Modern LaTeX

Second edition

Matt Kline

Print copies of this book are available at cost on TheBookPatch and Amazon.

The author apologizes for any typos, formatting mistakes, inaccuracies, and other flubs. He welcomes fixes and improvements in this book's Git repository at
https://github.com/mrkline/latex-book

Questions, comments, concerns, and diatribes can also be emailed to
matt <at> bitbashing.io

The author does not have a checking account with the Bank of San Serriffe, but he will happily buy you a drink when we meet as thanks for your help.

Second edition (online PDF), typeset October 25, 2022.

To Max, who once told me about a cool program he used to type up his college papers.

Contents

1. Typography and You

Life is a parade of written language. Ads, apps, articles, emails, essays, menus, messages, and more constantly shove text in your face. And when you read that text, you see so much more than the author's verbiage. Consciously or not, you notice the shapes and sizes of letters. You notice how those letters are arranged into words, how those words are arranged into paragraphs, how those paragraphs are arranged onto pages and screens. You notice *typography*.

> Typography is why these two lines remind you of awful essays you wrote in school. Do many books look this way? Why not?

There is a reason street signs don't look like this:

> E Gorham St.

And why a very important switch in a spaceship is labeled like this:

> CM/SM SEP

Not like this:

> *CM/SM Sep*

Effective writing isn't just about the words you choose, it's also about their look and layout. Good typography isn't just art—it's a tool to help people understand you better, faster. And if you want to leverage that tool, you should try LaTeX!

What is LaTeX?

LaTeX (pronounced "lay-tech" or "lah-tech") is an alternative to word processors like Microsoft Word, Apple Pages, Google Docs, and LibreOffice. These other applications follow the principle of *What You See Is What You Get* (WYSIWYG), where what is on screen is the same as what comes out of your printer. LaTeX is different. Here, documents are written as "plain" text files, using *markup* to specify how the final result should look. If you've done any web design, this is a similar process—just as HTML and CSS describe the page you want browsers to draw, markup describes the appearance of your document to LaTeX.

```
\LaTeX{} (pronounced ``lay-tech'' or ``lah-tech'') is an
alternative to word processors like Microsoft Word,
Apple Pages, Google Docs, and LibreOffice.
These other applications follow the principle of
\introduce{What You See Is What You Get}
\acronym{(wysiwyg)}, where what is on screen is the same
as what comes out of your printer.
\LaTeX{} is different. Here, documents are written as
``plain'' text files, using \introduce{markup}
to specify how the final result should look.
If you've done any web design, this is a similar
process---just as \acronym{html} and \acronym{css}
describe the page you want browsers to draw,
markup describes the appearance of your document to \LaTeX.
```

The LaTeX markup for the paragraph above

This might seem strange if you haven't worked with markup before, but it comes with a few advantages:

1. You can handle your writing's content and its presentation separately. At the start of each document, you describe the design you want. LaTeX takes it from there, consistently formatting your whole text just the way you asked. Compare this to a WYSIWYG system, where you constantly deal with appearances as you write. If you changed the look of a caption, were you sure to find all the other captions and do the same? If the program arranges something in a way you don't like, is it hard to fix?

2. You can define your own commands, then tweak them to instantly adjust every place they're used. For example, the `\introduce` and `\acronym` commands in the example above are my own creations. One *italicizes* text, and the other sets words in SMALL CAPS with a bit of extra L E T T E R S P A C I N G so the characters don't look TOO CROWDED. If I decide that I'd prefer new terms to ***have this look***, or that acronyms should be formatted LIKE THIS, I just change the two lines that define those commands, and every instance in this book immediately takes on the new look.

3. Being able to save the document as plain text has its own benefits:
    - You can edit it with any basic text editor.
    - Structure is immediately visible and simple to replicate.*
    - You can automate content creation using scripts and programs.
    - You can track your changes with version control software, like Git or Mercurial.

Another guide?

You might wonder why the world needs another guide for LaTeX. After all, it has been around for decades. A quick search finds nearly a dozen books on the topic. There are plenty of resources online.

