IBTEX

A Template for Writing the IB Extended Essay and Internal Assessments.

mholson

Words: 1234

Contents

1 Introduction to IBTEX

§1.1 Purpose

This template is designed for International Baccalaureate™ students who are looking to write an extended essay or an internal assessment that is formatted using LATEX. The template is based on the KOMA-Script scrbook class and further customized using a number of custom style files, including `mhotext.sty`[1] that is inspired by the code Evan Chan used create Napkin[2].

While I have done my best to ensure this template meets the submission requirement of the IB, I cannot guarantee your document will meet the these standards. It is the responsibility of the student to ensure that any work submitted to the IB for grading is formatted correctly by ensuring they are following the official IB documentation and the advice giving by their supervising teacher.

While there is extensive online support and documentation on how to publish using the LATEX programming language, it can still be overwhelming for anyone who is getting started. I have been navigating the TEX ecosystem since 2005 and still learning. I want to share some of the knowledge and ideas that I have accumulated along the way, so that you can get writing as quickly as possible. You should be focused on writing an amazing manuscript and then put this template take care of the typesetting for you.

Before going any further, I feel that it is important that we establish that this documentation is written for high school students with little to no prior TEX knowledge. While you should be able to typeset your document without an understanding of that TEX is, how it works and a concise overview of TEX is and how it has developed over the past 40^{+} years, I would recommend that you jump to chapter 06 before going any further If you are not quite sure what the difference is between LATEX and TEX, then I would highly recommend that you jump to chapter 06 for concise overview of TEX is and how it has developed over the past 40^{+} years. While it is not imperative that you have an understanding of how TEX has evolved, it does bring some useful context to how it works today.

§1.2 What is LATEX?

so let's try to distill things down the necessities with a little historical context to guide us. The programming language TEX was created by Donal Knuth in the late 1970s to automate the process of formatting text ready for publication, a process called typesetting. It worked by the user creating a `foo.tex` text file formatted with commands collectively called plainTEX. This file would then be interpreted by a computer program called the TEX engine, which would output a device independent file, `foo.dvi`, that could be sent to printer or converted to another file format such as `foo.ps` or `foo.pdf`.

[1] https://github.com/mholson/mhoDotFiles/texmf/
[2] http://web.evanchen.cc/napkin.html

However, in the 1980s Knuth strategically decided to that no new features would be added to the TEXengine as Knuth valued the benefits of an engine that could produce the same output over new features being added. This has held true up to today with the exception of one last feature being added in 1989 resulting in TEX3 [1]. With each bug fix update to TEX3 the version number converges to π and in 2021 it reached version 3.141592653[2].

Since TEX is a programming language we are going to need a text editor to write our code and a compiler to transform that code into a pdf document. To keep things simple, I would highly recommend using Overleaf[3], which will provide you with both a modern text editor and a range of compilers that can all be used from the comfort of your favorite web browser. If you want to compile your code on your own machine, then you will need to learn how to do that elsewhere.

Before you start coding, I would highly recommend that you spend some time reading The Not So Short Introduction to LATEX 2_ε [2]. Maybe start with at least the first two chapters just to give you some history and context to TEX. Keep a link to this documentation close by as it makes for a great reference manual.

This document has been written so that you should be able to get into the weeds if you feel the need, but more importantly ensure that you can get writing almost immediately. So now for a very generalized overview of TEX.

You are probably familiar with the idea of writing your document all in one file using a word processor like MS Word or Google Documents. A shift in mindset will be needed as your document will be now be an object called a project made up of multiple files and folders.

§1.3 The Big Idea

Basically, we are going to input code stored in a `main.tex` file into a compiler that will generate an output file. The code read by the compiler is written in using rhe written using the TEX programming language,

§1.4 Project Structure

> **Example 1.** Including an image. When inserting an image into a MS Word document, you would insert the image and it would become part of the document and you would no longer see that image file associated with your document.
>
> When inserting an image into a LATEX document, we upload a copy of the image into our project and then insert a line of code that will place that image into our document.

While the number of files can grow when writing a LATEX document, there are some pretty awesome advantages such as only having to modify or update an image file and the changes will be visible in your document following the next compilation of your code.

