

How to Reproduce this Book Exactly with \LaTeX

A Self-contained Tutorial on Writing Mathematical Notes

C. L. Loi

September 29, 2025

"How to Reproduce this Book Exactly with L^AT_EX"
Copyright ©, C. L. Loi, 2025. All rights reserved.

Contents



1	The Basic Set-up and Structure of a L^AT_EX Book	1
1.1	Class, Commands, Options, and Packages	1
1.2	Structure Hierarchy	3
1.2.1	Chapters and (Sub-)Sections	3
1.2.2	Generating Table of Contents	4
1.2.3	Organizing the T _E X Files behind the Scenes	5
1.3	Testing the Book Layout by Lipsum	5
2	Formatting of Text and Paragraphs	7
2.1	About Fonts	7
2.1.1	The Three Font Family Types	7
2.1.2	Changing the Actual Font for a Font Family	8
2.2	Text Attributes	9
2.2.1	Font Size	9
2.2.2	Font Shapes	11
2.2.3	Text Color	12
2.3	Paragraphs and Positioning	13
2.3.1	Paragraphs and Line Breaks	13
2.3.2	Justification and Indents	14
2.3.3	Lengths and Sizes	16
2.3.4	Horizontal and Vertical Spaces	17
2.3.5	Boxes and Rules	20
2.4	Verbatim Mode	22

3	The Fundamentals of Writing Mathematics in \LaTeX	25
3.1	The Two Math Modes	25
3.1.1	Inline Math Mode and Basic Math Syntax	25
3.1.2	Display Math Mode	28
3.2	Advanced Mathematical Expressions and Notations	33
3.2.1	Calculus	33
3.2.2	Logic and Description	34
3.2.3	Vectors and Matrices	36
3.2.4	Other Formatting Trivia	39
4	Various Special Structures in \LaTeX	45
4.1	Lists	45
4.1.1	Unordered Lists	45
4.1.2	Ordered Lists	46
4.2	Figures and Tables	48
4.2.1	Figures	48
4.2.2	Tables	52
4.3	Minipages and Multiple Columns	55
5	Self-defined Commands and Environments	61
5.1	Self-defined Commands	61
5.2	Flow Control	63
5.2.1	If-then-else Statements	63
5.2.2	For Loops	65
5.3	Self-defined Environments	69
6	More on Book Layout Design	71
7	Framed Theorems and Exercises	73
8	Plotting with Tikz	75
9	Miscellaneous	77

The Basic Set-up and Structure of a L^AT_EX Book

Introduction The first chapter discusses how to properly configure L^AT_EX files and organize the content structure so that we can generate our first readable L^AT_EX book PDF.

1.1 Class, Commands, Options, and Packages

Class For each L^AT_EX document, we need to specify its *class*. Throughout this book, we will use the `scrbook` class provided by the **KOMA-Script**. To do so, we write `\documentclass{scrbook}` at the very beginning (*preamble*, everything before `\begin{document}`) of the main T_EX file. Although not explored in this book, some other notable classes that may be of use include `beamer`, `moderncv`, and `article` (or `scrartcl`).

Commands and Options The `scrbook` class provides several *options* to customize the format of the book. We can either supply the arguments when declaring the class, or use the command `\KOMAoptions` in the preamble. A

command works like a function in common programming languages and performs some specific action. Commands in \LaTeX are denoted by the backslash `\` as the first character. In this book, we have used

```
\KOMAOptions{paper=a4, fontsize=12pt, chapterprefix=true, twoside=
  semi, DIV=classic, parskip=half}
```

The arguments are typed inside the curly brackets `{}` following the name of the command. Clearly, here the **paper** option requires the pages to be in A4 size while **fontsize** indicates that the font is 12 pt large. The remaining options will be explained as we go through the later chapters.

Packages To enable extra functionalities, we need to import *packages*. We can write along the lines of `\usepackage[<options>]{<package_name>}` in the preamble to do so. We will not list all the required packages now at once, but only when they are needed. The first package we usually need is the **fontenc** package with the **T1** option, flagged inside a pair of square brackets.

Exercise(s)

1.1) Try to import the **fontenc** package with the **T1** option as suggested above. There may not be any noticeable difference, but at least you should not be receiving errors.

1.2) Also, try to achieve the same class setting through the `\documentclass[<options>]{scrbook}` declaration instead of the `\KOMAOptions` command.

1.2 Structure Hierarchy

1.2.1 Chapters and (Sub-)Sections

Chapters, Sections In most of the books, the entire content is divided into *chapters*, which in turn usually consist of several *sections*. To mark the beginning of a chapter or section in \LaTeX , we place the commands `\chapter{<chapter_name>}` or `\section{<section_name>}` within the `document` environment, which contains the main content and is marked by a pair of `\begin{document}` and `\end{document}` declarations. As mentioned in the beginning, the preamble has to be inserted before such a `document` group. So, to typeset the very first section at the start, we write

```
% <preamble before the main document>
\begin{document}
...
\chapter{The Basic Set-up and Structure of a \LaTeX{} Book}
...
\section{Class, Options, and Packages}
\paragraph{Class}
For each \LaTeX{} document, we need to specify its \textit{class}.
    Throughout this book, ...
...
\end{document}
```

The `%` symbol indicates a trailing *comment* (highlighted in green) whose purpose is to leave some note about the code. Comments are neither interpreted nor displayed. The \LaTeX system records and updates the numbering for chapters/sections internally (More in Chapter ?? about counters). The `\textit{<text>}` command presents the text in italic shape.

Subsections, Paragraphs Attentive readers may have already figured out that it is possible to stack an extra level (a *subsection*) in the content hierarchy. This is

aptly done not long ago by the `\subsection{<section_name>}` command:

```
\section{Structure Hierarchy}

\subsection{Chapters and (Sub-)Sections}

\paragraph{Chapters, Sections}
As in any other book, the entire content is divided into \textit{chapters}, ...
```

He/she may also notice that we have used the `\paragraph` command a few times to attach an unnumbered heading for each *paragraph*. There are also starred versions like `\chapter*{<chapter_name>}`, `\section*{<section_name>}`, `\subsection*{<section_name>}`, and so on, which neither display nor increment the numbering/counters.

1.2.2 Generating Table of Contents

Table of Contents After establishing the structure of the book, it is convenient to generate a *table of contents (TOC)* as well. In the `scrbook` class, it is easily done by adding the command `\tableofcontents` within the main `document` environment. To control the depth of layers shown, we can call `\setcountertocdepth{<integer>}` in the preamble, where the `integer` usually ranges from -1 to 3 (0: chapters, 1: sections, 2: subsections).

Exercise(s)

1.3) Try to add some (numbered or unnumbered) chapters, sections, subsections, or even subsubsections (which are, not surprisingly, produced by `\subsubsection`) to see how they are displayed in the book. You may want to check out `\part`.

1.4) As a follow-up to the last exercise, turn on the table of contents and confirm how the new entries are linked to it. Also, try to adjust the value for `\setcountertocdepth` as proposed above to see the effect.

1.2.3 Organizing the T_EX Files behind the Scenes

include As the size of the project scales up, it is often helpful to keep the files sorted in a clean order for maintenance. We can put the content of each chapter into separate T_EX files, and then use the `\include{<tex_file_name>}` command to import them into the main script. For example, this chapter is stored as `ch1_basic_structure.tex` in my project space, and in the main T_EX file, we shall write something like

```
% <preamble again>
\begin{document}

\tableofcontents
\include{ch1_basic_structure}
...
\end{document}
```

1.3 Testing the Book Layout by Lipsum

Dummy Text Sometimes we may need to insert some placeholder text into the document to test how well the book will look in a specific layout. In this case, we can borrow the standard dummy text *Lorem Ipsum* (or in short *Lipsum*) widely used by the community. Just import the **lipsum** generator package, and add `\lipsum[<paragraph_no.>]` to the desired locations. For example, the code segment

...
produces the following text exactly: `\par`
`\lipsum[1-2]`

produces the following text exactly:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

The `\par` command signals the end of a paragraph and appends a vertical line spacing afterwards.

Formatting of Text and Paragraphs

Introduction This chapter describes how to adjust the various aspects of text, such as fonts, size/shape/style, and positioning.

2.1 About Fonts

2.1.1 The Three Font Family Types

(Sans) Serif, Typewriter In any \LaTeX document, the text can be typed in three different *font families*: *serif*, *sans serif*, and *typewriter*. In this book, headings (of chapters, sections, etc.) are in the sans serif family, while the remaining main text is in serif. Table 2.1 below demonstrates how to select a specific font family for a piece of text. For instance, both

```
...  
produces the following output: \par  
\textsf{\lipsum[3]} % or {\sffamily \lipsum[3]}, mind the curly  
brackets {} to limit the scope of the \sffamily command.
```

Font Family	Command	Switch	Output
Serif	<code>\textrm{Hello World!}</code>	<code>\rmfamily</code>	Hello World!
Sans Serif	<code>\textsf{Hello World!}</code>	<code>\sffamily</code>	Hello World!
Typewriter	<code>\texttt{Hello World!}</code>	<code>\ttfamily</code>	Hello World!

Table 2.1: The commands for switching between the three font families and how they appear in the PDF.

produces the following output:

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

2.1.2 Changing the Actual Font for a Font Family

Font Libraries Each of the previous font families is internally assigned a specific *font*. To change the actual font, we can call the corresponding font package. The **LaTeX Font Catalogue** <https://tug.org/FontCatalogue/> provides a comprehensive list of available fonts and the way to import them. This book has substituted the **Noto Sans** font for the sans serif family, via the preamble

```
\usepackage[T1]{fontenc}
\usepackage[sf]{noto}
```

Exercise(s)

2.1) Change the font family just for the dummy Lipsum paragraph above to typewriter.

2.2) Choose a font of your liking from the Font Catalogue to replace the original one in the book.

2.2 Text Attributes

2.2.1 Font Size

Size Commands In Section 1.1, we talked about setting the base global font size by `\KOMAOPTIONS`. However, to control the *local font size* for some places, we can use the *size commands*, listed in Table 2.2 below. For example, writing

```
... produces \par
{\small Though she be but little} {\LARGE she is fierce} \\ % scope
\scriptsize % switch
taken from Shakespeare's A Midsummer Night's Dream
\normalsize % back to default ...
```

produces

Though she be but little she is fierce

taken from Shakespeare's A Midsummer Night's Dream

The `\\` sign breaks the current line and starts a new line right below. And again, the curly brackets `{ }` limit the effect of command(s) within their scope.

