UNSW Mathematics Society

An Introduction to LaTeX

Version 1.0

Compiled by Rui Tong

Contents

Preface

TEX is a typesetting program that has ultimately been optimised for many specialised disciplines. Mathematics is one of which it is used for. LATEX builds a lot of commands on top of TEX, as otherwise the formatting would take ages to code up. These commands basically made our lives a lot more easier, and is why many people now use it for any purpose that involves mathematics. This also includes other disciplines of science, finance and actuarial analysis, engineering and much more.

That is one of the best things about LATEX. You might be learning it for the sake of typesetting some mathematics, but the skill can be carried with you to other causes!

This guide has been compiled with the aim of providing you a debatably basic, but hopefully adequate understanding of what LATEX has to offer. It is slightly targeted towards students currently taking a first or second year mathematics course, however it should contain enough detail to last you a while, potentially even up to honours. Despite being mildly lengthy, it is still intended to be used for introductory purposes. By no means does this guide serve as an end to everything - there is still a lot more about LATEX left for you to discover. (In fact, even for us to discover!)

Inevitably, there may be some errors scattered across this guide. If you find any, please let us know at our email unswmathsoc@gmail.com or by sending a message to our Facebook page.

We hope this guide will serve reasonable benefit for you and perhaps even give you a strong appreciation for what LATEX can do. All the best in your studies, and who knows, maybe you will use LATEX in your future endeavours!

Chapter 1

Commencing LaTeX

You essentially have two options in using LaTeX. You can either set everything up on your personal computer, or render everything online. Once you do all of this once, it's likely you never have to visit this part again.

1.1 Local or online?

A nice explanation was provided here (click me) in regards to which option you recommend. Otherwise personally, I consider this debate similar to whether you want Google Drive or an online version of Onedrive as opposed to a Microsoft Word document on your device.

- If you intend to allow collaborators onto your file, use an online platform.
- If you want to have your files automatically backed up, use an online platform.
- If you intend to have lots of extremely huge files that take up too much memory otherwise, use your personal computer.
- If your files are extremely large, or cluttered with images or anything of the sort, probably use your personal computer. Some things skyrocket the compilation time on online platforms.
- If you plan to use some very niche packages in the future, and are not bothered to download them, an online platform may be nice as they will handle that for you. But it's unlikely you'll be able to rely on them forever due to issues with memory, unless you purchase premium plans.

1.2 Setting up for Windows

There's quite some extensive debate between which option to take on Windows. Perhaps, as far as you'd likely care:

- MiKTeX does a good job at giving you all the basics, and offers a how-to-install section for you to follow.
- As far as I'm aware of, proTeXt provides added functionality *on top of* MikTeX. Note that the *TeXstudio* text editor is automatically downloaded with this choice.

- TEXLive is, from what I understand, the best alternative to MiKTEX

Personally, I've always downloaded MikTEX, because I too was oblivious and just wanted simplicity. I have not had problems with it, but you may opt for a different option. Here is an article I found on the differences between MikTEXand TEXLive if you're really interested.

Once you download and install your LATEX software, you will require a text editor to type it in, and likely also compile it for you. You can open the text editor and type all of your LATEX code in there. You can also open and save files, much like you would do with Microsoft Word. TEXfiles are saved with the `.tex` extension.

Recommended options are TEXmaker, TEXstudio and TEXworks. Note again that if you downloaded proTEXt, TEXstudio was already downloaded for you. Personally I used TEXmaker, as this was recommended by the 2015 director Brendan Trinh.

1.3 Setting up for Mac

The LATEX group recommends MacTEX for Macintosh users. The website may be accessed here. If you plan to use LATEX not too often, the BasicTEX download is sufficient. Otherwise if you're willing to use approximately 7 GB of memory, there's no other harm in downloading the full option. (The full option saves you the hassle of downloading tons of packages later in the long run.)

- MacTEX can be downloaded here using the bolded link that says "MacTeX.pkg". The installation process is essentially the same as for other Mac downloads.
- BasicTEXcan be downloaded here where it says BasicTeX.pkg.

A text editor will also be required for Mac for typing and compilation purposes. TEXShop is downloaded by default when you install MacTEX. Alternatively TEXmaker, TEXstudio and TEXworks are still great options. If you are very proficient with terminal text editors like vim and Emacs you may opt to use them as well. You can compile LATEX files using `pdflatex` (which you may need to install via home brew, but I'm not sure). The syntax is just `pdflatex file.tex`.

1.4 Using an online LATEX editor

The only two options I know of are LATEXBase and Overleaf. Personally, I'm used to Overleaf as it offers a variety of templates for fancier uses of LATEX, and it offers UNSW students a free professional account if they register/link a UNSW email to it.

On online platforms, the packages are all handled for you, so you don't need to worry about it. Again, note that it will likely be slower for extremely large files. But this won't matter for all of you.

On Overleaf, all of your files will be displayed as a list (which you can also do on Google Drive). GitHub integration is offered if you're interested. You can also specify tags for each of your files to make searching easier, if you have lots. When you open an Overleaf file, you'll also have a **Menu** in the top left corner. You can use this to download a copy of your `.tex` file, or change some of the settings in Overleaf.

1.5 Structure of a typical LaTeX document

A typical `.tex` file will be of the following format:

```
\documentclass[possible parameters]{class}

**--PREAMBLE--**

\begin{document}

**--MAIN BODY OF TEXT--**

\end{document}
```

- The **document class** specifies essentially what document you're after. A full list of them may be found here. The following are some common ones.
  - `article`: This is basically the default class. It is used for your standard, reasonably simplistic documents. Roughly 80% of my LaTeX usage altogether involves this class.
  - `report`: This class, in some sense, adds extra features on the `article` class to build a report. It is recommended for significantly longer files, where having chapters and parts may be useful.
  - `book`: This class is very simliar to the `report` class. However it involves some optimisations for books, for example by default the pages are two-sided.
  - `exam`: This is a pre-built class for writing papers.
  - `letter`: This class is optimised specifically for mail.
  - `beamer`: This is one of many classes used for creating presentations using LaTeX. We cover it briefly later.

  For the interested, here is some discussion on the differences between the first three classes.

- Some parameters you can include in the document class are available here. In practice, I've only found the following two important.
  - Default font size - `10pt` (default), `11pt` or `12pt`. Resizing comes later.
  - Page size - `letterpaper`, `a4paper` (default), `a5paper` and some more.

  Parameters are specified one after the other, with commas separating them. For example, because I typically use A4 paper and 12pt sized font, I tend to write `\documentclass[12pt, a4paper]{article}`. To my knowledge, the exact ordering of the parameters does not matter.

- The **preamble** consists of all of the packages, commands and any other functionality you wish to include. By default, packages consist of various inbuilt commands people use to customise their output file. Packages are loaded using the syntax `\usepackage{nameofpackage}`.

Note that loading too many packages may slow down compilation time considerably. You should only load packages when you need them! If you like having a list of packages ready to be included, just comment out any that you don't need!

Some packages are introduced throughout this guide, along with explanations on a few of the things they do. Full descriptions on what packages do may be found in their documentation. Be warned though - many of those are extremely long.

- **Commands** are basically what you use to access the features in LaTeX! Every command is preceeded by the backslash symbol: `\command`. Some commands may require you to provide arguments, for example bold-font requires you to provide the text you wish to bold as an argument: `\textbf{bolded text}`. Each command is different, and the number of arguments required depends on which command you're currently using.

- LaTeX has a special feature known as **environments**. Environments typically take the form:

  ```
  \begin{environmentname}[optional parameters]{mandatory parameters}
  **--text--**
  \end{environmentname}
  ```

  Basically, environments have the begin/end structure, which distinguishes them syntax wise from commands. Note that some environments have no required parameters either.
  For the interested, the subtle distinction between commands in environments can be found here.

Once you finish coding your `.tex` file, you compile it to produce the output file. For our purposes, compiling is where the computer arranges everything typed in your file, adds appropriate design, and gives you what you require. Usually I recommend using **pdfLaTeX** for this, which just so happens to give your output file as a PDF.

