! If you feel the alignment is off, try to mess around with `&` placement a bit.

3.3.1 Commenting

Sometimes, you may want to include a line of explanation within your align environment. However, you may find that creating two separate environments disrupts the alignment. From the example above:

Input:

```
\begin{align*}
    (x + 1)(x + 2)(x + 6) &= (x^2 + 3x + 2)(x + 6) \\
    &= x^3 + 9x^2 + 20x + 12
\end{align*}
Note that we can pull out an $x$ from the first three terms:
\begin{align*}
    &= x(x^2 + 9x + 20) + 12 \\
    &= x(x^2 + 4x + 5x + 20) + 12 \\
    &= x(x(x + 4) + 5(x + 4)) + 12 \\
    &= x(x + 4)(x + 5) + 12
\end{align*}
```

Output:

$$\begin{aligned}(x+1)(x+2)(x+6) &= (x^2+3x+2)(x+6)\\ &= x^3+9x^2+20x+12\end{aligned}$$

Note that we can pull out an x from the first three terms:

$$\begin{aligned}&= x(x^2+9x+20)+12\\ &= x(x^2+4x+5x+20)+12\\ &= x(x(x+4)+5(x+4))+12\\ &= x(x+4)(x+5)+12\end{aligned}$$

See how the alignment shifts? Both sets of lines are individually centered on the = signs, but not with each other. For instances where a bit of justification for a particularly tricky step is needed, `\intertext{...}` comes to the rescue!

Input:

```
\begin{align*}
    (x + 1)(x + 2)(x + 6) &= (x^2 + 3x + 2)(x + 6) \\
    &= x^3 + 9x^2 + 20x + 12 \\
    \intertext{Note that we can pull out an $x$ from the first
    three terms:}
    &= x(x^2 + 9x + 20) + 12 \\
    &= x(x^2 + 4x + 5x + 20) + 12 \\
    &= x(x(x + 4) + 5(x + 4)) + 12 \\
    &= x(x + 4)(x + 5) + 12
\end{align*}
```

Output:

$$\begin{aligned}(x+1)(x+2)(x+6) &= (x^2+3x+2)(x+6)\\ &= x^3+9x^2+20x+12\\ \intertext{Note that we can pull out an x from the first three terms:} &= x(x^2+9x+20)+12\\ &= x(x^2+4x+5x+20)+12\\ &= x(x(x+4)+5(x+4))+12\\ &= x(x+4)(x+5)+12\end{aligned}$$

So our alignment is preserved through the comment! Also note that within the intertext environment, we're automatically in text mode (but can enter math mode normally).

For shorter notes, another options is viable – the `\tag{...}` command, which will appear in parentheses inline, aligned right, and in text mode:

Input:

```
\begin{align*}
    (x + 1)(x + 2)(x + 6) &= (x^2 + 3x + 2)(x + 6) \\
    &= x^3 + 9x^2 + 20x + 12 \\
    &= x(x^2 + 9x + 20) + 12 \tag{Pulling out an $x$} \\
    &= x(x^2 + 4x + 5x + 20) + 12 \\
    &= x(x(x + 4) + 5(x + 4)) + 12 \\
    &= x(x + 4)(x + 5) + 12
\end{align*}
```

Output:

$$\begin{aligned}(x+1)(x+2)(x+6) &= (x^2+3x+2)(x+6)\\ &= x^3+9x^2+20x+12\\ &= x(x^2+9x+20)+12 && \text{(Pulling out an } x\text{)}\\ &= x(x^2+4x+5x+20)+12\\ &= x(x(x+4)+5(x+4))+12\\ &= x(x+4)(x+5)+12\end{aligned}$$

Note that using the `\tag{}` will remove numbering from that line. To suppress numbering on an individual line if numbering is active (in the non-starred environment), add `\nonumber` to the end of the line.

Input:

```
\begin{align}
    (x + 1)(x + 2)(x + 6) &= (x^2 + 3x + 2)(x + 6) \nonumber \\
    &= x^3 + 9x^2 + 20x + 12 \tag{expanding}\\
    &= x(x + 4)(x + 5) + 12
\end{align}
```

Output:

$$\begin{aligned}(x+1)(x+2)(x+6) &= (x^2+3x+2)(x+6)\\ &= x^3+9x^2+20x+12 && \text{(expanding)}\\ &= x(x+4)(x+5)+12 && (1)\end{aligned}$$

If you want to refer to specific equations, equation numbers can be helpful; otherwise, we usually leave them off to avoid clutter. To mark something for later reference, use `\label{XYZ}`; then use `\ref{XYZ}` to refer to it later (this also adds a link in digital versions). Adding labels will be helpful in case you end up changing the document later; you won't have to correct any hard-coded labels.