Table 2.2: The various commands for text size.¹

Command	Output
<code>\tiny</code>	Who am I?
<code>\scriptsize</code>	Who am I?
<code>\footnotesize</code>	Who am I?
<code>\small</code>	Who am I?
<code>\normalsize</code>	Who am I?
<code>\large</code>	Who am I?
<code>\Large</code>	Who am I?
<code>\LARGE</code>	Who am I?
<code>\huge</code>	Who am I?
<code>\Huge</code>	Who am I?

selectfont It is also possible to fix a numerical value for the font size using `\fontsize{<font_size>}{<line_spacing>}` and `\selectfont`. As an illustration, the code

```
... leads to \par
{\fontsize{15pt}{21pt}\selectfont May those who accept their fate be
  granted happiness. May those who defy their fate be granted glory.
  \\\
-- Princess Tutu \par} % the \par is needed for renewing the
  parameters
```

leads to

¹`\huge` and `\Huge` have the same size when the font size is 12 pt (but different for 10 or 11 pt).

Font Style	Command	Switch	Output
Bold	<code>\textbf{"10 Downing"}</code>	<code>\bfseries</code>	"10 Downing"
Medium	<code>\textmd{"10 Downing"}</code>	<code>\mdseries</code>	"10 Downing"
Italic	<code>\textit{"10 Downing"}</code>	<code>\itshape</code>	<i>"10 Downing"</i>
Slanted	<code>\textsl{"10 Downing"}</code>	<code>\slshape</code>	<i>"10 Downing"</i>
Small Caps	<code>\textsc{"10 Downing"}</code>	<code>\scshape</code>	"10 DOWNING"
Upright	<code>\textup{"10 Downing"}</code>	<code>\upshape</code>	"10 Downing"

Table 2.3: The commands for different font styles. The medium/upright style is effectively the default normal.

May those who accept their fate be granted happiness. May those who defy their fate be granted glory.

– Princess Tutu

2.2.2 Font Shapes

Italic, Bold, and More Similar to font families, there are different *font shapes/styles* such as the commonly seen italic or bold. Table 2.3 above shows the relevant commands to invoke them. Adding to the previous example, we can write

```
... which produces \par
\textit{\small Though she be but little} {\LARGE \bfseries \scshape
  she is fierce} \\ % scope
\scriptsize % switch
taken from \slshape \underline{Shakespeare's A Midsummer Night's
  Dream}
\normalsize \upshape \par % back to default
...
```

which produces

Though she be but little **SHE IS FIERCE**

taken from Shakespeare's A Midsummer Night's Dream

We also have `\underline` and `\emph`. You may want to try them out.

2.2.3 Text Color

xcolor While there are built-in colors in the L^AT_EX system, we can load a variety of additional colors from the **xcolor** package, often with the **svgnames** and **dvipsnames** flags as

```
\usepackage[svgnames, dvipsnames]{xcolor}
```

The reference color list can be found in https://www.overleaf.com/learn/latex/Using_colors_in_LaTeX. To set the color for a piece of text, we can enclose it with the `\textcolor{<color_name>}{<text>}` command. It is also possible to change the color within a group by `\color{<color_name>}`. For instance,

```
... outputs \par
\textcolor{Red}{Roses are red,} \\\
\textcolor{Blue}{violets are blue,} \\\
{\color{Purple} Sugar is sweet and so are you.} % again, remember to
    limit the scope by the curly brackets!
```

outputs

Roses are red,
violets are blue,
Sugar is sweet and so are you.

Self-defined colors It is also possible to design a custom color by the command `\definecolor{<color_name>}{<color_model>}{<values>}`. There are 4 possible color models: **rgb**, **RGB**, **cmyk**, and **gray**. For example,


```
...  
\definecolor{mint}{rgb}{0.24, 0.71, 0.54} % in the preamble  
...  
\textcolor{mint}{Mint Tears}
```

gives Mint Tears. Color codes can be checked via <https://latexcolor.com/>.

In addition, we can mix colors by the expression `<color_1>!<mix_ratio>!<color_2>`. For instance,

```
\textcolor{Blue!40!Green}{Copper (II)} \textcolor{Orange!50}{Sulphate  
}
```

is displayed as Copper (II) Sulphate.

2.3 Paragraphs and Positioning

2.3.1 Paragraphs and Line Breaks

New Lines As explained before, the `\` symbol issues a *line break*, and the `\par` command ends a paragraph and starts a new one.

Both of them initiate a *new line*, but with (without) an extra *line skip/line spacing* for `\par` (`\`). There is also `\newline` which is seldom used.

A blank line in a \TeX file has the same effect as `\par`. They, in fact, end the so-called *horizontal mode* and distribute the text into lines held in the current vertical list (see \TeX StackExchange 82664).

The effects of `\`, `\par`, and blank lines can be observed right in this subsection, which is typed as

```
\paragraph{New Lines}  
As explained before, ... ends a paragraph and starts a new one. \  
Both of them initiate a \textit{new line}, ... which is seldom used.  
% here is a blank line with this comment only
```

```
A blank line in a \TeX{} file ... held in the current vertical list (
  see ...). \par
The effects of \texttt{\textbackslash\textbackslash}, \texttt{\
  \textbackslash par}, and blank lines can be observed right in this
  subsection, which is typed as
... % this code block
```

2.3.2 Justification and Indents

raggedleft/right, centering The `\raggedright` and `\raggedleft` commands produce *left/right-justified* text respectively. As you may have figured out, this paragraph is "*ragged right*" (although not very obvious, notice \rightarrow) so that the text sticks to the left boundary, but the right side is now uneven.

Meanwhile, this lipsum text is "*ragged left*": Quisque ullamcorper placerat ipsum.

Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl.

Vivamus quis tortor vitae risus porta vehicula.

The default setting is *fully-justified* so that the text extends to both edges like this one. `\raggedleft` and `\raggedright` act like a switch, changing all paragraphs beyond, and we may want to put them within a group enclosed by curly brackets.

We also have `\centering` which is quite self-explanatory and is demonstrated here. For these three commands to work properly, we require `\par` to finish, similar to before. The code to generate the above paragraphs is

```
\paragraph{Raggedleft/right, Centering}
```

```
{\raggedright The \texttt{\textbackslash raggedright} and \texttt{\textbackslash raggedleft} commands produce ... but the right side is now uneven. \par}
{\raggedleft Meanwhile, this lipsum text is "ragged left": \lipsum[4]\par}
The default setting is \textit{fully-justified} ... we may want to put them within a group enclosed by curly brackets. \par
{\centering We also have \texttt{\textbackslash centering} ... The code to generate the above paragraphs is \par} ...
```

flushleft/right, center (Environments) The alternative to the above is to put the text into a `flushleft`/`flushright`/`center` environment. An *environment* contains content that is to be processed and displayed according to the specific design indicated by the environment itself. Environments always start with the `\begin{<env_name>}` and end with the `\end{<env_name>}` statements. For example, the previous part can also be reproduced by

```
...
\begin{flushright}
Meanwhile, this lipsum text is "ragged left": \lipsum[4]
\end{flushright}
...
\begin{center}
We also have \texttt{\textbackslash centering} ... The code to generate the above paragraphs is
\end{center}
```

`flushleft` (`flushright`) corresponds to (`\raggedright`) `\raggedleft`. If you test this new code, notice the increased separation² around the environments.

Indents, parskip Attentive readers may have figured out that there is no *indent* for paragraphs in the book, and they are only separated by a slight vertical spacing.

²This is dictated by `\topsep`, see the next subsection.

This is controlled by the `parskip=half` value inside `\KOMAOPTIONS` in the preamble, which means that paragraphs are identified with a vertical spacing of half a line. The two other options `parskip=no` and `parskip=full` use indents (without vertical spacing) and one full line instead.

Also, we can control indents manually by adding `\indent`³ or `\noindent` to the start of paragraphs.

microtype Finally, for a better typesetting behavior (e.g. hyphenation), it is recommended to always import the **microtype** package, which provides helpful patches on this.

Exercise(s)

2.3) Try experimenting with different `parskip` options (there are additional modifiers like `half-`, `half+`, `half*`, similar for `full`) for the KOMA-script class, as well as the on-and-off of indents.

2.3.3 Lengths and Sizes

Length Units Before learning how to adjust the size of objects and spacing, we need to be able to express and measure lengths in \LaTeX . There are various *length units* for this, summarized in Table 2.4 below.

Length Values, setlength Subsequently, the *lengths* of different markers are stored as parameters, listed in Table 2.5. By using `\setlength{<length_param>}{<length_value>}`, we can modify them and adjust distances on the page.

³It will not work if `parskip` is `half` or `full`.

Unit	Description
pt	The usual "point" unit adopted in other documenting software.
mm/cm/in	A millimeter/A centimeter/An inch.
ex	The height of a lowercase "x" character in the current font. (usually used for vertical distance)
em	The width of an uppercase "M" in the current font. (usually used for horizontal distance)
mu	1/18 of an em with respect to the math symbols. (usually used in math mode)

Table 2.4: The various length units in L^AT_EX.

2.3.4 Horizontal and Vertical Spaces

hspace, vspace To control the position of different objects or blocks, the primary way is via the `\hspace{<length>}` and `\vspace{<length>}` commands. As their names hint, they add a horizontal/vertical space of fixed lengths. For example, the code

```
\hspace{3ex} Hello \hspace{5ex} World \vspace{1.5em} !!! \\
Ouch...
```

gives

Hello World !!!

Ouch...

The first two **hspace** commands should work as you have expected, but notice that on the other hand, **vspace** in the middle of a line will only take effect after it, and so the exclamation marks above are not moved down (but "Ouch..." is). Finally, they accept negative lengths, and you may want to play with that.

Parameter	Description
<code>\baselineskip</code>	Vertical distance between adjacent lines within a paragraph.
<code>\columnsep</code>	Distance between columns.
<code>\columnwidth</code>	The width of a column.
<code>\fboxsep</code> and <code>\fboxrule</code>	The padding and line width around boxes.
<code>\linewidth</code>	The width of a line.
<code>\paperheight</code> and <code>\paperwidth</code>	The height and width of the page.
<code>\parindent</code>	The length of the indent before a paragraph.
<code>\parskip</code>	The vertical spacing between paragraphs.
<code>\textheight</code> and <code>\textwidth</code>	The height and width of the text area in a page.
<code>\topmargin</code>	The length of the top margin.
<code>\topsep</code> and <code>\itemsep</code>	The vertical space added above and below an environment, as well as around the items within it.