I am starting to get ahead of myself here. Let's

[3] https://overleaf.com

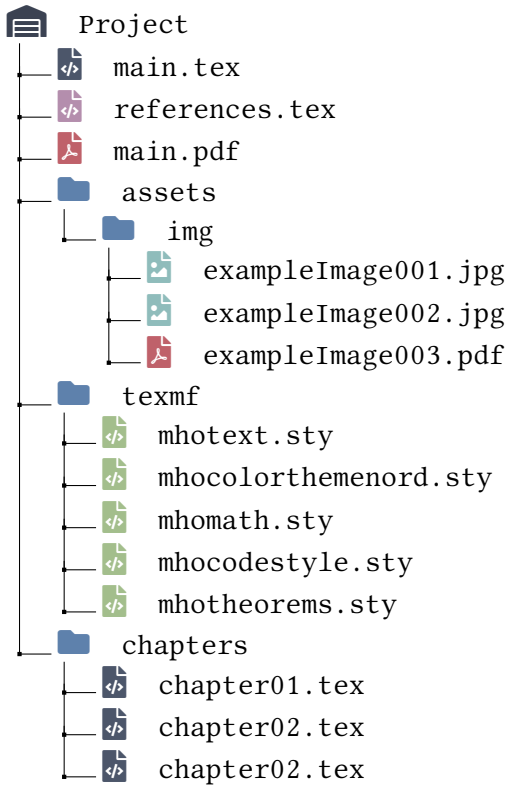


Figure 1.1: Example of a project file structure.

It doesn't stop there! All the template files necessary to format your document will also exist in your project. Don't worry, these files will be tucked away in a folder called `texmf`.

Our project is going to be made up of three folders and one main file called `main.tex`. Let's describe the contents of those three folders.

texmf You probably will not do very much with the files located in this folder as these are the template files that are responsible for formatting the content of your essay. In most cases, you can change the formatting elements of your document right from the `main.tex` file. It is also good practice to not edit these files directly, so that your theme can easily be upgraded if necessary.

assets An asset is a file that you are going to include in your document. In most cases, the only files that you will be including in your document will be images and thus you can upload your images directly to this folder. You can have as many subfolders within your assets folder, but again this folder should be good enough to upload your images into.

chapters This folder is really optional. You could technically write your document all within the `main.tex` file; however, it might be beneficial to break your document up into chapters and write these in separate files. If you are not going to be structuring your paper using chapters, then you should probably just write your document in the `main.tex` file.

Like I said before, I want to get you up and writing as quickly as possible. We are going to get right into editing the `main.tex` file and producing a pdf file with some content. Action!

§1.5 The Main File

The `main.tex` file is where all the action takes place. It is in this file where you will write your content or at least link to the content found in other files in your project. This file can be as simple or as complicated as you would like it to be. I have provided a skeleton template of this file in your template project. So let's start looking at the code.

For us the file is going to start of with multiple lines of text that are going to be ignored by the compiler, which means it will not be included in our pdf output.

§1.6 texmf - The Template Files

Most of you will never need to look in this folder; however, I thought I would provide a short summary of what files are included and how they are being used to format your document.

§1.7 Requirements

All the file dependencies outside the scope of a typical TEX distribution are available from my `texmf` repository hosted on Github[4]. The Overleaf™version will come with batteries included in the `texmf` directory, which in include.

`maabook.cls` the main template file.

`mhotext.cls` customizations made the to main template file.

`mhotheorems.sty` styling custom theorem environments

`mhocolorthemenord` custom color theme based on Nord Theme.

`mhocodestyle.sty` styling code via the listings package.

`mhomath.sty` some of my custom math commands.

§1.8 Getting Started

> **Definition 1** (Code Comment)**.** Any code that is written in your file that is ignored by the compiler is called **commented text**. All characters on the same line following a `%` symbol will be regarded as commented text and ignored.

Comments are great for including additional information, making temporary changes to your document (without having to erase) and documenting your code for others to follow — including yourself!