Unfortunately, most LaTeX guides have two fatal flaws: they are long, and they are old. Beginners don't want—or need—hundreds of pages just to learn the basics, and older guides waste your time with outdated information. When LaTeX was first released in 1986, none of the publishing technologies we use today existed. Adobe wouldn't debut their Portable Document Format for seven more years, and desktop publishing was a fledgling curiosity. This shows—badly—in many LaTeX guides. If you look for instructions to change your document's font, you get swamped with bespoke nonsense.†

*Compare this to WYSIWYG systems, where it's not always obvious how certain formatting was produced or how to match it.

†Take these criticisms with a grain of salt. The fact that LaTeX is still here after all of the technology around it became obsolete—multiple times—is a testament to its staying power.

The good news is that LaTeX has improved by leaps and bounds in recent years. It's time for a guide that doesn't weigh you down with decades of legacy or try (in vain) to be a comprehensive reference. After all, you're a smart, resourceful person who knows how to use a search engine. This book will:

1. Teach you the fundamentals of LaTeX.
2. Point you to places where you can learn more.
3. Show you how to take advantage of modern typesetting technologies.*
4. End promptly thereafter.

Let's begin.

*By modern, I mean "from the mid-1990s", but most web browsers and desktop publishing software are only just starting to catch up.

2. Installation

When you install LaTeX on your computer, it comes packaged as a *distribution* that contains:

1. LaTeX, the program—the thing that turns markup into typeset documents.*
2. A common set of LaTeX *packages*. Packages are bundles of code that do all sorts of things, like provide new commands or change a document's style. We'll see lots of them in action throughout this book.
3. Editors and other helpful tools.

Each major operating system has its own LaTeX distribution:

Mac OS has MacTeX. Grab it from `http://www.tug.org/mactex` and install it using the instructions there.

Windows has MikTeX. Install it from `https://miktex.org/download`. MikTeX has the helpful ability to automatically download additional packages as your documents use them for the first time.

Linux and BSD use TeX Live. Like most software, it is provided through your OS's package manager. Linux distributions usually offer a `texlive-full` or `texlive-most` package that installs everything you need.†

*Well, actually, multiple LaTeX programs, but we're getting to that.

†If you would prefer a smaller install, Linux distributions usually break TeX Live into multiple packages. Look for ones with names like `texlive-core`, `texlive-luatex` and `texlive-xetex`. As you use LaTeX more, you may need less-common packages, which usually have names like `texlive-latexextra`, `texlive-science`, and so on. Of course, all of this may vary from one Linux distribution to another.

Editors

Since LaTeX source files are regular text files, you write them with the usual choices: Vim, Emacs, VS Code, and so on.* There are also editors designed specifically for LaTeX, which often come with a built-in PDF viewer. (You can find a good list on the LaTeX Wikibook, in its installation chapter. See Appendix B.)

Online options

If you don't want to install LaTeX on your computer, try online editors like ShareLaTeX or Overleaf. This book doesn't focus on these web-based tools, but the same basics apply. Of course, you have less control over certain aspects, like available fonts, the version of LaTeX used, and so on.

*If you've never used any of these, try a few. They're popular with programmers and other folks who shuffle text around screens all day. Just don't use Notepad. Life is too short.

3. Hello, LaTeX!

Now that you have LaTeX installed, let's try it out. Open up your favorite text editor and save the following as `hello.tex`:

```
\documentclass{article}
% Say hello
\begin{document}
Hello, World!
\end{document}
```

Next, we run this file through LaTeX (the program)* to get our document. The installation placed several different versions—or *engines*—on your machine, but throughout this book, we use the newest ones: LuaLaTeX and XeLaTeX.†

If you are using a LaTeX-specific editor, it should have some menu to select the engine you would like to use, along with a button to generate your document. Otherwise, run the following from your terminal:‡

```
$ xelatex hello.tex
```

Feel free to try `lualatex` instead—there are a few differences between the two that we will discuss later, but either is fine for now. With luck, you should see some output that ends in a message like:

```
Output written on hello.pdf (1 page).
Transcript written on hello.log.
```

*Not to be confused with LaTeX the lunchbox, LaTeX the breakfast cereal, or LaTeX the flamethrower. The kids love this stuff!