Table 2.5: Commonly involved length parameters in L^AT_EX.

It is also to achieve the same effect after a line break by writing something along the lines of `\[\<length>]`, e.g.

```
Don't come any closer!!!\[-1em]
Nope *Taking out the axe*
```

```
Don't come any closer!!!
Nope *Taking out the axe*
```

hspace*, vspace* There also exist starred versions of `\hspace*{<length>}` and `\vspace*{<length>}`. The difference is that the original ones will be "gobbled up" (see T_EX StackExchange 89082) and disappear at line breaks, but

the new ones will not. To see this clearly, let's try

```
x\hspace{3ex}y\\
\hspace{4ex}y?\
\hspace*{4ex}y!
```

which gives

```
x    y
y?
    y!
```

hfill, vfill, fill, stretch In the case where a fixed distance is only needed in a certain place, while other remaining empty spaces can extend automatically, we can make use of the `\hfill`, `\vfill` commands, or more generally `\fill`, plus `\stretch{<factor>}`. `\hfill` and `\vfill` will take up all the possible spaces after other `hspace` or `vspace` commands are calculated.

If there are multiple `\hfill` or `\vfill`, then the length will be partitioned equally. To assign different weightings to the partition, we can go back and write `\hspace{\stretch{<factor>}}` (similarly for `\vspace`). For example,

```
\hfill Hope \hspace{4cm} Faith \hspace*{\stretch{2}} \\
\hspace*{\stretch{2}} Love \hspace{4cm} Luck \hspace*{\fill} \par %
    the asterisks * are needed!
```

yields

```

                Hope                Faith
                Love                Luck
```

Notice how we have to use the starred forms to circumvent the gobbling. (Try not using them and see how it fails!)

smallskip, medskip, bigskip Finally, there are also shorthands for generating vertical line skips: `\smallskip`, `\medskip`, and `\bigskip`. Note that they are just `\vspace` with `\smallskipamount`, `\medskipamount`, and `\bigskipamount` under the hood.

2.3.5 Boxes and Rules

mbox, fbox By calling `\mbox{<text>}`, a piece of text may be placed and contained inside a *horizontal box*. This also means that the text will not be disrupted by automatic line breaks or stretched (see [TeX StackExchange 475056](#)), and can spill out of the main area into the margin. There is also `\fbox{<text>}` as a wrapped version of `\mbox` with a frame around it, and we will use it for a visualized comparison: The code

```
Preparation is the key to success, but a good plan today is better
than a perfect plan tomorrow.
\fbbox{Preparation is the key to success, but a good plan today is
better than a perfect plan tomorrow.}
```

produces: Preparation is the key to success, but a good plan today is better than a perfect plan tomorrow. Preparation is the key to success, but a good plan today is better than a perfect plan tomorrow.

From this, we can clearly see how the horizontal box extends all the way outside.

makebox, framebox An improved version for the box commands above consists of `\makebox[<width>][<alignment>]{<text>}` and also similarly `\framebox[<width>][<alignment>]{<text>}`, where we can specify the width of the box and how the text inside is justified (**l**, **c**, **r**, **s**: left, center, right, spread) inside the box. For example,

```
\framebox[100pt][c]{I fit inside!} and \
\framebox[130pt][l]{Unfortunately, this one is too small for me...}
```


generates I fit inside! and

Unfortunately, this one is too small for me...

These box commands can be manipulated to control the distribution of text.

parbox Meanwhile, *vertical boxes* where the text inside can break just like normally can be constructed by the `\parbox[<alignment>]{<width>}{<text>}` command. The effect is not hard to inspect from the input

that produces `\parbox[b]{100pt}{Empty your mind, be formless,
shapeless, like water.} ...`

Empty your mind,
be formless, shape-

that produces less, like water. This time, the alignment option (**t**, **c**, **b**: top, center, bottom) decides how the `\parbox` will be positioned relative to the current line. To add a frame around it, simply enclose it with an extra `\fbox`.

raisebox Sometimes we may want to raise or lower a text while pretending it still occupies some space with a fixed extent. Then the `\raisebox{<vertical_distance>}[<extend_above>][<extend_below>]{<text>}` command will do the job. This is demonstrated by including a `\fbox` to visualize the effect:

`\fbox{\raisebox{15pt}[10pt][10pt]{I am a rising star!}}` and this is my stage!

I am a rising star!
 and this is my stage!

This command can be very useful in achieving several invisible spacing tricks.

Rules Another useful ingredient is the possibility to draw *rules* as lines. The basic command is `\rule{<horizontal_extent>}{<vertical_extent>}`. For example, `\rule{5ex}{1ex}` generates this: . We also have more primitive versions of `\hrule` and `\vrule`. The code below will yield

```
\vrule \hspace{6pt} If you remove me, the vertical rule to the left  
will disappear! \hrule
```

| If you remove me, the vertical rule to the left will disappear!

Exercise(s)

2.4) Use the `\setlength` command to change different lengths and test what the result would look like, e.g. `\setlength{\parindent}{5cm}`.

2.5) Copy your favorite quote or paragraph to the document, and use the commands/techniques introduced in these two sections to make it beautiful and stylish.

2.4 Verbatim Mode

verb To type short inline code pieces, we can use the *verbatim* mode through the `\verb|<content>|` command. This preserves the input exactly as it is typed, without invoking any would-be \LaTeX command or special character. For example, entering `\verb|func|` will output **func** here. However, a major pitfall is that `\verb` can fail when it is used inside the argument of a command.⁴ Since we may use the `\include` command to import each chapter separately as suggested by Section 1.2.3, this will be problematic. An alternative is to use `\texttt{<content>}`, with `\textbackslash` as the replacement for `\`, and writing `_` for `_`, `\{` and `\}` for `{` and `}`.

lstlisting When we need to display larger blocks of code, we can use the `listings` package and its `lstlisting` environment. Actually, it has already

⁴Interested readers can search fragile commands.

been used (shown as yellow areas) in this book many times. A self-explanatory example⁵ is

```
\begin{lstlisting} % can pass the overriding option [style=<style_
    name>]
\textit{I guess this counts as a recursion...}
\end{lstlisting}
```

To design the appearance of the code blocks, we can define our own `lstlisting` style. The one adopted in the book is given by

```
\lstdefinestyle{lstTeXstyle}{ % give a name for the lstlisting style
    language=[latex]TeX,
    basicstyle=\footnotesize\ttfamily, % the font style
    backgroundcolor=\color{Goldenrod!20},
    keywordstyle=\color{blue!80}\bfseries, % for highlighting
        functions
    commentstyle=\color{Green},
    breaklines=true,
    numbers=none, % none, left, or right
    showstringspaces=false,
    belowskip=0pt}
\lstset{style=lstTeXstyle} % set the style
```

Most of the options above are not hard to understand, but you may want to fiddle with the last four of them.

Like `\verb`, this package also comes up with `\lstinline` for writing inline code.

Exercise(s)

2.6) Take any of the code blocks in this book and reproduce it using the `lstlisting` environment.

⁵It is a bit involved to make this one work, the option `escapeinside` (as well as its variants) is intentionally left out below, but you should look it up.

The Fundamentals of Writing Mathematics in \LaTeX

Introduction This chapter covers the basic methods regarding how to typeset and align different mathematical expressions and formulae in \LaTeX .

3.1 The Two Math Modes

3.1.1 Inline Math Mode and Basic Math Syntax

Inline Math by $\text{\$}$ To be able to write mathematical expressions in \LaTeX , we need to first enter the so-called *math mode*. There are two types of math mode in \LaTeX , and the simpler one will be the *inline* math mode. As its name suggests, it renders the mathematical expressions as a usual part of a paragraph. We can enter the inline mode by enclosing an expression with the dollar signs like $\text{\$}<\text{expression}>\text{\$}$. For example, typing $\text{\$}3x+4y-z = 5\text{\$}$ here readily outputs $3x + 4y - z = 5$.

Basic Operators The plus, minus, divide, and equal signs $+$, $-$, $/$, $=$ are just the usual ones and can be typed directly in math mode. Meanwhile, the multiplication sign (\times) has to be typed explicitly as `\times`, and we may also use the dot sign (\cdot) through `\cdot` instead. Round and square brackets in math mode are also simply given by `()`, `[]`.

Superscripts and Subscripts Superscripts (e.g. raising to a power) and subscripts can be added via `^{\<superscript>}` and `_{\<subscript>}`. For example, `C^{\<2n>}_{\<n>}` is output as C_n^{2n} .

Fractions, smash Fractions can be typeset easily as `\frac{\<numerator>}{\<denominator>}`, e.g. `\frac{2x^2}{3x+1}` produces $\frac{2x^2}{3x+1}$. However, notice that this `\frac` in the inline mode is shrunk. One workaround is to simply use the slash `/` instead, but we can also replace `\frac` by `\dfrac`, which gives $\frac{2x^2}{3x+1}$. Unfortunately, this leads to another issue where the full-size fraction interferes with the line spacing (the lines directly above and below the `\dfrac` are slightly pushed away if you look closely). A quick fix is to enclose it with the `\smash{}` command to tell L^AT_EX to ignore its extent.

Common Mathematical Functions, Symbols The commands for some notable, frequently used mathematical functions and symbols are summarized in Table 3.1 below.

3.1 The Two Math Modes

Function/Symbol(s)	Command(s)	Description
$\sin, \cos, \tan, \csc, \sec, \cot$	<code>\sin()</code> , <code>\cos()</code> , <code>\tan()</code> , <code>\csc()</code> , <code>\sec()</code> , <code>\cot()</code>	Trigonometric Functions.
\exp, \log, \ln	<code>\exp()</code> , <code>\log()</code> , <code>\ln()</code>	Exponential and (Natural) Logarithm.
$\sqrt{x}, \sqrt[3]{x}$	<code>\sqrt{x}</code> , <code>\sqrt[3]{x}</code>	Square (Cubic) Root of x .
i, e, π	<code>i</code> , <code>e</code> , <code>\pi</code>	Important constants: The imaginary number, e , and π .
$\alpha, \beta, \gamma, \dots$	<code>\alpha</code> , <code>\beta</code> , <code>\gamma</code>	Greek letters. (see the full list at http://www.phys.uri.edu/~nigh/TeX/sym1.html)
\pm, ∞	<code>\pm</code> , <code>\infty</code>	The plus/minus sign and infinity symbol.
\sum_i^n, \int_a^b	<code>\sum_{i}^n</code> , <code>\int_{a}^b</code>	Summation and integral signs with lower and upper limits.