[4] https://github.com/mholson/mhoTexmf/tree/main/texmf

```
% This line would be ignored by the compiler.
This line is read by the compiler.

The compiler is reading % and now it is ignoring.
```

This line is read by the compiler.
The compiler is reading

One of the benefits of writing in LATEX is the ability to write comments that are not typeset. Any text following a % character will be ignored. Not only is this ideal for leaving comments and ideas, but it is als a way to make temporary changes to your document.

```
% This code will be ignored
This text will be visible % while this text will be ignored.
```

This text will be visible

Much of your document will be filled with commands that take the form . In fact the first non-comment line of your document will be the command that defines the document template.

```
\documentclass[a4paper]{maatext}
```

more text here

2 Typesetting Mathematics

This chapter will focus on inserting mathematics into your paper. LaTeX comes with basic mathematics support out of the box, so we will need to incorporate some additional packages to enhance the range of formulas that can be included in our document. You can think of it as adding the functionality of an equation editor into your current word processor. As you become more familiar with the syntax, you will quickly appreciate being able to type your formulas with the same ease you write text. LaTeX comes with two modes that are of interest to us:

1. normal and
2. math, which has the two styles
    a) inline and
    b) display.

Normal mode is the default mode and where you will write the non-mathematical text of your document. If you want to include expressions and equations, then you are going to need to use either an inline or display math mode, which are environments specific for mathematics. However, we are going to want to include more advanced mathematical formatting such as multiline equations and for this we are going to include some additional packages including those from the $\mathcal{AMS}$, American Mathematics Society, and the mathtools[4] packages which will bring some enhancements to the $\mathcal{AMS}$ math package[3]. Don't worry, these packages are loaded by default when using this style file; however, you should be aware which packages are being used so that you know which documentation to reference for help beyond this quick start up guide.

To get us started, I am going to briefly introduce both the inline and display math environments that are available. Then I will provide some of the more common features available from the $\mathcal{AMS}$math package. Once we have established the environments in which we can write mathematics, we can look at how we can format special characters and symbols such as operators, operands and function names. To make our lives a bit easier, we will include some additional packages along the way and I will share some of the custom macros that are available through the mhomath.sty file.

§2.1 Inline Mathematics

In order for the code that you write to be interpreted by the compiler as mathematics, you need to toggle between **normal mode** and **inline math mode** by using the math environment.

```
\begin{math}
  ⟨mathematics content goes here⟩
\end{math}
```

```
Solve all equations of the form
\begin{math}
  ax^2 + bx + c = 0
\end{math}
for
\begin{math}
  x \in \mathbb{R}
\end{math}
by completing the square.
```

Solve all equations of the form $ax^2 + bx + c = 0$ for $x \in \mathbb{R}$ by completing the square.

From the above example, we can see that the code is verbose and difficult to read. Thankfully, this environment has a shortcut notation that should **ALWAYS** be used. The macro `\(` is used to toggle into inline math mode and the macro `\)` is used to toggle back to normal mode.

```
< normal mode text > \( < math environment code > \) < normal mode text >
```

We can now revisit our original example using this simplified notation.

```
Solve all equations of the form \( ax^2 + bx + c = 0 \)
for \( x \in \mathbb{R} \) by completing the square.
```

Solve all equations of the form $ax^2 + bx + c = 0$ for $x \in \mathbb{R}$ by completing the square.

It is helpful to know that spaces in math mode are ignored by the compiler, which means we can be generous with how we use spaces to format our code such that it is easier to parse. As your expressions become more complex, the additional spaces in your code should improve its legibility. Well, at least that has been my experience. Let's consider our example without the extra spaces.

```
Solve all equations of the form \(ax^2+bx+c=0\) for
\(x\in\mathbb{R}\) by completing the square.
```

Solve all equations of the form $ax^2 + bx + c = 0$ for $x \in \mathbb{R}$ by completing the square.

The shortcut inline math environment, `\(`⟨*math environment code*⟩`\)`, that we have been using is from LaTeX. However, there is a plain TeX , shortcut for the math environment that remains popular today which is using the dollar sign character as the delimiter.

```
< normal mode text > $ < math environment code > $ < normal mode text >
```

I prefer the `\(` ⟨*math environment code*⟩ `\)` notation as it can be helpful in resolving errors. However, if you are new to using LaTeX , then you might want to use the dollar sign delimiters as they have the advantage of making your code slightly easier to read and faster to type. At this point in the development of LaTeX as a language, it is really a matter of personal preference. Just be consistent!

```
Solve all equations of the form $ ax^2 + bx + c = 0 $ for
$ x \in \mathbb{R} $ by completing the square.
```

Solve all equations of the form $ax^2 + bx + c = 0$ for $x \in \mathbb{R}$ by completing the square.