Input:

```
\begin{align}
    (x + 1)(x + 2)(x + 6) &= (x^2 + 3x + 2)(x + 6) \nonumber\\
    &= x^3 + 9x^2 + 20x + 12 \tag{expanding} \\
    &= x(x + 4)(x + 5) + 12 \label{eqn:result}
\end{align}
So from equation \ref{eqn:result}, we see that...
```

Output:

$$\begin{aligned}(x+1)(x+2)(x+6) &= (x^2+3x+2)(x+6)\\ &= x^3+9x^2+20x+12 && \text{(expanding)}\\ &= x(x+4)(x+5)+12 && (2)\end{aligned}$$

So from equation 2, we see that...

Labels can be defined in math mode or text mode and can also be useful for generally linking throughout the document (it's how we make the links between sections in this guide, for example).

3.4 Quotation/Quote

Sometimes, you just want a little bit of space. Maybe you're working on a lemma and want to emphasize that the proof is separate from the proof of the main result. In this case, you'll likely use the the `\begin{quote} ... \end{quote}` or `\begin{quotation} ... \end{quotation}` environments. The two are almost identical, except for some minor differences in indentation and spacing; it's basically a matter of preference. Both environments will increase the margins a bit on both sides, providing a slightly narrower output (nice for smaller results).

3.5 Enumerate

Numbered lists are a great way to organize your ideas and keep track of thoughts. They're also useful for, say, labelling homework problems. Thankfully, LaTeX comes with this capability built-in, through the `\begin{enumerate} ... \end{enumerate}` environment. Within this environment, use `\item` to advance the numbering; otherwise, this is just like writing normally. You can also nest these environments; LaTeX will automatically choose appropriate labels, and keep track of the nesting on its own

Input:

```
\begin{enumerate}
    \item Category 1
    \begin{enumerate}
        \item Example 1
        \item Example 2
    \end{enumerate}
    \item Category 2
    \begin{enumerate}
        \item Example 3
        \item[c)] Example 4
            \begin{enumerate}
                \item Explanation
            \end{enumerate}
    \end{enumerate}
\end{enumerate}
```

Output:

```
1. Category 1

   (a) Example 1
   (b) Example 2

2. Category 2

   (a) Example 3
    c) Example 4
           i. Explanation
```

Note that you can specify the label for a particular item by including the desired label in brackets, as shown above. There are also other packages that allow you to further customize numbering (if you're interested, check out `enumitem`), but we generally won't need these.

3.6 Itemize

What if you wanted a list without numbers? Well, LaTeX can do that too! The `itemize` environment works just like `enumerate`, except it uses bullets instead of numbers.

Input:

```
\begin{itemize}
    \item Category 1
    \begin{itemize}
        \item Example 1
        \item Example 2
    \end{itemize}
    \item Category 2
    \begin{itemize}
        \item Example 3
        \item[;] Example 4
            \begin{itemize}
                \item Explanation
            \end{itemize}
    \end{itemize}
\end{itemize}
```

Output:

- Category 1
  - Example 1
  - Example 2
- Category 2
  - Example 3
  - ; Example 4
    - $*$ Explanation

Again, note that we can assign individual items specific bullets using the square brackets.

3.7 Cases

The `\begin{cases}...\end{cases}` environment essentially creates a self-contained miniature `align*` environment, with a large left brace which scales to encompass the entire selection. This environment can only be used if you're already in math mode. You'll probably use this most commonly for piecewise functions. For example, consider the following function:

Input:

```
\[
f(x) =
\begin{cases}
    x^3+1 &\text{if } x \leq 0 \\
    x^2 &\text{if } x > 0
\end{cases}
\]
```

Output:

$$f(x) = \begin{cases} x^3+1 & \text{if } x \leq 0 \\ x^2 & \text{if } x > 0 \end{cases}$$

Note that you can extend this to as many lines as you want, and, just like with the `align*` environment, include multiple columns of alignment (the same spacing rules apply).

4 Images and Figures and Diagrams, Oh My!

You may, in the course of your writing, find it necessary to include supplementary figures or diagrams. Perhaps a particular graph example can be drawn out to emphasize a point or provide an example. Or maybe you just happened to take an amazing selfie and needed somewhere to share it. Luckily, LaTeX makes it fairly straightforward to do so. You'll need to import the `graphicx` package, by including `\usepackage{graphicx}` somewhere in your preamble. Once you do, you can use `\includegraphics{filename}` to insert images into your document. You don't need to specify the extension. If you want to use an image in a different directory, you can also specify the full path. You can also provide additional scaling and rotation arguments in brackets; for example `\includegraphics[scale=1.5, angle=45]{filename}` to scale and rotate the image or `\includegraphics[height=2cm, width=\textwidth]{filename}` to specify both dimensions (if you only give one, the other will scale to maintain aspect ratio).