†See Appendix A for a comparison of the various LaTeX engines.

‡How to work a terminal, make sure the newly-installed LaTeX programs are in your `PATH`, and so on are all outside the scope of this book. As is tradition, the leading dollar sign in this example just denotes a console prompt, and shouldn't actually be typed.

And in your current directory, you should find a newly minted `hello.pdf`. Open it up and you should see a page with this at the top:

> Hello, World!

Congrats, you created your first document! Let's unpack what we just did.

All LaTeX documents begin with a `\documentclass` command, which picks a base "style" to use. Many classes are available—and you can even create your own—but common ones include `article`, `report`, `book`, and `beamer`.* For the average document, `article` is probably a good choice. The next line, `% Say hello`, is a *comment*. LaTeX ignores the rest of a line once it sees a percent sign, so we use it to leave notes for anybody reading the document's source.† Finally, `\begin{document}` tells LaTeX that what follows is the actual document content, and `\end{document}` states that we are finished.

Let's cover some more basics.

Spacing

LaTeX generally handles inter-word spacing for you, regardless of how many times you mash the space bar or tab key. For example, typing the following into your editor

```
The number  of   spaces    between words doesn't   matter.
The same is true for space between sentences.

An empty line ends the previous paragraph and
starts the next.
```

yields

> The number of spaces between words doesn't matter. The same is true for space between sentences.
>
> An empty line ends the previous paragraph and starts the next.

*This last one is for slideshows, named after a German term for a projector.

†Including, perhaps most importantly, a confused version of your future self!

Notice how LaTeX automatically follows typographic conventions, such as indenting new paragraphs and leaving a little more space between sentences than the space between words. One quirk to be aware of is that comments "eat" any leading space on the following line, so

```
This% weird, right?
  is strange.
```

gives

This is strange.

Commands

LaTeX provides many commands to format your text, and you can also define your own! Commands always begin with a backslash (`\`), contain only letters, and are case-sensitive.* Some commands need more information, or *arguments*: `\documentclass`, for example, needs to know which class we want. Arguments are enclosed in consecutive pairs of braces, so if some command needed two arguments, we would type:

```
\somecommand{argument1}{argument2}
```

Many commands also take optional arguments. They precede the mandatory ones, are enclosed in square brackets, and are separated by commas. Say you want to print your document as double-sided pages† in 11 point type. We make these demands as optional `\documentclass` arguments:

```
\documentclass[11pt, twoside]{article}
```

*`\foo` is different from `\Foo`, for example.

†`twoside` introduces commands that only make sense for double-sided printing, like one that skips to the start of the next odd page. It also lets you have different margins for even and odd pages, which is useful for texts like this book.

Other commands take no arguments at all—`\LaTeX`, which prints the LaTeX logo, is one example. These commands consume any space that follows them. For example,

```
\LaTeX is great, but it can be a bit odd sometimes.
```

will give you

> LaTeXis great, but it can be a bit odd sometimes.

You can fix this with an empty pair of braces following the command. Of course, you don't need braces if there is no space to preserve:

```
Let's learn \LaTeX! \LaTeX{} is a powerful tool,
but a few of its rules are a little weird.
```

gets us

> Let's learn LaTeX! LaTeX is a powerful tool, but a few of its rules are a little weird.

Special characters and line breaks

Some characters have special meanings in LaTeX. We just saw, for example, that `%` starts a comment and `\` starts a command. The full list of special characters is:

```
# $ % ^ & _ { } ~ \
```

Each has a corresponding command for actually printing it in your document. Respectively, they are:

```
\# \$ \% \^{} \& \_ \{ \} \~{} \textbackslash
```

Regardless of whatever follows them, the caret (`^`) and tilde (`~`) always need braces. This is a relic from days when they produced *diacritical marks*: once upon a time, LaTeX users would typeset "jalapeño" with `jalape\~no`. Today we just type `ñ` into our source file.*

If you're wondering why we print `\` with `\textbackslash` instead of `\\`, it is because the latter is the command to force a line break.

```
Give me \\
a brand new line!
```

obeys:

Give me
a brand new line!