Table 3.1: Commonly used mathematical commands in L^AT_EX.

Exercise(s)

3.1) Try to reproduce the following mathematical expressions.

a) $ax^2 + by^2 + c(z - 4)^2 = R^2;$

b) $g(x) = \frac{1}{e^{-qx} + 1};$

c) $A_{ij}^2 = A_{ik}A_{kj};$

d) $\int_0^\infty \frac{\sin(\pi x)}{x} dx = ?;^1$

e) $\beta \pm \ln\left(\sqrt{\frac{\alpha}{10}}\right)i.$

3.1.2 Display Math Mode

equation The second type of math mode is the *display* math mode, which involves putting the expressions inside an environment on their own. The most frequently used one is the **equation** group, which processes a single line of equation or formula. For instance,

```
\begin{equation}
f(t) = 1 - e^{-at}
\end{equation}
```

results in

$$f(t) = 1 - e^{-at} \quad (3.1)$$

Notice that the **equation** is automatically numbered.

align More often than not, we want to show the detailed steps involved in a calculation. The **align** environment enables us to write them in multiple lines, in addition to providing the **&** character as the anchor for aligning these lines. The **** symbol is again used as a line break just like in any ordinary text. As an example,

```
\begin{align}
&\frac{d}{dx}(2x+3)^5 = {}& [5(2x+3)^4] \left[ \frac{d}{dx}(2x+3) \right] & \& \text{(Chain Rule)} \\
&= {}& [5(2x+3)^4](2) = 10(2x+3)^4
\end{align}
```

will give

$$\frac{d}{dx}(2x+3)^5 = [5(2x+3)^4] \left[\frac{d}{dx}(2x+3) \right] \quad (\text{Chain Rule}) \quad (3.2)$$

$$= [5(2x+3)^4](2) = 10(2x+3)^4 \quad (3.3)$$

¹To the curious readers, the result is equal to $\pi/2$.

There are some points worth mentioning. First, the **align** environment will create a line number for each individual line by default. Second, the two lines above are aligned via the first **&** character in them, as expected. Third, by adding some extra **&**, we can append any note to the right. In fact, odd-numbered **&** control the exact alignment positions and even-numbered **&** dictate the partition of pieces. Finally, the **{}** after **=** are needed for appropriate spacing (try removing them!).

split Sometimes, an entire expression is too long to be captured in a single line and may require us to break it into multiple lines, while still treating it as a whole entity. The **split** sub-environment then comes in handy. It works like **align** but can be embedded in another larger **align** group. For example,

```
\begin{align}
(2x+3)^5 ={}& \sum_{k=0}^5 C^5_k (2x)^k(3)^{5-k}\\
\begin{split}
= {}& 32x^5 + 240x^4 + 720x^3 \\
& + 1080x^2 + 810x + 243
\end{split}
\end{split}
\end{align}
```

produces

$$(2x + 3)^5 = \sum_{k=0}^5 C_k^5 (2x)^k (3)^{5-k} \quad (3.4)$$

$$= 32x^5 + 240x^4 + 720x^3 + 1080x^2 + 810x + 243 \quad (3.5)$$

As you can see, **split** only occupies a single equation number (in the middle) and the **&** inside it can "communicate" with those outside **split**.

aligned On the contrary, we have the related **aligned**, and the readers can try (strongly recommended as an exercise)

```
\begin{align}
(2x+3)^5 = {}& \sum_{k=0}^5 C^5_k (2x)^k (3)^{5-k} \\
= {}& \\
\begin{aligned}
& 32x^5 + 240x^4 + 720x^3 \\
& + 1080x^2 + 810x + 243
\end{aligned}
\end{align}
```

to see the difference (particularly the `&`). There is also `multline`, however, most of the usages are already covered by `split` and `aligned`, so we will not discuss it.

Starred Equations Sometimes, the equations may not be worthy of assigning an equation number. By using the starred versions of these environments (`equation*`, `align*`, etc.), the equation numbers will be suppressed. For example,

```
\begin{equation*}
1 + 1 = 2
\end{equation*}
```

yields

$$1 + 1 = 2$$

A quick alternative is to use the `\[<math>\]` shorthand.

nonumber We can also use `\nonumber` to manually prevent numbering for any line. For instance,

```
\begin{align}
\int xe^{-x} dx &= -\int x d(e^{-x}) \nonumber \\
&= -[xe^{-x}] + \int e^{-x} dx && (\text{Integration by Parts}) \nonumber \\
&= -xe^{-x} - e^{-x} + C
\end{align}
```

will give

$$\begin{aligned}
 \int x e^{-x} dx &= - \int x d(e^{-x}) \\
 &= -[x e^{-x}] + \int e^{-x} dx && \text{(Integration by Parts)} \\
 &= -x e^{-x} - e^{-x} + C
 \end{aligned} \tag{3.6}$$

Equation Numbers Referencing From time to time, we may need to refer to previous equations during the derivation of a new one. This is straightforward if the equations are numbered, where we can explicitly attach a *label* to the specific lines by `\label{<name>}`. Subsequently, we can call the equation numbers by `\ref{<name>}`. To demonstrate, we may update the integration by parts example in the above paragraph:

```
...
&= -xe^{-x} - e^{-x} + C \label{eqn:IBP1}
```

then `(\ref{eqn:IBP1})` will properly return (3.6).

It is also possible to achieve letter numbering in the **subequations** mode, e.g.

```
\begin{subequations}
\begin{align}
\cos (2x) &= \cos^2 x - \sin^2 x \\
\sin (2x) &= 2 \sin x \cos x
\end{align}
\end{subequations}
```

will generate

$$\cos(2x) = \cos^2 x - \sin^2 x \tag{3.7a}$$

$$\sin(2x) = 2 \sin x \cos x \tag{3.7b}$$

allowdisplaybreaks When we are using the `align` environment (or other similar ones), the blocks may become too lengthy to be included in a single page. By appending the switch `\allowdisplaybreaks` (in the preamble), the L^AT_EX system will then be allowed to break them across multiple pages. This may or may not be desirable and will depend on the situation. As a side note, if an inline expression in a paragraph is too long and hangs outside the main text area, we may add the command `\allowbreak` so that a line break may be inserted there.

Exercise(s)

3.2) Try to reproduce the paragraphs below with the numbered equations. Notice that the enlarged brackets can be obtained by `\left(<math>\right)`. Solving

$$x^2 \frac{d^2 y}{dx^2} - 3x \frac{dy}{dx} + 3y = 0 \quad (3.8)$$

Let $z = \ln x$, then

$$\frac{dy}{dx} = \frac{dy}{dz} \frac{dz}{dx} = \frac{1}{x} \frac{dy}{dz} \quad (3.9)$$

$$\frac{d^2 y}{dx^2} = \frac{d}{dx} \left(\frac{dy}{dx} \right) = \frac{d}{dx} \left(\frac{1}{x} \frac{dy}{dz} \right) \quad (\text{continuing from (3.9)})$$

$$\begin{aligned} &= \frac{1}{x} \frac{d}{dx} \left(\frac{dy}{dz} \right) - \frac{1}{x^2} \frac{dy}{dz} \\ &= \frac{1}{x} \frac{dz}{dx} \frac{d}{dz} \left(\frac{dy}{dz} \right) - \frac{1}{x^2} \frac{dy}{dz} \\ &= \frac{1}{x^2} \frac{d^2 y}{dz^2} - \frac{1}{x^2} \frac{dy}{dz} \end{aligned} \quad (3.10)$$

Substituting (3.9) and (3.10) into (3.8), we have ...

Advanced Mathematical Expressions and 3.2 Notations

amsmath, amssymb, mathtools Before getting into the main section, it is necessary to load the prerequisite **amsmath**, **amssymb**, and **mathtools** packages for the symbols, as well as enhancing the mathematical typesetting.

3.2.1 Calculus

Differentiation and Integral Symbols Table 3.2 below is a list of notable symbols used to denote derivatives and integrals for calculus, aside from Table 3.1.

Function/Symbol(s)	Command(s)	Description
∂, ∂_x	<code>\partial</code> , <code>\partial_x</code>	Partial derivatives (with respect to x).
∇, Δ	<code>\nabla</code> , <code>\Delta</code>	The del and Laplacian operators.
$\lim_{x \rightarrow 0}$, \liminf , \limsup	<code>\lim_{x \rightarrow 0}</code> , <code>\liminf</code> , <code>\limsup</code>	Limit (inferior and superior).
\iint_S, \iiint, \oint	<code>\iint_S</code> , <code>\iiint</code> , <code>\oint</code>	Double, triple ² , and contour integrals.

Table 3.2: Commonly used differentiation and integral symbols.

²If the limits of the multiple integral have to be spelled out explicitly, then just resort to using the original `\int_{...}^{...}` for multiple times.

3.2.2 Logic and Description

Logic and Set Symbols Meanwhile, Table 3.3 below contains a number of commonly used logical operators and set symbols.

Function/Symbol(s)	Command(s)	Description
$<, >, \leq, \geq, \ll, \gg$	<code><, >, \leq, \geq, \ll, \gg</code>	(Much) Smaller and greater than (or equal to).
\neq	<code>\neq</code>	Not equal to.
$\equiv, :=$	<code>\equiv, \coloneq</code>	Equivalence, Definition of a quantity.
\approx, \sim	<code>\approx, \sim</code>	Approximately equal to, similar to.
\min, \max	<code>\min, \max</code>	Minimum and Maximum.
$\forall, \exists, \nexists$	<code>\forall, \exists, \nexists</code>	For all, (not) exists.
\in, \notin	<code>\in, \notin</code>	In/not in a set.
\subset, \subseteq	<code>\subset, \subseteq</code>	Being a subset of (or equal to) another set.
$\bigcup_{i=1}^n, \bigcap_{i=1}^n$	<code>\bigcup_{i=1}^n, \bigcap_{i=1}^n</code>	Union and Intersection.
\emptyset	<code>\emptyset</code>	The empty set.
\perp	<code>\perp</code>	Perpendicular/orthogonal to.
$\binom{n}{k}$	<code>\binom{n}{k}</code>	The binomial coefficient.
$\dots, \cdots, \ddots, \vdots$	<code>\ldots, \cdots, \ddots, \vdots</code>	Various ellipses.