So this gets your image on the page. But that doesn't mean it'll look nice. While LaTeX generally gives you a lot of flexibility on how you lay out your document, it really likes to try to place images on its own (as an aside, LaTeX is really particular about not breaking things up, which is why you will often see images ending up flying around the document). While it generally renders objects in the order it encounters them, its image placement mechanism often does not, which can lead to a great deal of puzzlement. To help you deal with this, we'll teach you a magic word: **!htbp**. But how do we use this magical power?, you might ask. We'll enclose relevant images (or text) in a `\begin{figure} ... \end{figure}` environment. Here's the basic anatomy of a `figure` environment:

Input:

```
\begin{figure}[!htbp]
\centering % centers the figure; default alignment is left
\includegraphics{my-picture}
\caption{Caption} % optional, will display below the picture
\label{fig:my_label} % optional, allows \ref to link to the figure
\end{figure}
```

LaTeX will automatically keep track of the figure numbers and add the appropriate label to the caption. If you want the caption to appear above the image, just flip the order of these commands.

So that's all well and good. But what does the "!htbp" do? Well, the ! first tells LaTeX to ignore its usual rules for trying to place things. The remaining letters define the priority of different locations. LaTeX will first try to place the image "***h***ere," that is, as close to the given location in the text as it can. Failing this, it'll go to the ***t***op of the page, then the ***b***ottom, or a new ***p***age.

This can be one of the most finicky aspects of using LaTeX, so don't be afraid to ask questions. And in the long run, it really doesn't matter where your figures end up, so long as we can grade them (you may want to indicate where to find them if it's not obvious).

4.1 Tables

Tables can be a great way of organizing information... for example, truth tables. LaTeX tables use the `\begin{tabular} ... \end{tabular}` environment, which requires using alignment characters (`&`) and line breaks (\\\\) appropriately. Each cell will be in text mode by default, although you can enter mathmode as well. As an alternative, you can use the `\begin{array} ... \end{array}` environment, which is in mathmode by default. You will have to specify also the number of columns, alignment of each column (***l***eft, ***c***enter, ***r***ight), and placement of vertical bars immediately after the `\begin{tabular}` tag. Also note that any horizontal lines must be manually entered with `\hline`, which appears at the beginning of its line.

Input:

```
\begin{tabular}[h]{|c|c|c||c|}
\hline
    $p$ & $q$ & $r$ & $p \land q \land r$ \\ [0.5 ex] \hline
    $T$ & $T$ & $T$ & \\
    $T$ & $T$ & $F$ & \\
    $T$ & $F$ & $T$ & \\
    $T$ & $F$ & $F$ & \\
    $F$ & $T$ & $T$ & \\
    $F$ & $T$ & $F$ & \\
    $F$ & $F$ & $T$ & \\
    $F$ & $F$ & $F$ & \\
\hline
\end{tabular}
```

Output:

p	q	r	$p \land q \land r$
T	T	T	
T	T	F	
T	F	T	
T	F	F	
F	T	T	
F	T	F	
F	F	T	
F	F	F	

Tables in particular have a lot of advanced customization available (shading, merging cells, etc.). There's also online table generators if you don't want to build your own manually. Once you've made the table, you can use the techniques described above with `\begin{figure} ... \end{figure}` to move and adjust it. You could also just directly wrap it in `\begin{center} ... \end{center}` tags, which usually works.

5 List of Common Commands

Here's a list of the most common commands you'll encounter in CIT 592. If you need something and can't find it here, both our discussion forums on Coursera and Google will be helpful resources! Also try out Detexify to look up specific symbols.

5.1 Math Mode Commands

These are commands which are used for formatting mathematical notations. For brevity, we'll leave off the mathmode delimiters here and just provide what you would write if you're already in mathmode. Note that, if you're not in mathmode, it's likely that LaTeX will get angry at you for using these. This should be your go-to section for LaTeX notation questions, organized roughly in the order in which you will encounter them.

General Notation:

- `a^b` – a^b – exponentiation.
- `a_b` – a_b – subscript.
- `\times` – $\times$ – multiplication cross.
- `\cdot` – $\cdot$ – multiplication dot.
- `\frac{a}{b}` – $\frac{a}{b}$ – fraction.
- `\sqrt{x}` – $\sqrt{x}$ – square root.