A small thought on changing between the two different shortcut notations. It would be possible to do a quick search and replace from the LaTeX notation to the TeX notation, but not vice-versa.

§2.2 Display Style Math Mode

There will be times when you will want to emphasize or isolate an expression by centering it on the page and for this we are going to be using the **display style math mode** environment. This environment is restricted to a single line of output like an equation.

```
\begin{displaymath}
  ⟨mathematics content goes here⟩
\end{displaymath}
```

```
All the real number values of \( x \) that satisfy the quadratic equations
of the form \( ax^2 + bx + c = 0 \) where \( a, b, c \in \mathbb{R} \)
can be found using the quadratic formula,
\begin{displaymath}
  x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.
\end{displaymath}
```

All the real number values of x that satisfy the quadratic equations of the form $ax^2 + bx + c = 0$ where $a, b, c \in \mathbb{R}$ can be found using the quadratic formula,

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

Just like the inline math environment, you would **ALWAYS** use the LaTeX pair of `\[` ⟨*math environment code*⟩ `\]` delimiters as the shortcut. Rather than write the delimiters and mathematical expression on the same line as we did in inline math mode, we give each delimiter it's own line and place the math on the line in between the delimiters. We could express the quadratic formula from our previous examples as:

```
\[
  x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.
\]
```

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

It is very likely that you will come across the the double dollar sign shortcut notation for the display math environment, but it should **NEVER** be used as it will most likely bring up issues at some point.

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

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

If you are looking to include multiple lines in display math mode, then you might be tempted to use consecutive display style math mode environments, but you must resist as a better solution is available. In summary, you should **NEVER** use consecutive display math mode environments, which means you should never write something like:

```
\[
  2x^2 + 14x + 24 = 2( x^2 + 7x + 12 )
\]
\[
  = 2(x + 3)(x + 3)
\].
```

$$2x^2 + 14x + 24 = 2(x^2 + 7x + 12)$$
$$= 2(x + 3)(x + 3)$$

.

§2.3 Multiline Math Environments

Like I said, there are specific environments that you can use to achieve multiline equations. This style file includes the package mathtools, which will give us multiple environments to achieve multiline equations and more. The mathtools package imports the $\mathcal{AMS}$ math package and enhances its functionality. This means if we are using mathtools, then there is no need to import $\mathcal{AMS}$ math.

For complete documentation I suggest you check the official documentation:

- User's Guide for the amsmath Package
- The mathtools package

Before we look at the using multiline equation environments, I want to point out the single line math environment called equation, which is similar to the LaTeX display math environment

```
\begin{equation}
  ⟨mathematics content goes here⟩
\end{equation}
```

The first environment that we are going to consider is very similar to the the LaTeX display math environment, but it also supports multiline equations. By default the equations will be numbered.

```
\begin{equation}
  x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.
\end{equation}
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{2.1}$$

In fact, we get something very close to the LaTeX display math environment with the star version of this equation environment as the equation numbers are suppressed. All these structured math environments have a star version that suppresses numbering, so I will only include an example for the equation environment.

```
\begin{equation*}
  ⟨mathematics content goes here⟩
\end{equation*}
```

```
\begin{equation*}
  x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.
\end{equation*}
```

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

It's time to take a look at some of the multiline equation environments that are available for us to use in our work. To change the form of the expression $2x^2+14x+24$ using the equation environment.

```
\begin{equation*}
  2x^2 + 14x + 24 = 2( x^2 + 7x + 12 ) = 2(x + 3)(x + 3)
  \end{equation*}
```

$$2x^2 + 14x + 24 = 2(x^2 + 7x + 12) = 2(x + 3)(x + 3)$$

However, this might be easier to read if it was expressed using a multiline math environment.

It is possible include a split environment within the equation environment to achieve a multiline equation with the option to include expression splitting as well.