Table 3.3: Some important logical and set symbols.

3.2 Advanced Mathematical Expressions and Notations

Arrows and Braces The subsequent Table 3.4 shows different methods of making *arrows* and *braces*, possibly with text above/below them.

Function/Symbol(s)	Command(s)	Description
$\leftarrow, \rightarrow, \leftrightarrow$	<code>\leftarrow, \rightarrow, \leftrightarrow</code>	Single arrows.
$\Leftrightarrow, \Rightarrow (\implies), \Leftrightarrow$	<code>\Leftrightarrow, \Rightarrow, \implies, \Leftrightarrow</code>	Double arrows.
$\overset{u}{\leftarrow}_l, \overset{u}{\rightarrow}_l, \overset{u}{\leftrightarrow}_l$	<code>\xleftarrow[l]{u}, \xrightarrow[l]{u}, \xleftrightarrow[l]{u}</code>	Single arrows with labels.
$\overset{u}{\Leftrightarrow}_l, \overset{u}{\Rightarrow}_l, \overset{u}{\Leftrightarrow}_l$	<code>\xLeftrightarrow[l]{u}, \xRightarrow[l]{u}, \xleftrightarrow[l]{u}</code>	Double arrows with labels.
$\overleftarrow{xyz}, \overrightarrow{xyz}$	<code>\overleftarrow{xyz}, \overrightarrow{xyz}</code>	Over-arrows.
$\overline{xyz}, \underline{xyz}$	<code>\overline{xyz}, \underline{xyz}</code>	Overline and Underline.
$\overset{abc}{\overbrace{xyz}}, \underset{abc}{\underbrace{xyz}}$	<code>\overbrace{xyz}^{abc}, \underbrace{xyz}_{abc}</code>	Overbrace and underbrace with labels.

Table 3.4: Arrows and braces in L^AT_EX.

Delimiters Simple *delimiters* can be typed directly in math mode (except the curly brackets `{}` that require `\{ \}`), like

```
\begin{align*}
&\&\frac{1}{N}(1+\frac{n}{N}) &\& [\ln|x|]_a^b &\& \{x|f(x) \neq 0\} \\
\end{align*}
```

outputs

$$\frac{1}{N}\left(1 + \frac{n}{N}\right) \quad [\ln |x|]_a^b \quad \{x|f(x) \neq 0\}$$

However, if the content inside the delimiters is too tall, then we can append `\left` and `\right` before the delimiters on both sides to match the height. Note that they must be balanced. For example,

```
\begin{equation*}
\left[p + a\left(\frac{n}{V}\right)^2\right](V-nb) = nRT
\end{equation*}
```

is rendered as

$$\left[p + a\left(\frac{n}{V}\right)^2\right](V - nb) = nRT$$

Exercise(s)

3.3) Try to type the following statements.

a) $\oint Mdx + Ndy = \iint \left(\frac{\partial N}{\partial x} - \frac{\partial M}{\partial y}\right) dx dy;$

b) $\mu \ll \nu \Leftrightarrow \mu(A) = 0, \forall A | \nu(A) = 0;$

c) $A \subseteq B \cup (A \cap B^c).$

3.2.3 Vectors and Matrices

Denoting Vectors The most essential object in linear algebra is undoubtedly *vectors*, and we need a standard way to denote and distinguish them. One possible solution is to use an overhead arrow: the command `\vec{v}` will output \vec{v} . For longer variables, we can instead use `\overrightarrow` introduced in the last subsection. Another approach is to use boldface, which can be applied to general mathematical symbols if we load the `bm` package: `\bm{v}` will then produce \boldsymbol{v} .

3.2 Advanced Mathematical Expressions and Notations

Unit Vectors For unit vectors, we often use the hat symbol to denote them, e.g. `\hat{v}` gives \hat{v} . Particularly, for the three-dimensional standard unit vectors \hat{i} , \hat{j} , \hat{k} , we can type `\hat{\imath}`, `\hat{\jmath}`, and `\hat{k}`, where we use the versions `\imath`, `\jmath` without the usual dot at the top for placing the hat.

Matrices and Determinants: `bmatrix`, `vmatrix` Another class of objects closely related to vectors is *matrices*. To typeset a matrix in L^AT_EX, we use the `bmatrix` environment provided by the `amsmath` package. For example,

```
\begin{align*}
\begin{bmatrix}
1 & 2 & 3 \\
-4 & \sqrt{5} & c
\end{bmatrix}
\end{align*}
```

outputs

$$\begin{bmatrix} 1 & 2 & 3 \\ -4 & \sqrt{5} & c \end{bmatrix}$$

where `&` separates the entries into columns and `\\` marks the end of a row. By replacing `bmatrix` by `matrix`, `pmatrix`, or `Bmatrix`, the delimiters become none, round, and curly brackets correspondingly. Particularly, we have the `vmatrix` (vertical lines) group to represent determinants. For instance, writing

```
\begin{equation*}
\det(A) =
\begin{vmatrix}
a & b \\
c & d
\end{vmatrix} = ad-bc
\end{equation*}
```

leads to

$$\det(A) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

On top of that, we can manipulate the ellipses symbols to denote a matrix of an arbitrary shape. The following

```
\begin{equation*}
A_{m \times n} =
\begin{bmatrix}
a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\
a_{21} & a_{22} & a_{23} & & a_{2n} \\
a_{31} & a_{32} & a_{33} & & a_{3n} \\
\vdots & & & \ddots & \vdots \\
a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn}
\end{bmatrix}
\end{equation*}
```

produces

$$A_{m \times n} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & & a_{2n} \\ a_{31} & a_{32} & a_{33} & & a_{3n} \\ \vdots & & & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

A column vector can then be represented by a matrix consisting of a single column.

array For an advanced control of matrices, we can use the **array** environment instead. Let's first see how the code will look in the scenario of Gaussian Elimination, and then break down the details. Given

```
\begin{align*}
\left[ \begin{array}{@{}wc{10pt}wc{10pt}wc{10pt}|r}
1 & 2 & 1 & -1 \\
2 & 5 & 3 & 2 \\
0 & 1 & 1 & 0
\end{array} \right]
```

3.2 Advanced Mathematical Expressions and Notations

```
\end{array}\right]
& \to
\left[\begin{array}{@{}wc{10pt}wc{10pt}wc{10pt}|r}
1 & 2 & 1 & 1 \\
0 & 1 & 1 & 4 \\
0 & 1 & 1 & 0
\end{array}\right]
& R_2 - 2R_1 \to R_2
\end{align*}
```

the output will be

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & -1 \\ 2 & 5 & 3 & 2 \\ 0 & 1 & 1 & 0 \end{array} \right] \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & 1 & 1 & 4 \\ 0 & 1 & 1 & 0 \end{array} \right] \quad R_2 - 2R_1 \rightarrow R_2$$

The **array** group places each entry just like the **matrix** one. However, notice the input string `{@{}wc{10pt}wc{10pt}wc{10pt}|r}` before the main content. `@{}` replaces the default left padding with an empty space. **wc** indicates the entries along that column to take a fixed width (**w**) of 10 pt and are centered (**c**). This is repeated for the first three columns to the left. A bar `|` then generates a vertical separating line at the desired location. (For a horizontal line, put `\hline` between the rows inside.) Finally, **r** makes the entries right-aligned (similarly there is **l**) in the last column with a flexible width, and we surround the **array** environment with tall delimiters (see last subsection) manually.

3.2.4 Other Formatting Trivia

abs, norm from physics The **physics** package provides many symbols well-known in the area of physics. Particularly, it defines `\abs{<expression>}` and `\norm{<expression>}` commands for absolute value and norm (length/magnitude), which are quite convenient even for other usages. For example,

```
$\norm{\bm{x}}_1 = \sum_{i=1}^n \abs{x_i}$
```

gives $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$.

mathbb, mathcal Two other types of symbols that may be of interest come from `\mathbb{<character>}` and `\mathcal{<character>}` for sets and classes. For example, the set of all real numbers is commonly denoted by \mathbb{R} (`\mathbb{R}`), while the class of continuously differentiable functions is denoted by \mathcal{C}^1 (`\mathcal{C}^1`).

siunitx For other science applications, the physical quantities involved are often accompanied by units. The **siunitx** package helps facilitate the typesetting of units and expressing the exponents. For example, `\si{N} = \si{kg \per m \per square s}` is interpreted as $N = \text{kg m}^{-1} \text{s}^{-2}$, while `\SI{4.184e3}{J \per kg \per K}` generates $4.184 \times 10^3 \text{J kg}^{-1} \text{K}^{-1}$.

System of Equations To typeset a system of equations, we can use either **aligned** with a large curly bracket to the left, or the **cases** environment. There will be slight differences between these two methods. For instance,

```
\begin{equation*}
\left\{\begin{aligned}
3x + 4y + 5z &= 6 \\
x - 2y + 3z &= -4 \\
x^2 + y^2 &= 1
\end{aligned}\right.
\end{equation*}
```

will produce (notice that we need `\right.` at the end to make a placeholder delimiter to the right for balance)

$$\left\{\begin{array}{l} 3x + 4y + 5z = 6 \\ x - 2y + 3z = -4 \\ x^2 + y^2 = 1 \end{array}\right.$$

3.2 Advanced Mathematical Expressions and Notations

Alternatively, we can write

```
\begin{equation*}
\begin{cases}
3x + 4y + 5z = 6 \\
x - 2y + 3z = -4 \\
x^2 + y^2 = 1
\end{cases}
\end{equation*}
```

to achieve

$$\begin{cases} 3x + 4y + 5z = 6 \\ x - 2y + 3z = -4 \\ x^2 + y^2 = 1 \end{cases}$$

As its name suggests, **cases** is actually designed to represent the values of a variable in different cases, e.g. we may write

```
\begin{equation*}
\begin{aligned}
y(x) = & \\
\begin{cases}
1 & x \in \mathbb{Q} \\
0 & x \notin \mathbb{Q}
\end{cases}
\end{aligned}
\end{equation*}
```

to get

$$y(x) = \begin{cases} 1 & x \in \mathbb{Q} \\ 0 & x \notin \mathbb{Q} \end{cases}$$

Spacing in Math Mode In math mode, we often employ pre-defined commands instead of `\hspace` or `\vspace` to adjust the spacing. They are shown in Table 3.5 below.