- `\sqrt[n]{x}` – $\sqrt[n]{x}$ – n^{th} root.
- `\leq` – $\leq$ – less than or equal to.
- `\geq` – $\geq$ – greater than or equal to.
- `\not \geq` – $\not\geq$ – the `\not` slashes out the next symbol. Some symbols have their own custom negations (like `\ngeq` – $\ngeq$), but the differences are small enough that using `\not` is a good rule of thumb.
- `\neq` – $\neq$ – not equals (here's one of those custom negations).
- `{}` – grouping things into a single unit (for example, `2^{10}` gives 2^{10} while `2^10` gives 2^10). You'll use these more than you expect!
- `\infty` – ∞ – infinity (and beyond!).
- `\lfloor x \rfloor` – $\lfloor x \rfloor$ – floor function.
- `\lceil x \rceil` – $\lceil x \rceil$ – ceiling function.
- `\dots` – $\dots$ – dots!
- `\circ` – $\circ$ – function composition.
- `\ell` – ℓ – cool-looking "l" (for readability)
- `\sum_{k=1}^n` – $\sum_{k=1}^n$ – summation (see 7.5 for more details).
- `\prod_{k=1}^n` – $\prod_{k=1}^n$ – product.
- `\bigcup_{k=1}^n` – $\bigcup_{k=1}^n$ – union over indices.
- `\bigcap_{k=1}^n` – $\bigcap_{k=1}^n$ – intersection over indices.

Set Notation:

- `\{ ... \}` – $\{...\}$ – rendering braces (since braces alone are used for grouping, they must be escaped).
- `\emptyset` – $\emptyset$ – the empty set.
- `\in` – $\in$ – is an element of.
- `\notin` – $\notin$ – is not an element of (another custom negation!).
- `\subseteq` – $\subseteq$ – subset.
- `\subsetneq` – $\subsetneq$ – strict subset (sometimes, we'll use `\subset` ($\subset$) for this).
- `\supseteq` – $\supseteq$ – superset (and similar notations give the corresponding variants of above).
- `\cup` – $\cup$ – union.
- `\cap` – $\cap$ – intersection.
- `\setminus` – $\setminus$ – set difference.
- `\times` – $\times$ – Cartesian product.
- `|S|` – $|S|$ – cardinality (size) of S.

Note that some concepts can be written multiple ways. If unsure, stick to the one presented in class; because of different conventions, you might encounter alternative notations, so we list some of those here.

- `S^c` – S^c – the complement of S.

- `\overline{S}` – $\overline{S}$ – also the complement of S.
- `\mathcal{P}(S)` – $\mathcal{P}(S)$ – the power set of S (requires `amsfonts`).
- `2^S` – 2^S – also the power set of S.
- `S \oplus T` – $S \oplus T$ – the symmetric difference of S and T.
- `S \triangle T` – $S \triangle T$ – also the symmetric difference of S and T.

Logical Notation:

- `\land` – $\land$ – logical conjunction ("and").
- `\lor` – $\lor$ – logical disjunction ("or").
- `\lnot` – $\lnot$ – logical negation ("not").
- `\oplus` – $\oplus$ – exclusive "or" ("XOR").
- `\implies` – $\implies$ – logical implication.
- `\impliedby` – $\impliedby$ – reverse logical implication.
- `\iff` – $\iff$ – biconditional implication ("if and only if").
- `\equiv` – $\equiv$ – equivalence.
- `\forall` – $\forall$ – "for all" quantifier.
- `\exists` – $\exists$ – "there exists" quantifier.

Miscellaneous Notation:

- `\Pr[E]` – $\Pr[E]$ – probability.
- `\Pr[A \mid B]` – $\Pr[A \mid B]$ – conditional probability.
- `\to` – $\to$ – arrow for functions.
- `\mapsto` – $\mapsto$ – arrow for function action on specific element.
- `\chi(G)` – $\chi(G)$ – chromatic number (also note that all Greek letters in both upper- and lowercase are available similarly, unless they coincide with an English letter).

5.2 AMSmath Commands

These commands will require you to use the amsmath package (include the line `\usepackage{amsmath}` in the file preamble); for the most part, this will be in our default preamble, so you won't need to worry about it.