```
\begin{split}
  ⟨mathematics content goes here⟩
\end{split}
```

We will use the & character to vertically *align* each equation at the equal sign and two consecutive double backslash characters, \\ break to the next line.

```
\begin{equation*}
  \begin{split}
    2x^2 + 14x + 24
    &= 2( x^2 + 7x + 12 ) \\
    &= 2(x + 3)(x + 3)
  \end{split}
\end{equation*}
```

$$\begin{aligned} 2x^2 + 14x + 24 &= 2(x^2 + 7x + 12) \\ &= 2(x + 3)(x + 3) \end{aligned}$$

It is even possible to split up long expressions! The \quad macro is used to include 4 spaces.

```
\begin{equation*}
  \begin{split}
    a
    &= b - c + d - e + f \\
    & \quad - g + h - i + k \\
    &= l + m \\
    &= n
  \end{split}
\end{equation*}
```

$$\begin{aligned} a &= b - c + d - e + f \\ &\quad - g + h - i + k \\ &= l + m \\ &= n \end{aligned}$$

It might be the case that you have one very long expression that you would like to break up and this can be done with the multiline environment. Notice that this does not require to be embedded in the equation environment.

```
\begin{multiline}
  ⟨mathematics content goes here⟩
\end{multiline}
```

```
\begin{multline*}
    a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + ↩
  ↪r + s + t + u + v + w + x + y + z \\
    - (a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + ↩
  ↪q + r + s + t + u + v + w + x + y + z )
\end{multline*}
```

$$\begin{multline} a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+x+y+z \\ -(a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+x+y+z) \end{multline}$$

My preference is to have my equations to be vertically aligned at the equal sign. However, you might want your equations to be centered on the page. The gather environment will take care of this for you.

```
\begin{gather}
  ⟨mathematics content goes here⟩
\end{gather}
```

```
\begin{gather*}
    a = b + c + d + e \\
    a = f + g \\
    a = h
\end{gather*}
```

$$a = b + c + d + e$$
$$a = f + g$$
$$a = h$$

Notice that there is not & character being used for alignment in this environment.

It is now time to introduce the environment that I use for about 95 % of the multiple line expressions needed in my work and it is called align.

```
\begin{align*}
  ⟨mathematics content goes here⟩
\end{align*}
```

```
\begin{align*}
  2x^2 + 14x + 24
  &= 2( x^2 + 7x + 12 ) \\
  &= 2(x + 3)(x + 3).
\end{align*}
```

$$\begin{aligned} 2x^2 + 14x + 24 &= 2(x^2 + 7x + 12) \\ &= 2(x + 3)(x + 3). \end{aligned}$$

This environment allows you to comment you work using tags

```
\begin{align*}
  2x^2 + 14x + 24
  &= 2( x^2 + 7x + 12 ) \tag{factoring two}\\
  &= 2(x + 3)(x + 3). \tag{factoring the quadratic}
\end{align*}
```

$$\begin{aligned} 2x^2 + 14x + 24 &= 2(x^2 + 7x + 12) && \text{(factoring two)} \\ &= 2(x + 3)(x + 3). && \text{(factoring the quadratic)} \end{aligned}$$

There is no need for multiple align environments in a sentence since there is the `\intertext` macro. If you want less space between the lines, then I can suggest that you use the `\shortintertext` macro instead.

```
Given \( x = \cos x \) and \( y = \sin x \), we can show
\begin{align*}
  \shortintertext{using the Pythagorean identity}
  x^2 + y^2 &= 1 \\
  \intertext{substituting \(x \) and \(y \)}
  \left( \cos x \right)^2 + \left( \sin x \right)^2 &= 1 \\
  \shortintertext{alternative function squared form}
  \cos^2 x + \sin^2 &= 1.
\end{align*}
```

Given $x = \cos x$ and $y = \sin x$, we can show

using the Pythagorean identity

$$x^2 + y^2 = 1$$

substituting x and y

$$(\cos x)^2 + (\sin x)^2 = 1$$

alternative function squared form

$$\cos^2 x + \sin^2 = 1.$$

It is also possible to include two columns of equations

```
\begin{align*}
  a_{11}
  & =b_{11}
  &
  a_{12}
  & =b_{12}\\
  a_{21}
  & =b_{21}
  &
  a_{22}
  & =b_{22}+c_{22}
\end{align*}
```

$$a_{11} = b_{11} \qquad a_{12} = b_{12}$$
$$a_{21} = b_{21} \qquad a_{22} = b_{22} + c_{22}$$

If you would like the columns to be fully justified, then there is an alternative environment called flalign.