Command	Description	Effect
<code>\quad</code>	Space of 1 em in the current math font size (= 18 mu)	a b
<code>\qquad</code>	Double of <code>\quad</code> (= 36 mu)	a b
<code>\,</code>	3/18 of <code>\quad</code> /3 mu	a b
<code>\:</code>	4/18 of <code>\quad</code> /4 mu	a b
<code>\;</code>	5/18 of <code>\quad</code> /5 mu	a b
<code>\!</code>	−3/18 of <code>\quad</code> /−3 mu	ab
<code>\(space)</code>	Space as in normal text	a b

Table 3.5: Spacing commands in math mode.

Sizes We can control the font size in either math mode with the usual size commands in Table 2.2. For the inline mode, we can write something like

```
{\Large $N(0,1) \sim e^{-x^2/2}$}
```

to get $N(0, 1) \sim e^{-x^2/2}$. On the other hand, for the display mode, we may put the size command before the math environment, e.g.

```
\footnotesize
\begin{align*}
\mathcal{L}[y^{(n)}](s) &= s^n Y(s) - s^{n-1}y(0) - s^{n-2}y'(0) - \backslash
&\quad \text{cdots} - y^{(n-1)}(0) \backslash \backslash
&= s^n Y(s) - \sum_{k=0}^{n-1} s^{(n-1)-k}y^{(k)}(0)
\end{align*}
\normalsize % back to default font size
```

will yield

$$\begin{aligned}
 \mathcal{L}[y^{(n)}](s) &= s^n Y(s) - s^{n-1}y(0) - s^{n-2}y'(0) - \dots - y^{(n-1)}(0) \\
 &= s^n Y(s) - \sum_{k=0}^{n-1} s^{(n-1)-k}y^{(k)}(0)
 \end{aligned}$$

3.2 Advanced Mathematical Expressions and Notations

mathcolor To apply colors in math mode, we can replace the `\textcolor` command with `\mathcolor`. For example,

```
\begin{align*}
\mathcolor{Blue}{\frac{\partial \vec{u}}{\partial t}} + \mathcolor{Green}{\vec{u} \cdot \nabla \vec{u}} = \mathcolor{Red}{\vec{F}}
\end{align*}
```

is displayed as

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \vec{F}$$

Exercise(s)

3.4) Reproduce the following output.

$$\begin{cases} x + 2y = 3 \\ x - 3y = -2 \\ -x + y = 1 \end{cases} \Leftrightarrow \begin{bmatrix} 1 & 2 \\ 1 & -3 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix} \quad (3.11)$$

Various Special Structures in L^AT_EX

Introduction This chapter presents different special environments in L^AT_EX apart from the display math mode and verbatim form previously, such as lists, figures, tables, and minipages.

4.1 Lists

4.1.1 Unordered Lists

itemize The ability to organize things into a *list* is essential in any documenting system. In L^AT_EX, we can achieve this by using the **itemize** environment with the `\item` command. For example, if we write

```
\begin{itemize}
\item Canada
\item Japan \begin{itemize}
\item Tokyo
\item Kyoto
\end{itemize}
\item Korea \begin{itemize}
\item Seoul
\item Pusan
```

```
\end{itemize}  
\end{itemize}
```

then it will show up as

- Canada
- Japan
 - Tokyo
 - Kyoto
- Korea
 - Seoul
 - Pusan

Notice that the list can be nested and the items are *unordered/bulleted*.

4.1.2 Ordered Lists

enumerate Similarly, we can have an *ordered/numbered* list by using the **enumerate** environment. For example, by typing

```
\begin{enumerate}  
  \item A robot may not injure a human being ... % (continue)  
  \item A robot must obey the orders given it by human ...  
  \item A robot must protect its own existence ...  
\end{enumerate}
```

we acquire the following outcome:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.

2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

It is also possible to have nested **enumerate** groups.

enumitem The **enumitem** package enhances the typesetting of lists. One of the prime utilities is to change the starting labels or bullets for every item by providing the **label** option. For example,

```
\begin{enumerate}[label=\alph*]  
  \item Apple  
  \item Banana  
  \item Grape  
\end{enumerate}
```

generates

- a) Apple
- b) Banana
- c) Grape

There are other possible choices for **label**, e.g. **\arabic***, **\roman***, **\Roman***, **\Alph***. Another usage of the **enumitem** package is to make a continued list. For example, by ticking the **resume*** option:

```
\begin{enumerate}[resume*]  
  \item Watermelon  
  \item Orange  
\end{enumerate}
```

we have

- d) Watermelon

e) Orange

4.2 Figures and Tables

4.2.1 Figures

figure, includegraphics To import figures into the document, we need to load the `graphics` package and then use the `\includegraphics{<file_name>}` command. Given that the image is placed under the project directory, we hereby go through an example which is displayed as Figure 4.1 on the next page. The code to produce that figure is

```
\begin{figure}[ht!]  
\centering  
\includegraphics[width=0.4\linewidth]{graphics/Ada_Lovelace_portrait.  
  jpg} % replace the path with your own file  
\caption{The portrait of Ada Lovelace.}  
\label{fig:ada}  
\end{figure}
```

First, we need to enclose the `\includegraphics` command within a `figure` environment. The `ht!` option indicates that priority is given to put the figure structure exactly in the place where the code is inserted (**h**: here), or at the top of a page (**t**). The `width` option enforces the width of the image to the input value (and similarly there are `height` and `scale`). The `caption` command unsurprisingly generates the caption, while the `label` command works as it is in math mode and allows us to reference it by writing `\ref{fig:ada}`.

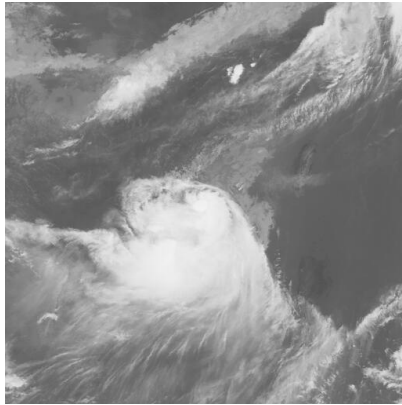
subcaption We can construct a set of subfigures within an overarching figure by utilizing the `subcaption` package and `subfigure` groups. To illustrate, the following code is deployed to generate Figure 4.2:



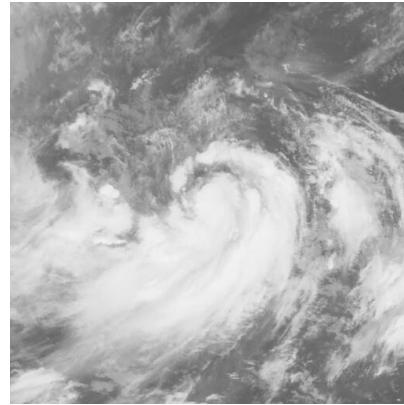
Figure 4.1: The portrait of Ada Lovelace.

```
\begin{figure}[ht!]  
\centering  
\begin{subfigure}[b]{0.45\textwidth}  
\centering  
\includegraphics[width=0.8\linewidth]{graphics/MTS108082203.200812.  
  jpg}  
\caption{Typhoon Nuri (2008).}  
\end{subfigure}  
\begin{subfigure}[b]{0.45\textwidth}  
\centering  
\includegraphics[width=0.8\linewidth]{graphics/MTS212072303.201208.  
  jpg}  
\caption{Typhoon Vicente (2012).}  
\end{subfigure}  
\caption{The infrared satellite images of various Tropical Cyclones  
  affecting Hong Kong.}  
\label{fig:TC1}  
\end{figure}
```

The **b** option sets the vertical alignment of **subfigure** at the bottom.



(a) Typhoon Nuri (2008).



(b) Typhoon Vicente (2012).

Figure 4.2: The infrared satellite images of various Tropical Cyclones affecting Hong Kong. (Source: Digital Typhoon)

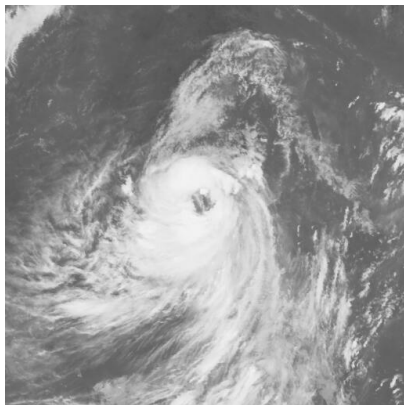
ContinuedFloat To make a longer figure of subfigures that spans multiple pages, we can simply arrange them into separate **figure** environments and call the `\ContinuedFloat` command in all the subsequent **figure** groups. Continuing from the last example, we may have

```
\begin{figure}[hb!]  
\ContinuedFloat % here!  
\caption{(Cont.) The infrared satellite images of various Tropical  
Cyclones affecting Hong Kong.}  
\centering  
\begin{subfigure}[b]{0.45\textwidth}  
...  
\caption{Typhoon Haima (2016).}  
\end{subfigure}  
...  
\begin{subfigure}[b]{0.45\textwidth}  
...  
\caption{Typhoon Saola (2023).}
```

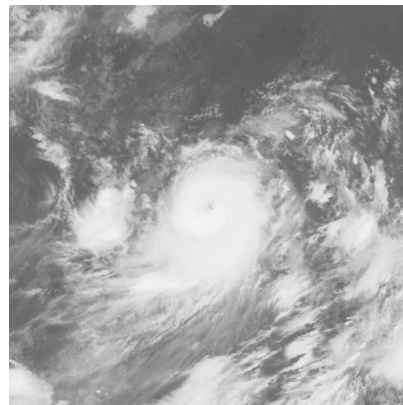
```
\end{subfigure}  
\end{figure}
```

producing

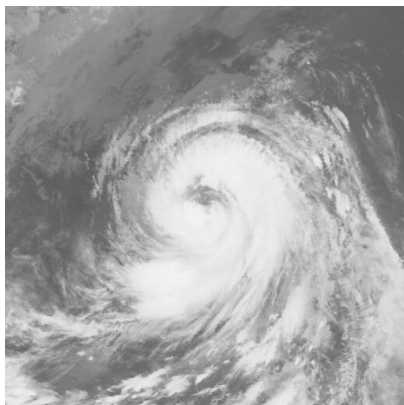
Figure 4.2: (Cont.) The infrared satellite images of various Tropical Cyclones affecting Hong Kong.