- `$\binom{n}{k}$` – $\binom{n}{k}$ – "n choose k."
- `\begin{align} ... \end{align}` – yep, you'll need amsmath to align things!
- `\intertext{...}` – goes hand-in-hand with the above command and similar.
- `\begin{cases} ... \end{cases}` – same thing here (cases is actually kind of a mini-align environment).
- `$\dfrac{x}{y}$` – $\dfrac{x}{y}$ – display mode fraction (i.e. a big one).
- `\begin{pmatrix} ... \end{pmatrix}` – if for some reason you find yourself needing a matrix. Follow the spacing instructions for tables (4.1) within the environment.

- `$\boxed{x}$` – $\boxed{x}$ – boxes the selection (useful for highlighting answers!)
- `$\cfrac{1}{2 + \cfrac{1}{3 + \cfrac{1}{4}}}$` – $\cfrac{1}{2 + \cfrac{1}{3 + \cfrac{1}{4}}}$ – nicer formatting for continued fractions.
- `\substack` – see Section 7.5.
- `\limits` — again, check out the info on summations; this can also be used for operators such as `\max`.
- `$\boldsymbol{\delta}$` – $\boldsymbol{\delta}$ – bolds math symbols, which `\mathbf{}` doesn't.

5.3 AMSsymb Commands

Like above, you'll need to use the amssymb package here (`\usepackage{amssymb}`); again, we'll usually have this loaded already for your convenience.

- `$\mathbb{...}$` – this and similar fonts (mathcal, mathscr, mathfrak, etc.) require amsfonts, which amssymb automatically loads.
  - `$\mathbb{N}$` – $\mathbb{N}$ – the natural numbers (yes, 0 is a natural number).
  - `$\mathbb{Z}$` – $\mathbb{Z}$ – the integers.
  - `$\mathbb{Z}^+$` – $\mathbb{Z}^+$ – the positive integers.
  - `$\mathbb{Q}$` – $\mathbb{Q}$ – the rationals.
  - `$\mathbb{R}$` – $\mathbb{R}$ – the reals.
- `$\varnothing$` – $\varnothing$ – alternative empty set (it's rounder!).
- `$\nmid$` – $\nmid$ – "does not divide"
- `$\checkmark$` – $\checkmark$ – check mark.
- `$\leadsto$` – $\leadsto$ – wavy arrow ("has a path to"). Use `latexsym` package for different style of arrow.

5.4 Text Formatting

Some options on different font options.

- `\text{abc}` – abc – in mathmode, allows you to write normal-looking text with spaces.
- `\textbf{abc}` – **abc** – bold text.
- `\textit{abc}` – *abc* – italicized text.
- `\underline{abc}` – $\underline{\text{abc}}$ – underlined text.
- `\textsf{abc}` – abc – sans-serif text.
- `\texttt{abc}` – `abc` – typewriter text.
- `\textcolor{blue}{abc}` – abc – colored text (requires `xcolor` package).
- `\colorbox{yellow}{abc}` – abc – highlighted text (requires `xcolor` package).

Note that these can sometimes be nested: **Wow!**
There's also some corresponding math font variants if you want to get fancy (these mostly require `amsmath`):

- `$\mathbf{x^2+1}$` – $\mathbf{x^2+1}$ – bold math.

- `$\mathit{x^2+1}$` – $\mathit{x^2+1}$ – italicized math.
- `$\underline{x^2+1}$` – $\underline{x^2+1}$ – underlined math.
- `$\mathsf{x^2+1}$` – $\mathsf{x^2+1}$ – sans-serif math.
- `$\mathtt{x^2+1}$` – $\mathtt{x^2+1}$ – typewriter math.

6 General Proof-Writing Tips

This section contains a brief list of tips and advice on how to write effective and clean proofs. A major focus of the course will be teaching you how to write proofs, so don't worry about this *too* much initially. We've included here a couple good general rules to follow to steer your writing efforts and save yourself (and possibly your grader!) some headaches. Note that we do grade for justification as well; a correct answer alone is usually insufficient for full credit. Some of these tips are for specific techniques or methods; they'll make more sense as we cover the relevant material in class.