On your personal device, LaTeX text editors are usually designed to give you a compilation button to click on. On Overleaf, the compilation button is basicallly above your PDF's preview. You should be compiling reasonably often, as you may introduce bugs (errors) in your code that would otherwise be undetected for a while. Debugging LaTeX - a typesetting software may be easier than debugging a program, but it's still draining nonetheless!

Usually there are also hotkeys for compilation. On Overleaf, you can use Ctrl+Enter.

Chapter 2

Generic LaTeX features

As much as LaTeX was optimised for mathematics and other specialised tasks, it'd be a bit redundant and extra if it couldn't do many basic features you'd easily find in Microsoft Word! This section demonstrates how various little features in Microsoft Word can be used in LaTeX as well.

Everything typed in LaTeX is done after the preamble section, within the `document` environment.

By default you are placed into text mode,

2.1 Basic typesetting

2.1.1 Basic text

Pretty much any text you type will literally show up as is, after you type the document. You do not have to do anything fancy if you want nothing more than a string of characters.

- A worthwhile remark to make though: On Overleaf, `\usepackage[utf8]{inputenc}` is automatically included in your preamble. You *may or may not* want it there, but it has the benefit of allowing you to include Unicode characters. If you don't intend to use many special characters, you can probably get away without it.
- LaTeX also automatically ignores it when you type space twice. You can try typing a double space (or even lots of spaces) between two words, yet you'll only see one word. This is fair enough, because you shouldn't need double spaces.

Every time you want to start a new paragraph, introduce a new equation, start a new environment or anything of the sort, use a new line in your code. This will help you navigate your code better.

2.1.2 Special characters

The reason why I say *pretty much* is because there are some symbols that you cannot type ordinarily. You can find a list of them here, however here are some standouts.

- $ - dollar sign, must be typed as `\$`.

- % - percentage sign, must be typed as `\%`.
- { } - braces (curly brackets), must be typed as `\{` and `\}`.
- & - ampersand, must be typed as `\&`.
- ~ - tilde, can be typed as `\~{}`.
- \ - backslash, must be typed as `\backslash`, or `\backslash{}` if you require a space after it. This is because the backslash symbol is actually reserved for commencing commands/environments, which you may or may not have realised!

2.1.3 Line breaking

For reasons I personally don't understand, LaTeX generally discourages the use of unnecessary line breaking. In fact, clearly I am a rebel in that I abuse it.

- Line breaking is done using `\\` (two backslashes).
- Line breaking does not skip a line; it just moves you to the next line. One way to skip a line is to use `\\ \\` (include the space). I should warn you that this is cheating, but I do it anyway.

2.1.4 Bold, italics and underlines

There are commands that allow you to do this in text mode.

- `\textbf{1}` is used to produce bolded text. The bolded text is in the curly braces.
- `\textit{1}` is used to produce italicised text in a similar way.
- `\underline{1}` is used to produce underlined text in a similar way.

For example, `\textit{MathSoc is great}` produces *MathSoc is great*.

To have text with multiple of the above formats we can nest the commands in any order. (That means, we use one command inside the other.) For example, this code gives the output below it.

```
\textbf{\textit{Hello}} \underline{\textbf{world!}}
```

Hello **world!**.

A note on strikethrough - if you plan to use strike outs in your text, you may want to load the `ulem` package. Then, `\sout{1}` can be used for striking through text.

2.1.5 Colouring

If you wish to use colours, you should add one of the following commands to your preamble.

- `\usepackage{color}` - this gives you the basic colour set. These are white, black, red, green, blue, cyan, magenta and yellow.

- `\usepackage{xcolor}` - this gives you some added functionality with colours.
- `\usepackage[dvipsnames]{xcolor}` - the optional parameter `dvipsnames` adds some more default built-in colours. Here is a full set of the colours:

Click for image source

Colours are applied by using `\color{1}`, where {1} contains the name of the colour you require.

Once a colour is applied, it will continue to be applied until it is 'stopped'. (Try it - you'll find basically an entire document gets coloured!) To ensure you don't colour more than what you wanted to, I initially recommend wrapping them around in braces. For example, this code gives the output below it.

```
Uncoloured stuff {\color{blue}blue coloured stuff} more uncoloured stuff
```

Uncoloured stuff blue coloured stuff more uncoloured stuff.

Whereas, for another example:

```
Some uncoloured stuff \color{magenta}some magenta stuff\\
Some stuff I was hoping to not be coloured on the next line...
```

Some uncoloured stuff some magenta stuff
Some stuff I was hoping to not be coloured on the next line...

For the experienced user, the braces effectively group some text into a chunk. When a colour is applied, the colour persists until this 'chunk' ends. However, when a colour is used in an environment, the colour won't flood out of the environment. If you want, say

a coloured `itemize` environment, you can just apply the colour after `\begin{itemize}` and it'll stop once your bullet points end!

2.1.6 Changing font size

The commands for resizing text work essentially the same way as for colouring. You have the following options available.

- `\tiny` - produces text of this size.
- `\scriptsize` - produces text of this size.
- `\footnotesize` - produces text of this size.
- `\small` - produces text of this size.
- `\normalsize` - produces text of this size. (Default size.)
- `\large` - produces text of this size.
- `\Large` - produces text of this size.
- `\LARGE` - produces text of this size.
- `\huge` - produces text of this size.
- `\Huge` - produces text of this size.

Furthermore, when declaring the document class, you may have noticed that only 10pt, 11pt and 12pt sizes are available. It is worth noting that if you want a different *default* size, you may load the `extarticle` or `extreport`. They have the same functionality, but also allow the sizes 8pt, 9pt, 10pt, 11pt, 12pt, 14pt, 17pt and 20pt.

You should always think about why you're resizing the text when you do! For certain things like titles this may be justified, but usually you should stick to the same size wherever possible!

2.2 Sections

The people that designed LaTeX know that sometimes splitting a document into sections helps readability. In the `article` document class, LaTeX offers you the following.

- `\section{1}` allows you to create the overarching sections. In articles, sections are the highest you can go in the hierarchy. The name of your section goes in the place of {1}.
- `\subsection{1}` essentially groups a smaller bunch of text in your sections into, amazingly, a subsection! They are made the same way,

- `\subsubsection{1}` allows you to go one step further with the above. It's the furthest you can go though - there is no subsubsubsection.
- `\paragraph{1}` allows you to split paragraphs from each other. Anything placed inside the {1} is treated as a paragraph title and shows in bold. Personally I have not had a need for it, but you may!
- `\subparagraph{1}` adds to paragraphs in a similar way to subsections.

If you're using the `report` or `book` document class, there are two additional commands you can use.

- `\chapter{1}` is higher than sections in the hierarchy. Each chapter has its own sections. When you start a new chapter, the counter for the sections does not carry over - it resets back to 1.
- `\part{1}` is higher than chapters in the hierarchy. A part is a means of grouping multiple chapters together. They're not used in this guide, but note that each time you start a new part, the chapter counter does **not** reset.

For each document class, you may also use `\tableofcontents` at the start of the file. This will generate the table of contents for you - you don't have to do it manually!

I don't provide specific examples of sections in this guide as it would muck with the contents page too much. However, this article has lots of chapters, and you can see each section within each chapter! Note that by default, sections and subsections are labelled with a numbering system. They are used throughout this entire guide.

There is a way to remove the numbering system for your sections - we basically put an asterisk after the command, before the braces that give it a name. For example, `\section*{My Section}` will give a section named 'My Section', but without a section number assigned. `\chapter*{My Chapter}` will give a chapter named 'My Chapter' but again wit no number assigned.

2.3 Bullet points and numbered lists

There are environments built into LaTeX that generate lists of points for you. For dot points, we use the `itemize` environment. Each point is specified by an `\item`. For a basic example:

```
\begin{itemize}
    \item Item 1
    \item Item 2
    \item Item 3
\end{itemize}
```

- Item 1
- Item 2

- Item 3

A numbered list is made using the `enumerate` environment. Once again items are used to record contents for each point.

```
\begin{enumerate}
    \item Item 1
    \item Item 2
    \item Item 3
\end{enumerate}
```

1. Item 1
2. Item 2
3. Item 3

2.3.1 Nesting dot points and numbered lists

Sometimes we like to record sub-dot points under each point. LaTeX allows us to handle this by literally creating `itemize` and `enumerate` environments, *inside* environments! As an example

```
\begin{enumerate}
    \item Item 1
    \begin{itemize}
        \item A point in item 1
        \begin{itemize}
            \item A subpoint
        \end{itemize}
        \item Another point in item 2
    \end{itemize}
    \item Item 2
    \item Item 3
    \begin{enumerate}
        \item Subitem 3a
        \item Subitem 3b
    \end{enumerate}
\end{enumerate}
```

1. Item 1
    - A point in item 1
        - A subpoint
    - Another point in item 2
2. Item 2
3. Item 3

(a) Subitem 3a

(b) Subitem 3b

LaTeX allows you to nest up to four levels, which is more than enough in practice. After experimenting, I found that if you try nesting a fifth level, LaTeX throws a 'Too deeply nested' error.