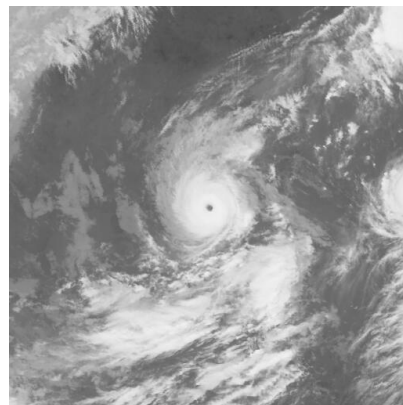
(c) Typhoon Haima (2016).



(d) Typhoon Hato (2017).



(e) Typhoon Mangkhut (2018).



(f) Typhoon Saola (2023).

4.2.2 Tables

table, tabularx Just like embedding figures, building a table requires us to place the content inside the corresponding **table** environment. While it is possible to use the native **tabular** class for the actual table itself, a more powerful version is provided by the **tabularx** package and its class that bears the same name. This is demonstrated via Table 4.1 thereafter, which is generated by

```
\begin{table}[ht!]
\centering
\begin{tabularx}{\textwidth}{|l|p{0.55\textwidth}}|>\raggedleft|X|>\raggedleft\arraybackslashX|}
\hline
Unit & Description & Attack & Defense \\
\hline
Infantry & The most basic unit and backbone of any army, all-around abilities with a cheap cost. & 20 & 25 \\
\hline
Cavalry & The shock unit in an army with very strong power. & 40 & 30 \\
\hline
Artillery & The support unit that provides bombardment support from far away. & 30 & 5 \\
\hline
\end{tabularx}
\caption{The unit statistics table for a hypothetical game.}
\label{tab:armyunits}
\end{table}
```

The **ht!** option, **caption**, and **label** work exactly as the figure counterpart. For the **tabularx** group, the first argument indicates the width of the entire table, set to **\textwidth** here. The second argument **{|l|p{0.55\textwidth}}|>\raggedleft|X|>\raggedleft\arraybackslashX|}** indicates the justification of the columns: the first column is left-aligned (**l**, similarly we have **c** and **r**) and its size will fit the text; the second column (**p**) forces a width of 0.55

Unit	Description	Attack	Defense
Infantry	The most basic unit and backbone of any army, all-around abilities with a cheap cost.	20	25
Cavalry	The shock unit in an army with very strong power.	40	30
Artillery	The support unit that provides bombardment support from far away.	30	5

Table 4.1: The unit statistics table for a hypothetical game.

times `\textwidth`; the remaining width is distributed evenly to last two columns (X). The part of `>\raggedleft` is applied to the X columns, making them right-aligned.¹ Finally, `|` and `\hline` produce vertical/horiztonal separating lines; `&` slices between the columns and `\\` marks the end of a row.

Also, note that `\ContinuedFloat` can also be applied to `table`.

captionbeside It is also possible to arrange the table so that the caption appears to the side of it. This is done by stacking the `captionbeside` environment provided by KOMA-script. For example, the code

```
\begin{table}[ht]
\begin{captionbeside}{This caption appears to the left of the
  Fibonacci numbers table.}[l][\textwidth]{
\adjustbox{valign=t}{
  \begin{tabularx}{0.4\textwidth}{|X|X|}
    \hline
    $n$ & $F_n$ \\
    \hline
    $1$ & $1$ \\
    \hline
    $2$ & $1$
  \end{tabularx}
}
```

¹`\arraybackslash` is needed in the last column, see [TeX StackExchange 372464](#).

```

\hline
$3$ & $2$ \\\
\hline
$4$ & $3$ \\\
\hline
$5$ & $5$ \\\
\hline
$6$ & $8$ \\\
\hline
\end{tabularx}}
}
\end{captionbeside}
\label{tab:fib}
\end{table}

```

produces Table 4.2 below.

Table 4.2: This caption appears to the left of the Fibonacci numbers table.

n	F_n
1	1
2	1
3	2
4	3
5	5
6	8

We fill the caption in the first argument, followed by the relative position of the caption (**l**: left) and the full width of the structure, finally with the actual **tabularx** object. We also have to additionally load the **adjustbox** package and use the corresponding command to tell the table to align itself at the top (**valign=t**). This also requires us to first set the **\KOMAOPTIONS** to take **captions=besidetop** (likewise we have **captions=besidebottom** and more).

Shared Numbering between Figures and Tables Sometimes we may want to share the numbering between *floats* (including figures, tables, and so on). This

is done by the following patch that can be inserted into the preamble:

```
\makeatletter
\let\c@table\c@figure
\let\ftype@table\ftype@figure
\makeatother
```

This involves the primitive T_EX functions, so we will not discuss them there. For more information, read StackOverflow 3865036, and T_EX StackExchange 8351 for what the `\makeatletter` and `\makeatother` commands do.

Exercise(s)

4.1) Try to import and load your favorite image into the document.

4.2) Recreate any one of the tables in Chapter 3.

4.3 Minipages and Multiple Columns

minipage Sometimes we may want to partition the content into smaller blocks that are embedded within the current page, and can be placed or ordered (e.g. parallel) in the way we want. The **minipage** environment basically acts like a more versatile version of a **parbox** environment and serves this purpose. For example, something like

```
yields \par
\begin{center}
\begin{minipage}[b]{0.48\textwidth}
\lipsum[6]
\end{minipage}
\hfill
\begin{minipage}[b]{0.48\textwidth}
\lipsum[7]
\end{minipage}
\end{center}
```

yields

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

The `[b]` option indicates the baseline is set at the bottom, and hence the two blocks will be bottom-aligned, provided that their width is fixed to 0.48 times `\textwidth` and thus they fit in the main text area.

parcolumns The `parcolumns` package can also achieve the above effect and is more specialized for typesetting different pieces in two or more parallel columns. It also supports page breaks. Using the same example, we can write

```
\begin{parcolumns}{2}
\colchunk[1]{\lipsum[6]}
\colchunk[2]{\lipsum[7]}
\colplacechunks
```

```
\colchunk[1]{\lipsum[8]}  
\colchunk[2]{\lipsum[9]}  
\end{parcolumns}
```

to get

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetur at, consectetur sed, eleifend

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetur a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed

ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed portitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consetetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

where the first argument of the environment clearly indicates the number of columns and the `\colplacechunks` command releases the loaded `\colchunk[<col_no.>]` and goes to the next paragraph.

multicol A task closely related to what **parcolumns** does above is to typeset a single, continuous stream of text along multiple columns, like in many academic papers. The **multicol** package is designed for this and will carry out the automatic splitting. For example, by encapsulating the text inside the **multicols** environment as

```
\begin{multicols}{2}
```

```
Zhuge Liang (born 181, Yangdu [now Yinan, Shandong province], China--  
died August 234, Wuzhangyuan [now in Shaanxi province], China) was  
a celebrated adviser to Liu Bei, founder of the Shu-Han dynasty  
(221--263/264).
```

```
...
```

4.3 Minipages and Multiple Columns

A mechanical and mathematical genius, Zhuge is credited with inventing a bow for shooting several arrows at once and with perfecting the Eight Dispositions, a series of military tactics. In the *Sanguozhi yanyi* (Romance of the Three Kingdoms), the great 14th-century historical novel, Zhuge is one of the main characters; he is portrayed as being able to control the wind and foretell the future.

`\end{multicols}`

(again with the number of columns indicated in the first argument) we may acquire the following layout:

<p>Zhuge Liang (born 181, Yangdu [now Yinan, Shandong province], China—died August 234, Wuzhangyuan [now in Shaanxi province], China) was a celebrated adviser to Liu Bei, founder of the Shu-Han dynasty (221–263/264).</p> <p>Quick Facts: Wade-Giles romanization: Chu-ko Liang Courtesy name: Kongming Born: 181, Yangdu [now Yinan, Shandong province], China Died: August 234, Wuzhangyuan [now in Shaanxi province], China (aged 53)</p> <p>Zhuge, to whom supernatural powers often are ascribed, has been a favoured character of many Chinese plays and stories. Legend states that Liu Bei, then a minor military figure, heard of Zhuge Liang's great wisdom and came three times to the wilderness retreat to which</p>	<p>Zhuge had retired to seek him out as an adviser. It is known that Zhuge helped Liu organize a large army and found a dynasty. Liu was so impressed with Zhuge's wisdom that on his deathbed Liu urged his son to depend on Zhuge's advice and urged Zhuge to ascend the throne himself if the prince were unable to rule. Some historical accounts indicate that Zhuge died from illness while leading a military campaign in 234.</p> <p>A mechanical and mathematical genius, Zhuge is credited with inventing a bow for shooting several arrows at once and with perfecting the Eight Dispositions, a series of military tactics. In the <i>Sanguozhi yanyi</i> (Romance of the Three Kingdoms), the great 14th-century historical novel, Zhuge is one of the main characters; he is portrayed as being able</p>
--	--

to control the wind and foretell the future. (Source: Encyclopaedia Britannica)

twocolumn We can also pass the `twocolumn=true` option to `\KOMAoptions` to demand the entire book to be formatted in two columns globally. However, note that it will greatly mess up the layout of this book. (The decision to adapt such a format should be made at an early time!)

Self-defined Commands and Environments

Introduction This chapter concerns the possibility for users to define new commands and environments in L^AT_EX, by leveraging flow control just like any other programming language.

5.1 Self-defined Commands

newcommand* The main way to define our own commands (*macros*) is to invoke the `\newcommand*{<command_name>}[<no._arg>]{<code>}`¹ statement. The necessity arises mainly when we want to repeatedly apply the same action, in addition to ensuring code readability and maintenance. Now, let's see a simple example of highlighting keywords in a particular style:

```
\newcommand*{\mykeyword}[1]{\textcolor{Red}{\textbf{#1}}}
```

This definition can be put either in the preamble (preferable) or any suitable location in the main document. Writing `\mykeyword{Attention!}` then gives **Attention!** The `#1` part indicates where the first argument will go during the code

¹There is an unstarred version, but just sticking to the starred version will be adequate.

execution, and the logic is similar when there are multiple arguments. Remember that when calling any command, each argument requires exactly one pair of curly brackets to receive it.