Use this power sparingly—deciding how to elegantly break paragraphs into lines is one of LaTeX's greatest skills.

Environments

We often format text in LaTeX by placing it in *environments*. These always start with `\begin{name}` and conclude with `\end{name}`, where `name` is that of the desired environment. Take the `quote` environment, which adds additional space on both sides of a block quotation:

```
Donald Knuth once wrote,
\begin{quote}
We should forget about small efficiencies,
say about 97\% of the time:
premature optimization is the root of all evil.
Yet we should not pass up our opportunities in
that critical 3\%.
\end{quote}
```

*Of course, this depends on your keyboard, your editor, and your language settings in your OS. We will talk more about languages and Unicode fun in chapter 11.

produces

> Donald Knuth once wrote,
>
> > We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.

Groups and scopes

Some commands change how LaTeX sets the text that follows them. `\itshape`, for example, *italicizes* everything that comes after it. To limit a command's influence to a certain area, surround it with braces.

```
{\itshape Sometimes we want italics}, but only sometimes.
```

becomes

> *Sometimes we want italics*, but only sometimes.

The braced region is called a *group*, and commands issued inside a group lose their power once it ends. Environments also create their own groups:

```
\begin{quote}
\itshape If I italicize a quote, the following text will
use upright type again.
\end{quote}
See? Back to normal.
```

typesets

> > *If I italicize a quote, the following text will use upright type again.*
>
> See? Back to normal.

You can also use groups to prevent spacing oddities with zero-argument commands: some prefer `{\LaTeX}` over `\LaTeX{}`.

4. Document Structure

Every LaTeX document is different, but all share a few common elements.

Preambles and packages

In the last chapter, you built your first document with:

```
\documentclass{article}

\begin{document}
Hello, World!
\end{document}
```

The space between `\documentclass` and the start of the `document` environment is called the *preamble*. Here we handle whatever setup we need, including importing packages. These add new commands, or modify the document in interesting ways. The ones in your LaTeX distribution come from the Comprehensive TeX Archive Network—or CTAN—at https://ctan.org.* You will also find package manuals there, so make it your first stop when learning how to use one.

To import a package, add a `\usepackage` command with its name as the argument. As a simple example, let's write a document with the `metalogo` package, which adds `\LuaLaTeX` and `\XeLaTeX`:

```
\documentclass{article}
\usepackage{metalogo}
```

*Curious readers might wonder what TeX is, and how it differs from LaTeX. The short answer is that TeX is the original program, and LaTeX is a set of common commands that were later built on top of it. A longer answer is at the end of this guide under Appendix A. We won't discuss how to use plain TeX here. That is for another book—The TeXbook.

```
\begin{document}
\XeLaTeX{} and \LuaLaTeX{} are neat.
\end{document}
```

should get you a PDF that reads

> XeLaTeX and LuaLaTeX are neat.

`\usepackage` accepts optional arguments and passes them to whatever code you are importing. The `geometry` package, for instance, takes your desired paper size and margins. For US Letter paper with one-inch margins, type:

```
\usepackage[
    letterpaper,
    left=1in, right=1in, top=1in, bottom=1in
]{geometry}
```

Arguments can be spaced however you like, so long as there are no empty lines between them.

Hierarchy

Authors often split their writing into sections to help readers navigate it. LaTeX offers seven different commands to break up your documents: `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph`, and `\subparagraph`. Issue the command where you want an area to start, providing its name as the argument. For example,

```
\documentclass{book}

\begin{document}

\chapter{The Start}
This is a very short chapter in a very short book.

\chapter{The End}
```

```
Is the book over yet?

\section{No!}
There's some more we must do before we go.

\section{Yes!}
Goodbye!
\end{document}
```

Some levels are only available in certain document classes—chapters, for example, only appear in books. And don't go too crazy with these commands. Most works just need a few levels to organize them.

These bits of structure are automatically numbered. The title of this chapter was produced with `\chapter{Document Structure}`, and LaTeX figured out that it was chapter 4.

On your own

As promised, this book won't try to be a comprehensive reference, but it *will* point you to places where you can learn more. We'll wrap up most chapters with some related topics that you can explore yourself.