- **Don't Skip Steps!** Read your proof and make sure each line follows directly from something already proven or some earlier step. If not, then add a line of justification in and continue.
- **Write Only What You Need!** If you end up not using a result, don't leave it in the proof. A good proof is sufficient to prove the claim and is free of extra content that can mislead the reader.
- **Know Your Wants!** Keep in mind what you want to prove – your "want to show" (WTS). Doing so will help keep you from getting sidetracked when trying different techniques and approaches.
- **Use Math Mode!** There's a reason this guide exists! Using the mathematical notation covered in class (which has often been developed in order to make proofs more straightforward and rigorous) will greatly enhance readability. This will help you check your own work to pick up any mistakes you might make. If the formatting is very bad, you might end up accidentally making some claims which are completely untrue!
- **Learn the Notation!** In general, try not to write "word proofs" that avoid the notation and definitions from class. It's fine to use words, and you should justify all your steps, which often requires words, but going out of your way to use words in place of the notation we use in class will often lead to a loss of rigor as some details are lost.
- **Road Map!** Be explicit about what methods you're using. In some cases, this can help you earn points even if you don't have a full solution.
- **Say Everything!** Don't assume that your reader will fill in the details. For example, if you claim two sets are disjoint, you may want to briefly explain why; when using Linearity of Expectation on a sum of random variables, you should always explicitly define the individual random variables and state something like $X = \sum X_i$ before you apply LOE.
- **PHP!** Clearly define your pigeons and holes when using PHP.
- **Induction!** Here's a couple rules to follow for induction proofs and save you a few points:
  - Change your variable in the induction hypothesis.
  - Give appropriate bounds and quantifiers in the induction hypothesis.
  - Start with an arbitrary size-$(k+1)$ instance and find the size-k subproblem.
  - Prove the appropriate base case(s).
  - If you say $P(k)$, be sure to explicitly define it (unless it's given to you).
  - If you don't use the induction hypothesis within the induction step, you're not doing induction.
  - Make sure the type of the induction hypothesis (strong/weak) matches what you use.

- **Probably Definitely!** Always define your probability space and relevant events.
- **Rubber Ducky, You're the One!** Similar to how rubber duck debugging works, rubber duck proof-reading is an effective way of checking your own work and seeing if your proof is sufficiently rigorous.
- **Quotable Quotations!** This technically is more of a general tip, but LaTeX handles quotations strangely. Normal double quotes will look like ”this.” Eww! To get opening quotes, use the (‘) key (next to the number 1 on your keyboard); closing quotes (’) are made with the normal apostrophe.

7 Advanced LaTeX

This is more of an “extra credit” section for those interested in exploring a bit more of what LaTeX really has to offer. You won't necessarily *need* to use these for CIT 592, but they can be effective formatting tools.

7.1 Custom Commands

LaTeX also gives you the ability to define your own commands. If you find yourself repeatedly using the same notation, but not wanting to type it out every time, you can define a handy dandy macro to do it for you! Or you can define a custom command to get some formatting *juuust right*, in a way that the standard packages don't account for. An example of the first case would be as follows: Say you happen to be dealing with the real numbers $\mathbb{R}$ a lot and don't feel like writing out `$\mathbb{R}$` every time you want to use it. Instead, you can define:

Input:
```
\newcommand{\R}{\mathbb{R}}
```

near the top of your document and just write `$\R$` whenever you need it, saving you seconds each time!

Or maybe you want to redefine some already existing notation. If you want to use a shorter arrow for implication, you might include in your preamble:

Input:
```
\renewcommand{\implies}{\Rightarrow}
```

Then, instead of using $\Longrightarrow$, LaTeX will render `\implies` as $\Rightarrow$.

Maybe you want to use some notation that isn't already implemented (or at least, you can't find any existing commands online). In this case, we can define our own notation using custom commands! We'll use `\genfrac` as an example here (you won't need to be too familiar with this, since most commands you need will already be defined for you).

Input:
```
\newcommand{\multichoose}[2]{\left(\!\genfrac{(}{)}{0pt}{0}{#1}{#2}
\!\right)}

$\multichoose{3}{1}$
```

Will now give us: $\left(\!\binom{3}{1}\!\right)$. Observe that we supply the number of arguments within the brackets; then refer to the arguments within the command using the `#` symbol and index of the argument.

7.2 Toggles

Toggles can be useful to switch the document between two different “modes”. Toggling can be convenient, but keep in mind that you're working with two versions of the same document at once, particularly if there's sensitive information that you don't want appearing.

To define a toggle, use `\newtoggle{ABC}`. If you want the toggle on, you'll have to use `\toggletrue{ABC}`. But this doesn't mean anything will happen! To make toggles useful, we have to define the desired behavior. Use the command `\iftoggle{ABC}{xyz}{uvw}`; if `ABC` is toggled true, then LaTeX will replace the command with `xyz` when rendering; if not, then the command will be replaced with `uvw` instead.