```
\begin{flalign*}
  ⟨mathematics content goes here⟩
\end{flalign*}
```

```
\begin{flalign*}
    a_{11}
    & =b_{11}
    &
    a_{12}
    & =b_{12}\\
    a_{21}
    & =b_{21}
    &
    a_{22}
    & =b_{22}+c_{22}
\end{flalign*}
```

$$a_{11} = b_{11} \qquad a_{12} = b_{12}$$
$$a_{21} = b_{21} \qquad a_{22} = b_{22} + c_{22}$$

§2.4 Cases Environment

There are a couple cases where you might want to treat some part of your formula as a block on the same line and the cases environment get the job done.

```
\begin{cases}
  ⟨mathematics content goes here⟩
\end{cases}
```

```
\[
\abs{x}=\begin{cases}
  x & \text{ if \( x \ge 0 \) } \\
  -x & \text{ if \( x < 0 \) }
\end{cases}
\]
```

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

This can also be used for systems of equations.

```
Solve the following system of equations for \( x \) and \( y \).
  \[
  \begin{cases}
    x + y = 12 \\
    2x - 3y = 9
  \end{cases}
  \]
```

Solve the following system of equations for x and y.

$$\begin{cases} x + y = 12 \\ 2x - 3y = 9 \end{cases}$$

3 Images, Figures & Tables

§3.1 Storing Images

Images can be easily included in your document by using the `\includegraphics` command included with the graphics package. Before we include our first image, we should probably create a folder to hold all our images to keep our folder lean in terms of the number of files. My personal preference is to keep all my images in the folder called `~/assets`, which is relative to where the main file is located - the root of the project folder. If you are looking for a more refined organization of your images, then it is also possible to populate your assets folder with subfolders.

When we reference an image file, we need to provide both the path and file name. Fortunately, there is a way to assign the path(s) where you store your images, so you do not have to include the path to each image. This can be done by including the `\graphicspath` command in the preamble of the script to assign the path of where your images are stored.

```
\graphicspath{{/assets/}}
```

It is also possible to define more than one path, which you might want to do if you are using subfolders to organize your images. Just be careful to ensure all your images have unique names if you do decide to store your images in multiple folders.

```
\graphicspath{{/assets/images}{/assets/pdfs/}{/assets/matplotlib/}{/assets/tikz/}}
```

This command will save you a lot of typing and frustration, so I would encourage you to try it.

§3.2 Images

To include an image into our document, we will use the `\includegraphics` command, which accepts the path and file name as its argument and can take on multiple options. If you made use of the `\graphicspath` command made available by default for this template, then the argument is simplify the file name. The file extension is optional, unless you have multiple files with the same name, but with different extensions. Yes, `foo.pdf` are valid image files to include into your document.

```
    \includegraphics{myimage.jpg}
    \includegraphics{myimage.pdf}
```

The option that I set most often when using the `\includegraphics` command is the width of the image. For example, let's say that I want to include an image called `image.pdf` into my document such that it is centered on the page and that the width of the image is set to a quarter of the text width of the page.

```
\begin{center}
\includegraphics[width=0.25\textwidth]{image}
\end{center}
```

Just be careful when working with raster images such as `foo.jpg` as these types of images do not scale well. I try to ensure that all my images are vector based such as a `foo.pdf`, which allows for beautiful scaling.

§3.3 Figures

The XeLaTeX engine is typesetting the document for us and placing images on the page is part of that process. An inherent problem with images is that they cannot be broken across two pages like a paragraph of text. A common solution is to create a page break each time you have an image that spans over two pages. Let's leave the typesetting to the engine because let's face it, if you have many images that are spanning two pages, your document is not going to look good with all that extra whitespace created by forcing those page breaks. The solution is a floating environment called a figure to embed your image.

A float works by moving the image that does not fit on the page to a later one and fill up the white space of the current page with text. It is time to let go of implicitly referencing images by their placement of the document. We should now be referencing our images and let the engine calculate the best placement of all our floats within our document. It might feel better knowing that you can give your image a caption to provide some context as it might no longer be related to the surrounding text. Just be ready for the engine to never place the float where you would have placed it.