2.3.2 The `paralist` package

Adding `\usepackage{paralist}` offers some customisations to your lists. You can explore the documentation for other factors it contributes.

For the `itemize` environment, it essentially allows you to specify the symbol you wish to use (as opposed to the default bullets, dashes and asterisks). It is given as an optional parameter after the environment is commenced as shown. The below example uses exclamation marks for each point.

```
\begin{itemize}[!]
    \item Fancy point 1
    \item Fancy point 2
    \item Fancy point 3
\end{itemize}
```

! Fancy point 1

! Fancy point 2

! Fancy point 3

In theory, you can type any string of characters in the square brackets, ranging from stars to math symbols to even words! But don't do things like that without good reason to.

For the `enumerate` environment, it allows you to specify what format you wish the numbers to appear in, as well as if you want brackets or dots to go alongside it. The formats available for numbers are:

- `A` - capital letters A, B, C, D, ...
- `a` - lower case letters a, b, c, d, ...
- `1` - Hindu-Arabic numerals 1, 2, 3, 4, ...
- `I` - upper case Roman numerals I, II, III. IV, ...
- `i` - lower case Roman numerals i, ii, iii, iv, ...

In the following example, I use upper case Roman numerals for the outer `enumerate` environment, and then lower case Roman numerals for the inner. I also use a single right bracket for the outer environment, and a dot for the inner environment.

```
\begin{enumerate}[I)]
    \item Item 1
    \item Item 2
    \begin{enumerate}[.]
        \item Subitem 2.1
        \item Subitem 2.2
    \end{enumerate}
    \item Item 3
\end{enumerate}
```

I) Item 1

II) Item 2

   i. Subitem 2.1

   ii. Subitem 2.2

III) Item 3

It is also possible to have a string of characters that *does not change for each dot point* alongside it. However I need to enclose any such text in braces as shown.

```
\begin{enumerate}[\text{Claim }1.]
    \item Here I make my first claim.
    \item Here I make my second claim.
    \item Here I make my third claim.
\end{enumerate}
```

Claim 1. Here I make my first claim.

Claim 2. Here I make my second claim.

Claim 3. Here I make my third claim.

Note: You can probably do a lot of things as well without `paralist`*. This section purely exists because I'm fairly sure Microsoft Word has a similar feature.*

2.4 Text alignment

By default, LaTeX uses justified text alignment in Microsoft Word. This is when for paragraphs that go on multiple lines, the text is aligned so that we have perfectly straight left and right edges throughout the document. Generally speaking, this is desirable in the real world.

You can introduce blocks of texts that are left, centre or right aligned using their corresponding environments.

- The `flushleft` environment is used for left aligned text.

- The `center` environment is used for centred text.
- The `flushright` environment is used for right aligned text.

As a small example involving flushright:

```
\begin{flushright}
This text will show up closer to the right.
\end{flushright}
```

This text will show up closer to the right.

You may have noticed that on Microsoft Word, left alignment may give you jagged edges on the right. This sort of thing will occur for `flushleft` as well. You can try it out!

2.5 Page margins

Page margins are set in the preamble for LaTeX, much like how they're set in the toolbar for Microsoft Word. They are changed using the `geometry` package. If you want to change the page margins, include this in your preamble:

```
\usepackage{geometry}
```

The geometry package is very powerful - you can refer to the documentation (user's manual) to explore further. I focus on the page margins here. Your margins can be changed using inches, millimetres, centimetres or pt (point).

2.5.1 Adjusting margins for the entire document

On the next line, after you include the geometry package, just add `\geometry{margin=length}` to change all 4 margins! (That is, the top, bottom, left and right margins.) For example,

```
\usepackage{geometry}
\geometry{margin = 1in}
```

will set all the margins to 1 inch.

There is an alternate syntax that will let you do this - this syntax adds the margin width as an optional parameter to the geometry package. Please only include this in your preamble or the syntax above, not both!

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

2.5.2 Adjusting each margin one at a time

If you do not want consistent margins in your document, you can change each of the left, right, top and bottom margins individually. The following example sets the top and bottom margins to 260 mm, and the left and right margins to 0.6 inches.

```
\usepackage{geometry}
\geometry{
    top = 260mm,
    bottom = 260mm,
    left = 0.6in,
    right = 0.6in
}
```

Yes, I believe the names `top`, `bottom`, `left` and `right` were very conveniently chosen.

Note that the order you type them in does not matter. Note also that you can omit some of them. For example if you only want to adjust the left and right margins, just delete the lines involving `top` and `bottom`.

2.5.3 Landscape oriented document

As an aside, the geometry package also give you the option of making the paper landscape oriented. This is done in a similar way to the above. Just add `\geometry{landscape}` as well after you load the geometry package!

2.6 Multiple columns

You have two options here.

2.6.1 Automatically rendering columns

You can tell Microsoft Word to just group a bunch of text into two or three nicely displayed columns. (Or more, if you had good reason to.) The procedure to doing this is reasonably straightforward, but you need to load the `multicol` package first. I.e. add this to your preamble.

```
\usepackage{multicol}
```

The syntax for the multicols environment is:

```
\begin{multicols}{no. of columns}
    ...a lot of text goes here...
\end{multicols}
```

We don't show an example of it here, but this is reasonably straightforward to experiment with!

2.6.2 Manually making columns

The nice thing about the above procedure is that the columns will always end at the same level. They're balanced - no column runs further down the page than the other. They're also designed so that they each have the same width.

But I have bumped into scenarios where I prefer the column widths to vary, most notably when trying to attach a picture next to some text. At times like this, I want to manually tell LaTeX what to do. The `minipage` environment is one way of working around it.

The name `minipage` really comes from making a 'smaller' page in a page. In a minipage, the page width needs to be specified. Trying to write specific values for minipage widths can be a bit hard. Fortunately, LaTeX has a command `\textwidth` that automatically finds the width that your text is allowed to stretch for.

When making minipages, we can essentially make their widths a scale times the `\textwidth` as shown below. Unless you have good reason to, your scales should sum to 1. Here, I have a minipage that's 0.7 times the text width, and another that is 0.3 times the text width. (The `minipage` environment syntax is similar to the `multicols` syntax, except the number of columns is replaced by the width.)

```
\begin{minipage}{0.7\textwidth}
    \color{blue}
    This is a random block of text. This is a random block of text.
    This is a random block of text. This is a random block of text.
    This is a random block of text. This is a random block of text.
    This is a random block of text. This is a random block of text.\\
\end{minipage}
\begin{minipage}{0.3\textwidth}
    \color{magenta}
\begin{flushright}
    Some more text on the other minipage, right aligned.\\
\end{flushright}
\end{minipage}
Some text outside the minipages.
```

This is a random block of text. This is a random block of text. This is a random block of text. This is a random block of text. This is a random block of text. This is a random block of text. This is a random block of text. This is a random block of text.

Some more text on the other minipage, right aligned.

Some text outside the minipages.

Note: `minipage` *is really a generic environment designed to just stick things next to each other. It is a hard environment to use, and technically speaking serves a different purpose to* `multicols`. `multicols` *focuses on allowing a whole block of text to go on columns, whereas* `minipage` *is more suited for putting arbitrary things side by side.*

You may want to avoid the `minipage` environment until you really need it.

2.7 Tables

The `tabular` environment is used to construct tables. This environment is a bit more intricate than all previous ones.