7.3 Sizing

We mentioned some ways to change font styles earlier (Section 5.4). But we didn't mention one additional alternative LaTeX offers – size! You can change the font size of a selected portion of text with `{\size ...}`, for the following sizes (case-sensitive):

- tiny
- scriptsize
- footnotesize
- small
- normalsize
- large
- Large
- LARGE
- huge
- Huge

While changing text sizes might be fun, LaTeX offers a more useful implementation of sizing for math symbols. For an example of why we may need this, consider the formula for PIE:

Input:

```
\[
| \bigcup_{i=1}^n S_i | = \sum_{k=1}^n (-1)^{k-1}
\sum_{J \subseteq [1..n], |J| = k } | \bigcap_{j \in J} S_j |
\]
```

Output:

$$| \bigcup_{i=1}^n S_i | = \sum_{k=1}^n (-1)^{k-1} \sum_{J \subseteq [1..n], |J| = k } | \bigcap_{j \in J} S_j |$$

The cardinality bars are a little small. To remedy this, we can use the following options for math symbol sizes, in the form `\size(` to resize the (symbol:

- `\big(` – $\big($
- `\Big(` – $\Big($
- `\bigg(` – $\bigg($
- `\Bigg(` – $\Bigg($

These sizing options are only available for certain symbols; the commonly used ones include, but are not limited to: `(,),|,[,],\{,\},\lfloor,\rfloor,\lceil,\rceil`. While these options are useful, you can actually let LaTeX take care of the guesswork for you – if you have corresponding symbols around an expression, you can use `\left` and `\right` instead of explicitly identifying sizes. Our earlier example then becomes:

Input:

```
\[
\left| \bigcup_{i=1}^n S_i \right| = \sum_{k=1}^n (-1)^{k-1}
\sum_{J \subseteq [1..n], |J| = k } \left| \bigcap_{j \in J}
S_j \right|
\]
```

Output:

$$\left|\bigcup_{i=1}^n S_i\right| = \sum_{k=1}^n (-1)^{k-1} \sum_{J\subseteq[1..n],|J|=k} \left|\bigcap_{j\in J} S_j\right|$$

One note here: if you have a lot of nested parenthesis, you may want to manually size them, since `\left` and `\right` won't change sizes for nesting. Usually, to use these, you will have to match them up. Sometimes, you will only have one symbol (say, a large brace you want to indicate a series of equations). In this case, you can use `.` (such as `\left.`) to fill in for the "missing" symbol and let LaTeX deal with the sizing (the `aligned` environment just makes a mini-align within mathmode).

Input:

```
\[
    \left.
    \begin{aligned}
    x &= 1, y = 3 \\
    y &= 2, y = 1
    \end{aligned} \right\} \text{Candidate solutions}
\]
```

Output:

$$\left.\begin{aligned} x &= 1, y = 3\\ y &= 2, y = 1\end{aligned}\right\}\text{Candidate solutions}$$

7.4 Spacing

As we may have mentioned before, LaTeX gives you a lot of flexibility in laying out your document. One key to this end is a series of commands that allow you to adjust spacing however you like. The most specific of these are the `\hspace{dist}` and `\vspace{dist}` which add in horizontal and vertical spacing, respectively, of the provided distance. Distance can be provided in various units (most commonly: in, cm, em (roughly the size of an "M", pt). We will commonly use the line break (`\\`), especially in `align` or similar multiple-line environments. You can use `\newline` or `\newpage`, which do about what you'd expect them to. It's also possible to customize the distance for a line break, by using `\\[dist]` – this is especially useful for helping space out tables with complicated expressions.

There's also a couple spacing commands (both in mathmode and text) for tweaking to get nice cushions, in increasing order: `\! \, \: \; \ \quad \qquad`. Note that `\!` is actually *negative* spacing – less than not writing anything! Using these in, for example

```
\{x^2 \ + \ 1 \ | \ x \ \in \ \mathbb{N} \ \}
```

will give the following results, in increasing order:

- $\{x^2\!+\!1\!|\!x\!\in\!\mathbb{N}\!\}$
- $\{x^2 + 1 | x \in \mathbb{N}\}$ (default spacing)
- $\{x^2\,+\,1\,|\,x\,\in\,\mathbb{N}\,\}$
- $\{x^2\:+\:1\:|\:x\:\in\:\mathbb{N}\:\}$

- $\{x^2 \, + \, 1 \, | \, x \, \in \, \mathbb{N} \, \}$
- $\{x^2 \; + \; 1 \; | \; x \; \in \; \mathbb{N} \; \}$
- $\{x^2 \quad + \quad 1 \quad | \quad x \quad \in \quad \mathbb{N} \quad \}$
- $\{x^2 \qquad + \qquad 1 \qquad | \qquad x \qquad \in \qquad \mathbb{N} \qquad \}$

Finally, the ~ symbol adds a non-breaking space. LaTeX usually justifies the text; if you throw in one of these bad boys, then the resulting space will never be broken by a newline (this can also be useful for forcing a little distance between things which are too close together).