```
\begin{figure}[⟨!htbp⟩]
  ⟨includegraphics here⟩
\end{figure}
```

You can see that the figure environment includes some optional arguments, called placement specifiers[2], that give you some control over where the engine will place the float. The order in which the placement specifiers are written is the priority the engine will give to placing your float within your documentation. If no placement identifiers are given, then it will default to [⟨*tbp*⟩]. See table section 3.3 for a summary which each optional argument does. You will probably want to reference your image at some point in your text and will most likely want to give it a caption. For this we are going to use the figure environment, which will calculate the best position to place your image within the document. Don't worry, there are ways to manually place the image where you would like it to appear.

[⟨*h*⟩]	Places the float **here** relative to the text you have written. This should only be used for small floats.
[⟨*t*⟩]	Place the float at the **top** of the next available page.
[⟨*b*⟩]	Place the float at the **bottom** of the next available page.
[⟨*p*⟩]	Place on a **page** only containing floats.
[⟨*!*⟩]	Force one of the above options even if it does not look good.

Table 3.1: Float Placement Specifiers

```
\begin{figure}[⟨h, t⟩]
  ⟨image content goes here⟩
\end{figure}
```

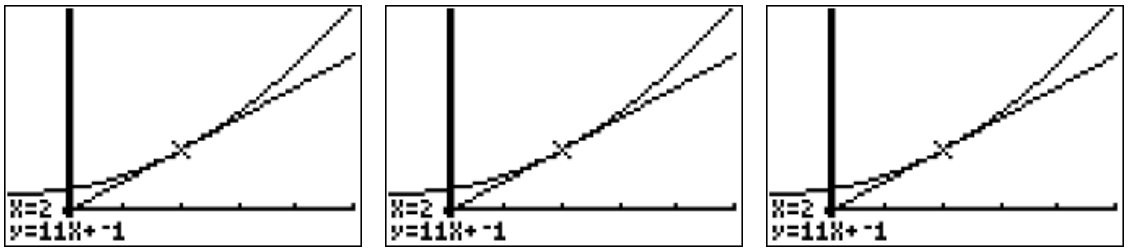
```
\begin{center}
\includegraphics[width = 0.25\textwidth]{image.jpg}
\end{center}
\caption{Here is my excellent image.}
\label{fig:img002}
```

Figure 3.1: Here is my excellent image.

This would generate figure 3.1.

§3.4 Multiple Horizontally Aligned Images

Quite often it is the case that we want to include multiple horizontally aligned images. The figure environment is not well equipped for this task, so we care going to make use of the subfigure package which will give us the subfigure environment that we can embed in our figure environment. This will enable us to assign both a caption and a label to each image separately.



(a) Step 1 (b) Step 2 (c) Step 3

Figure 3.2: The Steps Shown

```
\begin{subfigure}
  ⟨image content goes here⟩
\end{subfigure}
```

```
\begin{figure}
    \begin{center}
    \begin{subfigure}[b]{0.3\textwidth}
        \includegraphics[width=\textwidth]{20170509-123642}
        \caption{Step 1}
        \label{fig:step1a}
    \end{subfigure}
    ~   % > > > Add desired spacing between images, e. g. ~, \quad, \qquad,
              % \hfill etc. (or a blank line to force the subfigure onto
              % a new line)
    \begin{subfigure}[b]{0.3\textwidth}
        \includegraphics[width=\textwidth]{20170509-123642.png}
        \caption{Step 2}
        \label{fig:step2a}
    \end{subfigure}
    ~   % > > > Add desired spacing between images, e. g. ~, \quad, \qquad,
              % \hfill etc. (or a blank line to force the subfigure onto
              % a new line)
    \begin{subfigure}[b]{0.3\textwidth}
        \includegraphics[width=\textwidth]{20170509-123642.png}
        \caption{Step 3}
        \label{fig:step3a}
    \end{subfigure}
    \caption{The Steps Shown}\label{fig:threestepsa}
    \end{center}
\end{figure}
```

§3.5 Tables

It is very likely that you will want to include tables in your document. Tables are not always easy to generate using LaTeX; however the tabularray package makes formatting tables a little easier then some of the more popular table packages such as tabularx. Tabularray is well documented and supports tables in both text and math modes. The package and custom styles used for tables are included in the mhotables.sty custom package. This makes it easy for you to use your favorite tables package by only having to modify one line of code in the mhotext.sty. Of course, by doing so this documentation document will no longer compile as it is dependent on mhotables.sty.