- A mandatory parameter to give the `tabular` environment is the format for the *columns*. We'll see examples of these, but the only ones I imagine you'd need are the following.
  - `l` - left aligned text in the table
  - `c` - centre aligned text in the table
  - `r` - right aligned text in the table
  - `p{width}` - creates a column for a paragraph with a specified width like minipage
  - `|` (absolute value bar) - column separator
  - `||` (two absolute value bars) - double column separator
  - space - a new column, but not separated by a vertical line
- Rows are separated by the line break character
  . In environments,
  is not regarded as bad practice.
- In each row, to separate the columns an ampersand `&` is used.
- After each line break, `\hline` can be used to create a horizontal row separator. In a similar way, `\hline\hline` creates a double horizontal row sepraator.

The following table has 4 rows and 4 columns. It uses `\hline` after every line break to separate the rows, and `|`'s in the format to separate each column.

```
\begin{center}
    \begin{tabular}{c|c|c|c}
        Entry 1 & Entry 2 & Entry 3 & Entry 4 \\
        \hline
        Entry 5 & Entry 6 & Entry 7 & Entry 8 \\
        \hline
        Entry 9 & Entry 10 & Entry 11 & Entry 12 \\
        \hline
        Entry 13 & Entry 14 & Entry 15 & Entry 16
    \end{tabular}
\end{center}
```

Entry 1	Entry 2	Entry 3	Entry 4
Entry 5	Entry 6	Entry 7	Entry 8
Entry 9	Entry 10	Entry 11	Entry 12
Entry 13	Entry 14	Entry 15	Entry 16

The following table has 3 rows and 5 columns. To tweak things around a little:

- There are fewer vertical and horizontal lines that seperate the rows and columns, as we replace `|` with an empty space and use less `\hline`'s.

- The table has an outside border, made by `|`'s in the column format, and `\hline`'s at the start and end.
- The first column is right aligned, not centre aligned.
- This table also has double separators, made by `\hline\hline` and `||`, after the first row and first column.

```
\begin{center}
    \begin{tabular}{|r||c c c|c|}
        \hline
        Name & Test 1 mark & Test 2 mark & Test 3 mark & Assignment mark \\
        \hline\hline
        Austin & 45 & 90 & 91 & 100 \\
        Chloe & 76 & 85 & 68 & 100 \\
        \hline
    \end{tabular}
\end{center}
```

Name	Test 1 mark	Test 2 mark	Test 3 mark	Assignment mark
Austin	45	90	91	100
Chloe	76	85	68	100

The world of tables in LaTeX certainly does not end here. Very fancy tables can be made by including other packages in your preamble. As I am not experienced with them, I do not cover them here. I link to Overleaf's LaTeX guide for tables for those interested in making these.

2.8 Attaching images

The `graphicx` package must be loaded to allow images to be inserted into LaTeX. Ensure that this is in your preamble.

```
\usepackage{graphicx}
```

Note that there is also a package called `graphics`. But it is somewhat obsolete compared to `graphicx`.

The next step depends on if you're working on your own device, or via Overleaf.

- On your own device, ensure you have your `.tex` file saved in a folder. Then, put your image in the *same* folder as where your `.tex` file is. (Can be `.jpg` or `.png`.)
- On Overleaf, under the Menu at the top, there is a button that allows you to upload attachments. You will need to upload your image first.

The syntax for including images is `\includegraphics{image name}`.

In practice, however, my images tend to be too large. You may not have this issue, but I know I do. There is an optional parameter called `scale` that lets you resize the image to a smaller scale. The smaller your value of `scale` is, the smaller the image will be.

Here, I've uploaded a file called `mathsoc_logo.png` behind the scenes. The code

```
\begin{center}
    \includegraphics[scale=0.5]{mathsoc_logo.png}
\end{center}
```

gives me this. Clearly my image is way too large - it's going off the page!

I then change my scale to 0.1 and obtain something smaller.

```
\begin{center}
    \includegraphics[scale=0.1]{mathsoc_logo.png}
\end{center}
```

Worthwhile mention: On the front cover, I used `\includegraphics[scale=0.2]{mathsoc_logo.png}`.

2.9 Linking URLs

The `hyperref` package must be loaded to allow hyperlinking, and many other forms of referencing. Ensure to include this in your preamble.

```
\usepackage{hyperref}
```

In practice, much like in this file, I imagine you'd want your hyperlinks to be coloured. This is a setting we need to turn on, so we include this immediately afterwrds in the preamble.

```
\hypersetup{
colorlinks=true,
urlcolor=blue, % this colour can be changed obviously
}
```

There are two valid approaches for linking URLs.

- If you just want to link the ordinarily, you can just use `\url{url address}`.
- In practice, usually you want to display some meaningful text to your reader, instead of a long-winded URL address. To do this, you can use `\href{url address}{text to display}`. Here is a quick example:

```
\href{https://mathsoc.unsw.edu.au/resources/latex-guide/}{Click me!!!}
```

Click me!!!

Note: As mentioned, `hyperref` allows various other forms of linking. One major benefit is that it allows you to click on sections in the table of contents, and jump straight to that section! For more advanced uses, check out Overleaf's guide here.

2.10 Comments (in coding)

In programming, commenting is a common practice to annotate your code so that others may understand it more easily. It has an added benefit in that it can be used to temporarily eliminate a few lines of code, and can be brought back later without rewriting the entire thing from scratch.

Comments in LaTeX are achieved by using the percentage symbol `%`. How it works is that LaTeX will ignore everything after the percentage symbol, on the same line. It will then jump straight to the next line.

For example:

```
\color{magenta}
This text will show up \\ % This text will not show up
This text will also show up % This text will also not show up
```

This text will show up
This text will also show up

Comments can be used however you feel appropriate. I personally use comments when debugging LaTeX, and then remove each comment (by deleting the `%`) line by line, recompiling over and over again to figure out what's going on. Sometimes I also use it to keep

track of a few nitty gritty details along the way.

If you're one of those people that use LaTeX templates like me, you may have a preamble already set up, but it loads way more packages than what you require. You can add a percentage symbol in front of each `\usepackage{}` to turn that package off, and thus reduce compilation time a bit.

A note to programmers - sadly, LaTeX does not have any *proper* way of block commenting! Here's my advice.

- Try to avoid using it altogether. Try to save commenting for just 5 or so lines that you can comment manually.
- You can also dump code after the end of the document, i.e. `\end{document}`. LaTeX behaves nicely in that it won't read it.
- If you insist on block commenting, you can Google block commenting in LaTeX for some dodgy ideas on TeXStackExchange. They're all an abuse of the system, but they work!

2.11 Searching and replacing

This is actually no stranger to Microsoft Word users. Just use Ctrl+F! On Overleaf, this is also used for replacing, but this may vary for text editors on your own device.

2.12 A recommended package: `microtype`

It is generally recommended that you always add this to your preamble.

```
\usepackage{microtype}
```

Understanding the `microtype` package is extremely difficult, but fortunately just doing that one line of code is usually enough. The package basically improves the spacing of your letters significantly. You can investigate the documentation more if you really wish to, or Google search about it, but I recommend just leaving it as is!

2.13 Some more advanced functions

These are some things I feel experienced LaTeX users may wish to explore. I would not advise new LaTeX users to worry about this.

2.13.1 Macros: `newcommand`

We all know this - some commands take too long to type. In fact, once you start typing maths, sometimes you may need to type the same nastily-long formula 50 times and it gets exhausting! Simplifying this into a `newcommand` structure may make your life easier.

For commands that do not require any arguments, the syntax is

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

For example, in statistics, I may get tired of typing the density function of the normal distribution over and over again. I can then add, say,

```
\newcommand{\gaussian}{\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{(x-\mu)^2}
```

I then run

```
\[ f_{X} (x) = \gaussian. \]
```

and obtain

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

For commands that do require arguments, the syntax is

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

In the definition of the command, we use `#num` to indicate where each argument gets substituted in for. `#1` will substitute in for the first argument, `#2` for the second, `#3` for the third , and so on until we reach the number of arguments we have. For example, the following macro involves two arguments.

```
\newcommand{\BI}[2]{\textbf{#1} {\textit{#2}}}
```

When running the command, I place each argument in its individual set of braces. Example:

```
\color{magenta}
\BI{some text to come in bold}{some text to come in italics}
```

some text to come in bold *some text to come in italics*

Note that you can also make new environments, but I have avoided doing this thus far. Note also there is a `\renewcommand{}{}` analog, when you're trying to declare a new macro, but the command already exists. Be very cautious using this though - you should understand the risks of what you're doing!