7.5 Summation/Product Notation

Summations pop up a lot in 592 material – it'd be hard to count without them! The general format for writing summations is (note the mathmode) is:

Input:

```
\[ \sum_{k=1}^n k^2 \]
```

Output:

$$\sum_{k=1}^n k^2$$

Depending upon the intent, you may prefer to use inline summations $\sum_{k=1}^n k^2$ or display style summations $\sum\limits_{k=1}^n k^2$ (use `\sum\limits_{k=1}^n` or `\displaystyle\sum_{k=1}^n` instead). Inline style can be useful to keep spacing consistent, but can sometimes be more difficult to read. If you are in display mode already (that is, using `\[ ... \]` or similar), then you will automatically be in display style. Everything above also applies for the $\prod$ product notation, except with `\prod` in place of `\sum` and similarly for `\bigcup` ($\bigcup$) and `\bigcap` ($\bigcap$).

It might be the case that the conditions on the sum are kind of bulky and stretch things out a bit. For example, recall the formula for PIE:

Input:

```
\[
\left|\bigcup_{i=1}^n S_i\right| = \sum_{k=1}^n (-1)^{k-1}\sum_{
J \subseteq [1..n], |J| = k } \left| \bigcap_{j \in J} S_j \right|
\]
```

Output:

$$\left|\bigcup_{i=1}^n S_i\right| = \sum_{k=1}^n (-1)^{k-1}\sum_{J \subseteq [1..n], |J| = k } \left| \bigcap_{j \in J} S_j \right|$$

Observe that we're using the sizing commands from above. Note the decently large gap made by the big summation condition. We can tell LaTeX to stack the relevant conditions as follows:

Input:

```
\[
\left|\bigcup_{i=1}^n S_i\right| = \sum_{k=1}^n (-1)^{k-1} \sum_{
\substack{ J \subseteq [1..n] \\ |J| = k }} \left| \bigcap_{j \in J}
S_j \right|
\]
```

Output:

$$\left|\bigcup_{i=1}^n S_i\right| = \sum_{k=1}^n (-1)^{k-1} \sum_{\substack{ J \subseteq [1..n] \\ |J| = k }} \left| \bigcap_{j \in J} S_j \right|$$

There's now less empty space within the expression. But we can do better! Observe that there's still a bit of a gap on the left of the second summation. To take care of this, we use `\mathclap{}` from the `mathtools` package.

Input:

```
\[
\left|\bigcup_{i=1}^n S_i\right| = \sum_{k=1}^n (-1)^{k-1} \sum_{
\mathclap{ \substack{J \subseteq [1..n] \\ |J| = k } } } \left|
\bigcap_{j \in J} S_j \right|
\]
```

Output:

$$\left|\bigcup_{i=1}^n S_i\right| = \sum_{k=1}^n (-1)^{k-1} \sum_{\mathclap{\substack{J \subseteq [1..n] \\ |J| = k}}} \left|\bigcap_{j \in J} S_j \right|$$

The spacing on both sides of the sum is reduced. But this gives us some overlap between the conditions on the right pair! To take care of this, we manually insert spacing (see 7.4).

Input:

```
\[
\left|\bigcup_{i=1}^n S_i\right| = \sum_{k=1}^n (-1)^{k-1} \sum_{
\mathclap{\substack{J \subseteq [1..n] \\ |J| = k }}} ~ \, \left|
\bigcap_{j \in J} S_j \right|
\]
```

Output:

$$\left|\bigcup_{i=1}^n S_i\right| = \sum_{k=1}^n (-1)^{k-1} \sum_{\mathclap{\substack{J \subseteq [1..n] \\ |J| = k}}} ~ \, \left|\bigcap_{j \in J} S_j \right|$$

So these give some minor cosmetic improvements (again, these aren't strictly necessary to know). Note that, since we use `\mathclap`, the labels are ***c***entered under the summation - there's also similar commands `\mathllap` and `\mathrlap` for ***l***eft and ***r***ight alignment.

8 Final Remarks

This guide is intended to be a (somewhat) brief overview to the LaTeX you will need to succeed in CIT 592. Of course, this barely scratches the surface of what is possible with this powerful tool. There's a lot in here you probably won't need, and it might be missing something you do – if this is the case, feel free to ask a TA or on the discussion forum about any questions you might have. And don't be afraid to explore; there's a lot more to this wonderful world we haven't touched upon here.