```
%\RequirePackage{mhotables.sty}
\RequirePackage{tabularx}
```

The table itself is placed in a table environment, which will allow the XeLaTeX engine to calculate the best place for the table within your document very much the same way the figure environment works. It will also enable to give your table a descriptive label using the `\caption` command and give it a label so that you can reference it in your document. I usually propend tbl: to my table captions to make them easier to look up.

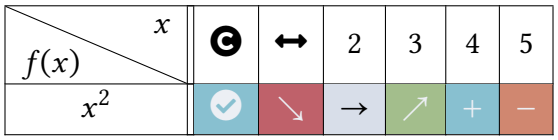
```
\begin{table}
    \begin{center}
    \begin{tblr}
    % Code to format the table
    \end{tblr}
\end{center}
\label{tbl:myfirsttable}
\caption{This table will be included in the document soon.}
\end{table}
```

```
\begin{center}
\(\begin{tblr}[
    caption = {These display fractions look awesome out of the box!},
    label = {tblr:fractions}
]{rrr}
\hline
\dfrac{2}{3} &  \dfrac{2}{3} &  \dfrac{1}{3} \\
\dfrac{2}{3} & -\dfrac{1}{3} & -\dfrac{2}{3} \\
\dfrac{1}{3} & -\dfrac{2}{3} &  \dfrac{2}{3} \\
\hline
\end{tblr} \)
\end{center}
```

$$\begin{array}{rrr}
\frac{2}{3} & \frac{2}{3} & \frac{1}{3} \\
\frac{2}{3} & -\frac{1}{3} & -\frac{2}{3} \\
\frac{1}{3} & -\frac{2}{3} & \frac{2}{3}
\end{array}$$

Another example in math mode, would be to make use of the `\diagbox` command which has been enabled by including `\UseTblrLibrarydiagbox` in the preamble of the style file. Don't worry this is included by default.

Table 3.2: hello world




```
\begin{center}
\(\begin{tblr}{|c||c|c|c|c|c|c||}
\hline
\diagbox{f(x)}{x}   &   0   &   1   &   2   &   3   &   4   &   5\\
\hline
x^2                 &   0   &   \SetCell{bg=nordEleven,fg=nordFour} \smile  ↩
  ↪&   3   &   9   &   16  &   25 \\
\hline
\end{tblr} \)
\end{center}
```

$f(x)$ \ x	0	1	2	3	4	5
x^2	0	‿	3	9	16	25

References

[1] *A Guide to the Many Flavours of TeX*. URL: https://www.overleaf.com/learn/latex/Articles/What%27s_in_a_Name%3A_A_Guide_to_the_Many_Flavours_of_TeX (visited on 07/07/2021).

[2] *CTAN: Package lshort*. URL: https://www.ctan.org/pkg/lshort (visited on 07/05/2021).

[3] *CTAN: User's Guide for the amsmath Package (Version 2.1)*. American Mathematical Society, LATEX Project, LaTeX project. 2020. URL: https://ftpmirror1.infania.net/mirror/CTAN/macros/latex/contrib/mathtools/mathtools.pdf (visited on 07/02/2021).

[4] Lars Madsen Morten Hogholm. *CTAN: The mathtools package*. LaTeX3 project. 2021. URL: https://ftpmirror1.infania.net/mirror/CTAN/macros/latex/contrib/mathtools/mathtools.pdf (visited on 07/02/2021).