2.13.2 An equivalent to header files for the preamble

Sometimes my preamble goes for up to 40 or even 75 lines and I need to scroll down before I can even find where my document starts! Fortunately, the preamble can be stored away.

On Overleaf, under the menu, you can add a *new* file. I usually create a file named `preamble.tex`. On your local device, just create this file in the same folder your main TEX file is in. (In this sense, it is somewhat similar to inserting images.)

Then, in my main document, I start with the following:

```
\documentclass[]{article} % or whatever you want it to be
\include{preamble}
\begin{document}
    ...
```

After I do this, I just dump my preamble into `preamble.tex`! Much tidier in my opinion.

Note: I've found that doing `\include{preamble.tex}` instead does not work for me, despite the fact that Overleaf's autocomplete makes me do that.

2.13.3 Manually creating spaces

There are two commands that provide you spaces for a given width automatically.

- `\vspace{width}` gives you a vertical space with the specified width.
- `\hspace{width}` gives you a horizontal space with the specified width.

There are situations where these are justified, but in general you should have good reason for using these commands. (For example, when formatting documents for specialised purposes.) For generic articles, I try to avoid these as much as possible.

Chapter 3

LaTeX features for mathematics

If it weren't for the fact it makes typesetting mathematics so much easier, many of us would probably not bother with LaTeX. This section is dedicated to the mathematical syntax behind LaTeX. In practice, you usually use whatever syntax you need, when you need it.

You must be in a math mode to use all the syntax reserved for mathematics.

3.1 Preparing to use mathematics

3.1.1 Packages

Without focusing too much on detail, the packages that I always recommend you use for math are:

- `amsmath` - This just improves the structure of mathematics in LaTeX files significantly.
- `mathtools` - Offers some fixes to `amsmath` as well as adding some more symbols and environments.
- `amssymb` - Offers more symbols that are usually handy for use.

I recommend always having the following two lines in your `.tex` file for documents involving mathematics. Note that `amsmath` is missing - the way `mathtools` is designed is that it automatically loads the former package for you. But you can also load `amsmath` first if you wish to.

```
\usepackage{mathtools}
\usepackage{amssymb}
```

Many users also include the `amsthm` package. This offers a proper way of declaring definitions, lemmas, theorems, corollaries, proofs and so on in LaTeX. We don't include this here as I don't feel it is necessary until honours level study.

3.1.2 Two math modes

Mathematics can be rendered in the *inline* mode or the *display* mode.

- Inline math is when you require that the mathematics shows up in a paragraph, i.e. among a lot of text. The code for your inline mathematics must go inside one of the following delimiterss.

  - `\( \)`
  - `$ $`

  Good practice is to use the former - the latter is syntactically ugly because you're using the same symbol to start and end your (mini-)environment. However the latter is definitely faster to type.

- Display math groups all of your maths into its own environment and is also automatically centre aligned. This is basically equivalent to writing a math equation down the middle of the page. The code for your display mathematics goes inside the `\[ \]` delimiterss.

By default, inline mathematics shows everything in a more compact style. For example, `\( \sum_{k=1}^n k = \frac{n(n+1)}{2} \)` shows up as $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$.

But if I switch for its display math counterpart `\[ \sum_{k=1}^n k = \frac{n(n+1)}{2} \]`, it looks more spaced out:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Some side remarks:

- The `\( \)` and `\[ \]` delimiters also work in Facebook Messenger on the computer now. This allows people to send mathematical equations in LaTeX through as messages. Note, however, it is rendered in a different way through MathJax.
- It is possible to override the compactness of inline LaTeX by adding `\displaystyle` right after `\(`. This isn't usually advised by experts though - it messes up spacing a lot.
- There is also the double dollar sign delimiter `$$ $$` for display maths. This is, however, **strongly not advised**. If you're really interested in why, there is a discussion on the topic here.
- Neither of the above environments support line breaking, which makes multi-lined equations annoying. See the 'Aligning equations' section for information about this.

3.2 Syntax galore

Every mathematical symbol existence can be found here. What follows here are the ones I feel will be sufficient for your needs, at least for a while.

3.2.1 Basics in math mode

Ordinary symbols are typed in the usual way. For example, `\( \color{magenta} 8! = 40320\)` outputs $8! = 40320$ as expected. If a symbol is not listed below, it's likely it is on your keyboard.

Description	Syntax	Inline mode	Display mode
Multiplication	`\( \times \)`	$\times$	$\times$
Division	`\( \div \)`	$\div$	$\div$
Inequal	`\( \neq \)` or `\( \ne \)`	$\neq$	$\neq$
Less than or equal	`\( \leq \)` or `\( \le \)` `\( \leqslant \)`	$\leq$ $\leqslant$	$\leq$ $\leqslant$
Greater than or equal	`\( \geq \)` or `\( \ge \)` `\( \geqslant \)`	$\geq$ $\geqslant$	$\geq$ $\geqslant$
Superscripts (powers)	`\( a^{x} \)`	a^{x}	a^{x}
Subscripts (bases)	`\( S_{n} \)`	S_{n}	S_{n}
Fractions	`\( \frac{a}{b} \)`	$\frac{a}{b}$	$\frac{a}{b}$
Surds (roots)	`\( \sqrt[3]{729} \)`	$\sqrt[3]{729}$	$\sqrt[3]{729}$

Note the use of braces for many of the commands. The braces are important when you want more than one character contained in a group. For example, if I wanted to type e-to-the-$2x$, if I type `\( e^2x \)` I get e^2x, and the x does not go in the superscript. I have to type `\( e^{2x} \)` to get my desired output e^{2x}.

3.2.2 Brackets

Description	Syntax	Inline mode	Display mode
Braces (sets)	`\{x\}`	$\{x\}$	$\{x\}$
Absolute values	`|x|`	$\|x\|$	$\|x\|$
Norms	`\\|x\\|`	$\|\|x\|\|$	$\|\|x\|\|$
Floor function	`\lfloor x \rfloor`	$\lfloor x \rfloor$	$\lfloor x \rfloor$
Ceiling function	`\lceil x \rceil`	$\lceil x \rceil$	$\lceil x \rceil$
Angle brackets	`\langle x \rangle`	$\langle x \rangle$	$\langle x \rangle$

Note that the absolute value brackets are on most keyboards. The angle brackets are not commonly used until second year mathematics, when inner products and cyclic groups are introduced.

Brackets don't work too well with fractions, surds and many other TEXstructures. Consider the following.

```
\[ ( \frac{1}{x} ) \]
```

$$(\frac{1}{x})$$

As you can see, the bracket doesn't wrap around the entire fraction. By default, brackets only wrap around ordinary sized text, and fractions are usually bigger than this. The workaround is to use `\left` and `\right` to automatically resize the bracket.

```
\[ \left( \frac{1}{x} \right) \]
```

$$\left(\frac{1}{x} \right)$$

To be fair, the syntax is a bit awkward and tiring to type very often. You may wish to consider adding this macro to your preamble.

```
\newcommand{\brac}[1]{\left(#1\right)}
```

This will let you type something like `\brac{ \frac{1}{x} }` for some easier brackets. You can declare more macros if you require other brackets as well.

Brackets in LaTeX can also be manually resized using `\big`, `\Big`, `\bigg` and `\Bigg`. There are few situations where you'd want to do this though.

3.2.3 Spaces

A space written in mathematics mode is ignored altogether. If you want to introduce spacing, you have to do it with the appropriate command.

Description	Syntax	Output
Negative space	`x \! y`	xy
No space	`x y`	xy
Small space	`x \, y`	$x\,y$
Medium space	`x \: y` or `x \> y`	$x\:y$
Large space	`x \; y`	$x\;y$
Space identical to normal text space	`x \ y`	$x\ y$
Space equal to current font size	`x \quad y`	$x\quad y$
Space double length of `quad`	`x \qquad y`	$x\qquad y$

Note how the negative space, if anything, makes the characters more closer to each other.