Optional Arguments As in many programming languages, there can also be optional arguments that may come with a default value when defining a command. The syntax will become `\newcommand*{<command_name>}[<no._arg>][<default_value>]{<code>}`. For example, if we define

```
\newcommand*{\homoeqn}[2][0]{$f(\#2x,\#2y) = \#2^{\#1}f(x,y)$}
```

Then `\homoeqn{t}` gives $f(tx,ty) = t^0 f(x,y)$ while `\homoeqn[1]{s}` gives $f(sx,sy) = s^1 f(x,y)$. When interpreting indices in the code statement, optional arguments will take precedence over the compulsory arguments. In this case, `#1` represents the optional argument with the default value of 0, which has been overridden by the new option 1 within the square brackets during the second call.

renewcommand* If we want to edit a command that is already defined by us or another package, we will need to use the `\renewcommand*` statement to properly update and overwrite the original command. It has the same format as `\newcommand*`. Using the same example, we can write

```
\renewcommand*{\mykeyword}[1]{\textcolor{Green}{\textit{\textbf{\#1}}}}
```

Writing `\mykeyword{Okay!}` now then gives *Okay!*

Exercise(s)

5.1) Create a command that takes two arguments and outputs a sentence in the form of: There are `<no._of_population>` (comma-separated) people in `<city>`. The `\num[group-separator={,}]` command by the `siunitx` package will be useful for processing the first argument. Try to execute it

multiple times with different inputs.

5.2 Flow Control

5.2.1 If-then-else Statements

ifthen Commands/functions are rather boring if there is no constraint or checking imposed. As you probably know, *if-then-else* statements are one of the major flow control constructs in all programming languages, and L^AT_EX is no exception. With these, we can produce more complex outcomes with commands. While there are primitive T_EX syntax such as `\if` and `\else` for that, it is easier and more natural (at least in my opinion) if we use the verbose `\ifthenelse` construct provided by the **ifthen** package. The format is

```
\ifthenelse{<boolean_test>}{<then clause>}{<else clause>}
```

The first argument contains a test that evaluates to some *boolean* value (true or false). If the test returns true, then the "then clause" in the second argument is executed. Otherwise, if it is false, then the "else clause" in the third argument is executed instead. The most basic test is to compare two quantities, and here is a very simple example: if we type

```
\ifthenelse{1 > 2}{Preposterous!}{Of course not...}
```

we should see "Of course not...".

pgfmath When designing a boolean test for the `\ifthenelse` statement above, we often need a way to compute the results of math expressions for comparison. The native T_EX does offer some commands such as `\numexpr` or `\dimexpr` for that, but here we will utilize the **pgfmath** package to parse math expressions. The usage mainly takes the form of `\pgfmathparse{<expression>}`.

For instance, `\pgfmathparse{2+2}\pgfmathresult` returns 4.0 where the `\pgfmathresults` command stores the last value processed and prints it out. Alternatively, we can save it to a macro by `\pgfmathsetmacro{<macro>}{<expression>}`. Using the same example, we can write something like

```
\pgfmathsetmacro{\myans}{2+2}
```

typing `\myans` then gives 4.0.

To learn more about how a `pgfmath` expression should be formatted, see <https://tikz.dev/math-parsing>.

lengthtest Subsequently, we can design a command that checks equality and looks like

```
\newcommand*{\myequal}[2]{\pgfmathsetmacro{\Lhs}{#1}\pgfmathsetmacro{\Rhs}{#2}% <- this % is needed to consume the spacing
\ifthenelse{\lengthtest{\Lhs pt = \Rhs pt}}{#1 is equal to #2}{#1 is not equal to #2.}}
```

Typing `\myequal{2*3}{6}` then outputs "2*3 is equal to 6" as expected. Notice that in the `ifthenelse` boolean test, we have not directly done the naive comparison as `\Lhs = \Rhs`. This is because the original method only handles integers, but the `pgfmath` calculation produces float numbers. To circumvent this, we must use the `\lengthtest` command, which is designed to compare decimal dimensions, and we will only need to add the same length unit to both sides. (TEX StackExchange 84625)

\AND, \OR, \NOT In an `ifthenelse` test, we may need to compose different booleans using logical operators. There are the self-explanatory `\AND`, `\OR`, and `\NOT` for that. As a demonstration,

```
\ifthenelse{\NOT \ (1 = 2) \AND \ (0 = 0)}{These make sense!}{What?}
```

gives "These make sense!". We enclose each smaller test with `\(\)`. Note that there is no precedence rule in **ifthen** and the evaluation goes from left to right straightly, except when using `\(\)`.

equal, isundefined There are two other boolean tests that can be helpful: `\equal{<string>}{<string>}` and `\isundefined{<command_name>}`, which check whether two strings are equal and if a command exists, respectively. A quick use is to determine if a string is empty by `\equal{<string>}{}`.

5.2.2 For Loops

pgffor Another essential type of flow control is *for loops*, which repeatedly execute a code block over some range of values. The **pgffor** offers this functionality with the `\foreach` construct. The format goes like

```
\foreach \<variable> in {<range>} {  
  % do something  
}
```

A toy example will be

```
\foreach \x in {5,...,1} {  
  \x \x!}  
Time is up! \x
```

that outputs

```
5!  
4!  
3!  
2!  
1! Time is up!
```

The `{5, ..., 1}` part is a shorthand for `{5, 4, 3, 2, 1}` and it also works for ascending order or other patterns.²

We can also simultaneously loop over multiple variables by separating them with `/`, for example:

```
\foreach \y/\z in {1/2, 2/3, 3/5, 4/7} {  
  \\\ Prime $\y$ \rightarrow \z$}
```

produces

Prime 1 \rightarrow 2

Prime 2 \rightarrow 3

Prime 3 \rightarrow 5

Prime 4 \rightarrow 7

Nested Loops Moreover, we can produce nested loops as in other programming languages. For instance, the block

```
\foreach \ii in {0,...,2} {\%  
  \foreach \jj in {1,...,4} {%  
    (\ii, \jj)%  
  }  
}
```

will generate the following pattern:

(0, 1)(0, 2)(0, 3)(0, 4)

(1, 1)(1, 2)(1, 3)(1, 4)

(2, 1)(2, 2)(2, 3)(2, 4)

count When iterating over a range of values, the corresponding index can be saved into a variable as **count**. For example, the previous example of printing out the first four primes can be replaced by the following equivalent snippet:

²For a general discussion, see [TeX StackExchange 142188](#).

```
\foreach \z [count=\y] in {2, 3, 5, 7} {
  \\\ Prime $\y$ \rightarrow \z$}
```

evaluate Also, in a `\foreach` loop, calculations can be applied over the iterated variable by the **evaluate** option. As an illustrative example, this loop

```
\foreach \x [evaluate=\x as \y using \x^2] in {1,...,10} {
  \\\ $\x^2 = \y$}
```

readily outputs

$$1^2 = 1.0$$

$$2^2 = 4.0$$

$$3^2 = 9.0$$

$$4^2 = 16.0$$

$$5^2 = 25.0$$

$$6^2 = 36.0$$

$$7^2 = 49.0$$

$$8^2 = 64.0$$

$$9^2 = 81.0$$

$$10^2 = 100.0$$

remember Another feature of a `\foreach` loop is the **remember** option, which stores the current variable and recalls it in the next iteration. Stealing the example from the **PGF** User Manual:

```
\foreach \x [remember=\x as \lastx (initially A)] in {B,...,H}{\lastx
  $\to$\x, }
```

produces $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E, E \rightarrow F, F \rightarrow G, G \rightarrow H$, in the expected order.

breakforeach An important component of any for loop is the ability to break it given some certain condition. This is done by the `\breakforeach` command.

This will be used in combination with an `\ifthenelse` statement. For example, the code snippet

```
\foreach \x in {1,...,100} {%  
\ifthenelse{\NOT \(\x = 13\)}{\x, }{13 is an unlucky number! Stop! \  
  breakforeach}  
}
```

outputs 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, Wait, 13 is an unlucky number! Stop!

Exercise(s)

5.2) Define a command that converts Celsius temperature to Fahrenheit and prints a warning if the Fahrenheit temperature is higher than 100 degrees.

5.3) By manipulating all the tools introduced in this chapter, efficiently imitate the following outputs. The `int` and `Mod` operators for `pgfmath` will be useful.

1 is not divisible by 3. 1 is not divisible by 5.
2 is not divisible by 3. 2 is not divisible by 5.
3 is divisible by 3. 3 is not divisible by 5.
4 is not divisible by 3. 4 is not divisible by 5.
5 is not divisible by 3. 5 is divisible by 5.
6 is divisible by 3. 6 is not divisible by 5.
7 is not divisible by 3. 7 is not divisible by 5.
8 is not divisible by 3. 8 is not divisible by 5.
9 is divisible by 3. 9 is not divisible by 5.
10 is not divisible by 3. 10 is divisible by 5.
11 is not divisible by 3. 11 is not divisible by 5.
12 is divisible by 3. 12 is not divisible by 5.
13 is not divisible by 3. 13 is not divisible by 5.
14 is not divisible by 3. 14 is not divisible by 5.

15 is divisible by both 3 and 5.
16 is not divisible by 3. 16 is not divisible by 5.
17 is not divisible by 3. 17 is not divisible by 5.
18 is divisible by 3. 18 is not divisible by 5.
19 is not divisible by 3. 19 is not divisible by 5.
20 is not divisible by 3. 20 is divisible by 5.

5.3 Self-defined Environments

newenvironment* Similarly, we can also define our own environments by the `\newenvironment*{env_name}{begin}{end}` method. `begin/end` stores the code to be run before the start/after the end of the new environment. For example, by defining

```
\newenvironment*{mylargeblueeqn}{\color{Blue}\LARGE\begin{equation}}{\end{equation}}
```

Then

```
\begin{mylargeblueeqn}
1+1 = 2
\end{mylargeblueeqn}
```

displays the expected blue, magnified equation:³

$$1 + 1 = 2 \tag{5.1}$$

³Note that replacing `equation` with `align` instead will cause an error, read [T_EX StackExchange 236664](#).

renewenvironment* There is also the concurrent `\renewenvironment*` for renewing the definition of an environment. However, we seldom need to (re)define an environment on our own: In a math document, the interface provided by the **tcolorbox** package (see Chapter ??) will fulfill most of the usages. So we will keep this section short.

More on Book Layout Design



Framed Theorems and Exercises



Plotting with Tikz



Miscellaneous