3.2.4 Fonts in math mode

These are some commonly used alternate fonts in mathematics mode. You can compare it to characters in the default mathematics font:

$ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890$

- Math roman: Sometimes used to force letters in mathematics to stand upright. Usage: `\mathrm{text}`.
  $\mathrm{ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890}$
- Math italicised: Technically speaking used for italicising, but the default math font is already slanted. But it also italicises the numbers, which is different. Usage: `\mathit{text}`
  $\mathit{ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890}$
- Math bold font: The *intended* way of producing bold text in math mode. Note that the characters are no longer italicised. Usage: `\mathbf{text}`
  $\mathbf{ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890}$

The following font families will only work on capital letters.

- Math calligraphy: A special calligraphical font. Usage: `\mathcal{TEXT}`.
  $\mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$
- Math blackboard bold: A special font set, commonly used for number sets (e.g. integers, real numbers, complex numbers). Usage: `\mathbb{TEXT}`
  $\mathbb{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$

3.2.5 Operators

Many famous mathematical functions should not be written in the usual mathematics format. Rather, the letters should be upright, much like what's given by `\mathrm{}`. Operators are commands that allow these functions to be distinguished from other variables.

Operators are made in a similar way as new commands. Something of the following form needs to be added to your preamble.

```
\DeclareMathOperator{\commandname}{nameofoperator}
```

For example, if I add `\DeclareMathOperator{\Var}{Var}` to my preamble, I will get an operator `\Var`.

If you only plan to use an operator once, and cannot be bothered to change your preamble, you can use `\operatorname{name}` instead in your math environment. As an example involving inverse hyperbolic sine:

```
\[\operatorname{arcsinh} x = \ln \left( x + \sqrt{x^2+1} \right)\]
```

$$\operatorname{arcsinh} x = \ln \left(x + \sqrt{x^2+1} \right)$$

Before you jump into making too many operators, you should be advised that many of them already exist! As a general rule of thumb, you should **always use the ones listed below wherever possible** (for example, `\sin` every time you use the sine function. In general, it just makes your maths a lot neater to read.

Description	Syntax	Output
Natural exponential	`\exp x`	$\exp x$
Natural logarithm	`\ln x` or `\log x`	$\ln x$, $\log x$
Logarithm with any base	`\log_{a} x`	$\log_a x$
Sine	`\sin x`	$\sin x$
Cosine	`\cos x`	$\cos x$
Tangent	`\tan x`	$\tan x$
Cotangent	`\cot x`	$\cot x$
Secant	`\sec x`	$\sec x$
Cosecant	`\csc x`	$\csc x$
Inverse sine	`\arcsin x`	$\arcsin x$
Inverse cosine	`\arccos x`	$\arccos x$
Inverse tangent	`\arctan x`	$\arctan x$
Hyperbolic sine	`\sinh x`	$\sinh x$
Hyperbolic cosine	`\cosh x`	$\cosh x$
Hyperbolic tangent	`\tanh x`	$\tanh x$
Hyperbolic cotangent	`\coth x`	$\coth x$
Probability	`\Pr (X=x)`	$\Pr(X = x)$
Determinant	`\det A`	$\det A$
Dimension	`\dim S`	$\dim S$
Kernel	`\ker S`	$\ker S$
Minimum	`\min S`	$\min S$
Maximum	`\max S`	$\max S$
Infimum	`\inf S`	$\inf S$
Supremum	`\sup S`	$\sup S$

3.2.6 Greek letters

The Greek alphabet is built in through commands in LaTeX. Note that if the upper case version of the letter looks like a letter in the English alphabet, it is not defined.

Some Greek letters also have a 'variant' form. It's intended to represent the same Greek letter, but looks physically different.

Syntax	Uppercase	Syntax	Lowercase	Syntax	Variant lowercase
	A	\alpha	α		
	B	\beta	β		
\Gamma	Γ	\gamma	γ		
\Delta	Δ	\delta	δ		
	E	\epsilon	ϵ	\varepsilon	ε
	Z	\zeta	ζ		
	H	\eta	η		
\Theta	Θ	\theta	θ	\vartheta	ϑ
	I	\iota	ι		
	K	\kappa	κ	\varkappa	$\varkappa$
\Lambda	Λ	\lambda	λ		
	M	\mu	μ		
	N	\nu	ν		
\Xi	Ξ	\xi	ξ		
	O		o		
\Pi	Π	\pi	π	\varpi	ϖ
	P	\rho	ρ	\varrho	ϱ
\Sigma	Σ	\sigma	σ	\varsigma	ς
	T	\tau	τ		
\Upsilon	Υ	\upsilon	υ		
\Phi	Φ	\phi	ϕ	\varphi	φ
	X	\chi	χ		
\Psi	Ψ	\psi	ψ		
\Omega	Ω	\omega	ω		

Note that omicron looks exactly like the English letter o so not even its lower case version is defined...

3.2.7 Calculus toolbox

Objects in calculus very often look different in the two math modes. Each symbol will be introduced as its own dot point, along with a corresponding example. However where applicable, both math modes will be used.

- Infinity: The command for infinity is, somewhat strangely, `infty`. This is to distinguish it from the infimum.

Syntax	Output
\infty	∞

- Limits: These are also operators in LaTeX, but like logarithms, subscripts are used to denote what the limits tend to. The arrow is usually generated with the `\to` command. The syntax is `\lim_{approach} expression`.

Syntax	Inline mode	Display mode
`\lim_{x \to 3} x^2 = 9`	$\lim_{x \to 3} x^2 = 9$	$\lim\limits_{x \to 3} x^2 = 9$

- Derivatives: $\frac{d}{dx}$ is usually just typed out as a fraction. However, if you want to use the $f'(x)$ notation, you need to use `\prime` inside a superscript.

Syntax	Output
`f^{\prime}(x) = 3x^2`	$f'(x) = 3x^2$

  Many students in high school may have learnt it as the 'f-dash' notation. For the second derivative, you can just add more `\prime`s in the superscript.

- Partial derivatives: Instead of using d's in the fraction, you can use `\partial` to generate the ∂ symbol.

Syntax	Inline mode	Display mode
`\frac{\partial}{\partial y}`	$\frac{\partial}{\partial y}$	$\dfrac{\partial}{\partial y}$

- Sums: Sigma notation is supported in LaTeX and usually used alongside superscripts and subscripts. They are used to provide summation boundaries. The `\sum` command is used.

Syntax	Inline mode	Display mode
`\sum_{k=1}^{n} k^2`	$\sum_{k=1}^{n} k^2$	$\displaystyle\sum_{k=1}^{n} k^2$

- Products: Capital pi notation works identically to sigma notation. Basically just replace `\sum` with `\prod`

Syntax	Inline mode	Display mode
`\prod_{k=1}^{n} k`	$\prod_{k=1}^{n} k$	$\displaystyle\prod_{k=1}^{n} k$

- Integrals: Integrals follow the exact same format as the above. Unsurprisingly, we change the command to `\int`

  The one nitpicky thing about integrals is that there should always be a space before your differential, say, dx. There are proper ways of dealing with this, but a lazy workaround that will suffice for now is just to use this space: `\,`.

Syntax	Inline mode	Display mode
`\int_{0}^{2}x\,dx = 2`	$\int_0^2 x\,dx = 2$	$\displaystyle\int_0^2 x\,dx = 2$

Indefinite integrals can be written by simply omitting the superscript and subscript.

- Multiple integrals: If you've already found apporpriate boundaries of integration, you can just use multiple `\int`s one after the other.

  However when you don't know these boundaries and are still integrating over a generic region, say Ω, there are other symbols recommended. For double integrals you should use `\iint`, and for triple integrals you should use `\iiint`. This is recommended because it reduces the amount of spacing between each integral symbol.

Syntax	Inline mode	Display mode
`\iint_{S} xy\, dx\,dy`	$\iint_S xy\,dx\,dy$	$\displaystyle\iint_S xy\,dx\,dy$

- Closed line integral: When doing vector calculus, if you wish to remind the reader you are integrating on a closed loop, there is the option to use `\oint`.

Syntax	Inline mode	Display mode
`\oint_{C} \mathbf{F} \cdot d\mathbf{r}`	$\oint_C \mathbf{F} \cdot d\mathbf{r}$	$\displaystyle\oint_C \mathbf{F} \cdot d\mathbf{r}$

  For an equivalent version for multiple integrals, i.e. `\oiint`, you need to include the `wasysym` package.

- Vector differential operators: The inverted triangle in the grad/div/curl is known as 'del' or 'nabla'. In LaTeX `\nabla` is how it is encoded.

Syntax	Output
`\nabla f(x,y,z)`	$\nabla f(x,y,z)$

- Piecewise functions: These are actually made through the `cases` environment inside math mode. An ampersand `&` is used to separate the condition from the expression, much like tables. The new line character `\\` is also used to move to the next line.

```
\[
    f(x) =
    \begin{cases}
        x - 3   & x \leq 0\\
        2x^2    & 0 < x < 3\\
        e^{-x}  & x \geq 3
    \end{cases}
\]
```

$$f(x) = \begin{cases} x - 3 & x \leq 0 \\ 2x^2 & 0 < x < 3 \\ e^{-x} & x \geq 3 \end{cases}$$

3.2.8 Algebra toolbox

Syntax in linear algebra is generally the same, but it can get a bit nasty to adapt to.

Vectors are generally typed using `\mathbf{}`, or with an arrow on top. However with the arrow notation, I personally favour `\overrightarrow{}` over `\vec{}`. This is because the latter always produces a small arrow, whereas `\overrightarrow{}` knows how to resize itself as appropriate.

```
\[ \vec{x} \, \overrightarrow{x} \, \vec{OA} \, \overrightarrow{OA} \]
```

$$\vec{x} \, \overrightarrow{x} \, \vec{OA} \, \overrightarrow{OA}$$

You may feel that the arrow is *too* big on $\overrightarrow{x}$ - fair enough if you do. But you can see why it's perhaps recommended when your vector goes from a point to another point.

Both the column vector notation and matrices are created through one of a series of `matrix` environments. Each of these allow you to display a 2D array of numbers.

- `matrix` will produce only an array of numbers.
- `bmatrix` will automatically enclose it in square brackets.
- `pmatrix` will automatically enclose it in parentheses.
- `vmatrix` will automatically enclose it in absolute value bars; this is sometimes used for the determinant. Note however, as mentioned earlier, there is a built in operator `\det` as well.

Matrices are similar to tables in that ampersands `&` are used to move along a row, and newline characters
`\\` take you to the next row. Unlike tables, you do not have to declare the number of rows *nor* columns beforehand.

Generally speaking, I recommend writing your matrix on multiple lines; one for each row. This helps you clean up errors more easily.

```
\[
    I =
    \begin{pmatrix}
        1 & 0 & 0\\
        0 & 1 & 0\\
        0 & 0 & 1
    \end{pmatrix}
\]
```

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

You may want to declare operators for the image, nullity, rank, projection and other concepts encountered in Math 1B. Be careful with span though - naming it `\span` won't work. I recommend naming it `\Span`.

Augmented matrices are not built into LaTeX and the environment needs to be created. This has already been discussed on TeXStackExchange already here. For your purposes, it should be enough to firstly add this to your preamble:

```
\newenvironment{amatrix}[1]{%
    \left(\begin{array}{@{}*{#1}{c}|c@{}}
}{%
    \end{array}\right)
}
```

Then, as far as this implementation goes only the last column will be augmented to the right. In the `amatrix` environment, as a compulsory parameter, you declare how many columns you want on the left. The example provided on the TeXStackExchange page specifies two columns on the left.

```
\[
    \begin{amatrix}{2}
       1 & 2 & 3 \\
       a & b & c
    \end{amatrix}
\]
```

$$\left(\begin{array}{cc|c} 1 & 2 & 3 \\ a & b & c \end{array}\right)$$

This code can be adapted if you require more columns following the vertical bar, but that's beyond the scope of the guide.

As for complex numbers, there are a few things to note.

- The complex conjugate is just one of many situations where you need a bar over the letter. This can be achieved using `\overline{z}`, which produces $\overline{z}$.
- The multi-valued argument is built in as the `\arg` operator. For example, `\arg z` just produces $\arg z$.
- The real and imaginary parts are technically speaking also built in as operators, but they use another font family `mathfrak` which I have never needed. Typing `\Re z` and `\Im z` gives me $\Re z$ and $\Im z$.

  You may not like this, so you can just add these two lines to your preamble to get the more classic versions of the real/imaginary parts.

  ```
  \renewcommand{\Re}{\operatorname{Re}}
  \renewcommand{\Im}{\operatorname{Im}}
  ```

  Basically, we just overwrite the definitions of the operators as appropriate.
- The principal argument is not built in and must be introduced as a new operator as well in your preamble: `\DeclareMathOperator{\Arg}{Arg}`.

3.2.9 Discrete mathematics toolbox

This section basically notes down symbols that MATH1081 students would probably benefit from, that aren't already in Mathematics 1A or 1B.

Topic 1:

Description	Syntax	Inline mode	Display mode
Empty set	`\emptyset`	$\emptyset$	$\emptyset$
Element of	`a \in A`	$a \in A$	$a \in A$
Subset of	`A \subseteq B`	$A \subseteq B$	$A \subseteq B$
Proper subset of	`A\subset B`	$A \subset B$	$A \subset B$
Set union	`A \cup B`	$A \cup B$	$A \cup B$
Set intersection	`A \cap B`	$A \cap B$	$A \cap B$
Set difference	`A \setminus B`	$A \setminus B$	$A \setminus B$
Big set union	`\bigcup_{i=1}^{n} A_i`	$\bigcup_{i=1}^{n} A_i$	$\bigcup_{i=1}^{n} A_i$
Big set intersection	`\bigcap_{i=1}^{n} A_i`	$\bigcap_{i=1}^{n} A_i$	$\bigcap_{i=1}^{n} A_i$

Note that `\ni` flips the `\in` symbol, which changes 'containment' into 'ownership'. Similarly `\subseteq` can be flipped into `\supseteq` for supersets. Note however that these notations are far more annoying to read in practice, and you definitely should not use them in MATH1081.

Topic 2:

- Divisibility should be written with the `\mid` syntax instead of absolute value bars, as this will introduce a space between the numbers. For example, `2\mid 4` gives $2 \mid 4$.
- `\nmid` can be used for 'not divides'.
- `\mod` is a built in operator for modulo arithmetic, but by default it introduces a large space before it. If you don't like this, you can use `\bmod` instead. For example, `5 \bmod 2 = 1` gives $5 \bmod 2 = 1$.
- Tilde is built in like `\vec` - it places the swiggle on a letter. For example, `\tile{a}` produces $\tilde{a}$.

  If you require the conventional symbol for equivalence relations, it is the same symbol as what's internationally used for similar triangles - `\sim`. For example, `x \sim y` produces $x \sim y$.
- The preceeds symbol in partial orders is accessed via `\preceq`. For example, `x \preceq y` produces $x \preceq y$.

  Some notes: A simliar syntax exists for 'strictly preceedes' - just remove the `eq` bit and type `\prec`. The sign can also be flipped to produce a 'succeeds' symbol, which can similary be written as `\succeq` or `\succ`. But you won't be using the succeeds symbol in this course.

Topic 3:

Description	Syntax	Inline mode	Display mode
Or	`p \vee q` or `p \lor q`	$p \vee q$	$p \vee q$
And	`p \wedge q` or `p \land q`	$p \wedge q$	$p \wedge q$
Multiple or	`\bigvee_{i=1}^{n} p_i`	$\bigvee_{i=1}^{n} p_i$	$\displaystyle\bigvee_{i=1}^{n} p_i$
Multiple and	`\bigwedge_{i=1}^{n} p_i`	$\bigwedge_{i=1}^{n} p_i$	$\displaystyle\bigwedge_{i=1}^{n} p_i$
For all	`\forall`	$\forall$	$\forall$
There exists	`\exists`	$\exists$	$\exists$
Uniquely exists	`\exists!`	$\exists!$	$\exists!$
Implies	`\implies`	$\implies$	$\implies$
Implied by	`\impliedby`	$\impliedby$	$\impliedby$
Equivalence	`\iff`	$\iff$	$\iff$

Note that in MATH1081, the tilde symbol is also used for the negation. But in various other contexts, `\neg p` is used, which produces $\neg p$. Note also that `\impliedby` should be avoided wherever possible - don't write a proof backwards!

Topic 4:

- The binomial coefficient, also used for a combination, uses the `\binom{}{}` command. Note that there are two arguments - one for the upper value, and one for the lower value. Example:

```
\[ \binom{n}{k} = \frac{n!}{k!(n-k)!} \]
```

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

3.2.10 Miscellaneous symbols

These symbols probably play little role in first and second year mathematics as a whole, but they may have other uses, hence they are mentioned here.

- In front of many relational operators like `\sim`, `\geq`, `\preceq` etc., you can add `\not` in front of it to produce a stroke through the symbol. This helps denote something is *not* something else.
- Dots have a variety of uses in mathematics. Here are some commonly used ones.
  - `\cdot` $\cdot$ is used in contexts involving multiplication, including the dot product. It is preferred over the period (classic full-stop) which should be reserved for decimals. Basically as hinted by its name, it centre aligns the dots.
  - `\dots` $\dots$ is commonly used when referring to a sequence of objects.
  - `\cdots` $\cdots$ can be used when doing the same operation on a sequence of objects, for example a sum.

- `\vdots` $\vdots$ gives vertical dots that become useful in some kind of listing, for example matrices of arbitrary size. They may also be used in aligning - see below.
- `\ddots` $\ddots$ is an alternate to `\vdots` for when diagonal dots are preferable.

- A huge amount of arrows have been programmed into LaTeX. Listing those out would flood the guide a bit too much, so I'll just link to one possible source for them here.

- As an example, occasionally when dealing with sums in sigma notation, sometimes there may be multiple conditions on what to sum over. The `\substack` command allows you to specify multiple conditions on a sum if appropriate. Each condition is specified on a new line. Example:

```
\[ \sum_{\substack{k=1 \\ 5 \mid k}}^{20} k = 5+10+15+20 =50 \]
```

$$\sum_{\substack{k=1 \\ 5 \mid k}}^{20} k = 5+10+15+20 =50$$

- Again using sums as an example, sometimes we might lose track of how many terms there are when expanding a sum out. An `\underbrace{}_{}` command can be used to help us keep track of this. In the first set of braces we indicate what we want the underbrace to stretch out for, and in the second set we provide our annotation. Example:

```
\[ \underbrace{2 + 2 + 2 + \dots + 2}_{n\text{ terms}} = 2n \]
```

$$\underbrace{2 + 2 + 2 + \dots + 2}_{n \text{ terms}} = 2n$$

3.2.11 A remark on plaintext

Ordinary text can also be typed in math mode. You just need to use the `\text{}` command. Your ordinary text goes in the brackets. The commands `\textbf{}` and `\textit{}` will also work.

However, personally I have found little to no use for ordinary text in a math environment anymore.

3.3 Aligning equations

This is one of LaTeX's most powerful features and one that I love the most. In my working out, I'll usually write not just one equation, but a whole list of equations one after the other. But to make my working out as tidy as possible, I would write equal signs (and any other relational operators) underneath each other.

The `align` and `align*` environments allows LaTeX to do it as well. You really should be

using this 100% of the time you have a whole group of equalities (or anything similar) to list out.

The difference between `align` and `align*` is that the former will label each equation for you. But it's unlikely that you'd need that ability for a while, so I demonstrate using only `align*`.

3.3.1 Demonstrating the classic `align*`

The align environment also makes use of `\\` for new lines and `&` as a separator. Usually, the ampersand is used to split the LHS from the RHS. The relation (for example, equal sign) is typically placed after the ampersand, into the RHS.

The align environment is a 'standalone' environment in the sense that it should not be used inside the `\( \)` or `\[ \]` delimiters. You should always jump straight into an `align*`. Of course, each equation should be on its own separate row.

The following is an example of `align*` being used for an integration by parts problem.

```
\begin{align*}
    \int x^2 \ln x\,dx
    &= \frac{x^3}{3}\ln x - \int \frac{x^3}{3}\frac{1}{x}\,dx\\
    &= \frac{x^3}{3}\ln x - \int \frac{x^2}{3}\,dx\\
    &= \frac{x^3}{3}\ln x - \frac{x^3}{9} + C.
\end{align*}
```

$$\begin{aligned}
\int x^2 \ln x \, dx &= \frac{x^3}{3}\ln x - \int \frac{x^3}{3}\frac{1}{x}\,dx \\
&= \frac{x^3}{3}\ln x - \int \frac{x^2}{3}\,dx \\
&= \frac{x^3}{3}\ln x - \frac{x^3}{9} + C.
\end{aligned}$$

Note: The first two equations could have been put on the same line. The only reason I didn't was because of formatting issues.

3.3.2 Tags

At certain times, you may wish to provide extra explanation on what you did. For example, in the first line, it would've been nice if I could indicate that I used integration by parts somewhere.

Tags are one way of achieving this. A tag is a block of text to the right, **by default in text mode**, that shows up in a bracket as though it were an annotation. A tag should be included in the same line of code corresponding to what gets displayed, but *before* the newline character. In the above example, ignoring all the other lines:

```
\begin{align*}
    \int x^2 \ln x\,dx
    &= \frac{x^3}{3}\ln x - \int \frac{x^3}{3}\frac{1}{x}\,dx
    \tag{integration by parts}\\
\end{align*}
```

$$\int x^2 \ln x \, dx = \frac{x^3}{3} \ln x - \int \frac{x^3}{3} \frac{1}{x} \, dx \qquad \text{(integration by parts)}$$

Notes:

- Tags also work inside the `\[ \]` delimiters.
- If you wish to insert mathematics in a tag, you have to open a set of `\( \)` or `$ $` delimiters inside said tag.

Another application of tags is in labelling equations manually. If you want consistent labelling then it may be better to switch back to the `align` environment, i.e without the asterisk. But if you only intend to refer back to one equation over and over again, something like `\tag{1}` on the same line as that equation may be very useful.

3.3.3 Beyond the `align` environment

Information for environments similar to `align` can be found in the documentation for the `amsmath` package here. Some worthwhile mentions, however:

- The `alignat*` environment can be used when dealing with three-sided equalities/inequalitites. The only reason I don't talk about them here is because I struggle to use it myself!
- The `gather*` environment is a good alternative for a series of equations that require no alignment at all.

Also, a quirk about these environments is that you cannot have blank lines in them. If you do, LaTeX will shout at you for it!

Chapter 4

I just want to type a maths assignment... help?

Yeah, look, I get 'ya. I wouldn't be bothered to read this whole thing either if I had a maths assignment due and I was feeling swamped.

Start by copying this code into your `.tex` file:

```
\documentclass[11pt, a4paper]{article} % resize however you feel appropriate.
\usepackage{microtype}
\usepackage{mathtools}
\usepackage{amssymb}
\usepackage[margin=1in]{geometry} % comment out if you don't mind default margins
%\usepackage{graphicx} % uncomment if you want pictures
%\usepackage{xcolor} % uncomment if you want colour

\begin{document}


\end{document}
```

Now ignore all the magical stuff and put everything inside that `\begin{document} \end{document}` block. If anything can be written without going into mathematics mode, *don't* go into mathematics mode. Just type it as you normally would in Microsoft Word. If you need a new line, just use `\\ \\`.

When you have a mathematics equation to type, if it can go in a paragraph, put it inside these delimeters: `\( \)}`. If it likely requires its own block down the middle of the page, use these instead: `\[ \]`. If you bump into something that involves anything similar to a whole line of equations down the page, however, use the `align*` environment. An example is provided in section 3.3.1 for you to adapt.

Whenever you need a mathematical object, it'll likely be addressed somewhere in this file. Use Ctrl+F to search for a keyword and hopefully it will appear!

And to be fair, a huge bulk of chapter 2 can be ignored if writing an assignment is all you care about with LaTeX. The hardest thing will literally be to search for all the mathematics syntax when you need it! Once you're adequately proficient, you can get away using LaTeX cheat sheets instead, and eventually you won't need any help at all.

Chapter 5

A small introduction to TikZ

This section will be completed in version 2 of the guide!

Chapter 6

A small introduction to beamer

This section will be completed in version 2 of the guide!