Consider learning how to:

- Automatically start your document with its title, your name, and the date using `\maketitle`.
- Build a table of contents with `\tableofcontents`.
- Control section numbering with `\setcounter{secnumdepth}` or starred commands like `\subsection*{foo}`.
- Create cross-references with `\label` and `\ref`.
- Use KOMA Script, a set of packages that let you customize nearly every aspect of your document, from heading fonts to footnotes.
- Include images with the `graphicx` package.

- Add hyperlinks with the `hyperref` package.
- Split large documents into multiple source files using `\input`.

5. Formatting Text

Emphasis

Sometimes you need some extra punch to get your point across. The simplest way to emphasize text in LaTeX is with the `\emph` command, which *italicizes* its argument:

```
\emph{Oh my!}
```

gives us

Oh my!

We have other tools at our disposal:

```
We can also use \textbf{boldface} or \textsc{small caps}.
```

producing

We can also use **boldface** or SMALL CAPS.

Be judicious when you use emphasis, especially boldface, which excels at drawing the reader's attention away from everything around it. **Too much is distracting.**

Meeting the whole (type) family

Boldface and italics are just a few of the many styles you can use. A (mostly) complete list follows:

Command	Alternative	Style
`\textnormal{...}`	`{\normalfont ...}`	the default
`\emph{...}`	`{\em ...}`	*emphasis, typically italics*
`\textrm{...}`	`{\rmfamily ...}`	roman (serif) type
`\textsf{...}`	`{\sffamily ...}`	sans serif type
`\texttt{...}`	`{\ttfamily ...}`	`teletype (monospaced)`
`\textit{...}`	`{\itshape ...}`	*italics*
`\textsl{...}`	`{\slshape ...}`	*slanted, or oblique type*
`\textsc{...}`	`{\scshape ...}`	Small Capitals
`\textbf{...}`	`{\bfseries ...}`	**boldface**

Prefer the first form, which takes the text to format as an argument, over the second, which affects the group it is issued in. The former automatically improves spacing around the formatted text. For example, *italic type* amidst upright type should be followed by a slight amount of additional space, called an "italic correction". The latter is your only option when formatting multiple paragraphs or defining the style of other commands.*

Sizes

The font size of *body text*—that is, your main content—is usually ten points,† but can be adjusted by passing arguments to `\documentclass`.‡ To scale text relative to this default size, use the following commands:

`\tiny`	Example Text
`\scriptsize`	Example Text
`\footnotesize`	Example Text
`\small`	Example Text
`\normalsize`	Example Text
`\large`	Example Text

*For instance, this book's section headers are styled with `\Large\itshape`.

†The standard digital publishing point, sometimes called the PostScript point, is $1/72$ of an inch. LaTeX, for historical reasons, defines its point (`pt`) as $100/7227$ of an inch and the former as "big points", or `bp`. Use whichever you would like.

‡Stock LaTeX classes accept `10pt`, `11pt`, or `12pt` as optional arguments. KOMA Script classes accept arbitrary sizes with `fontsize=<size>`.

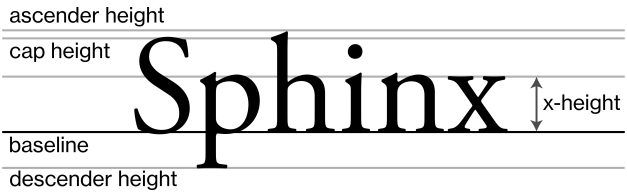
`\Large` Example Text
`\LARGE` Example Text
`\huge` Example Text
`\Huge` Example Text

If you look carefully, you will find some subtleties at play here. LaTeX's default type family, Latin Modern, comes in several *optical sizes*. Smaller fonts aren't just shrunken versions of their big siblings—they have thicker strokes, exaggerated features, and more generous spacing to improve legibility at their size.

> If I make 5 point type the same height as 11 point type, you can easily spot the differences.

Back when fonts were made out of metal, multiple optical sizes were standard. But many digital fonts only have one, since each optical size requires a great deal of careful design.*

Points and optical sizes don't tell the whole story. Each typeface has different proportions which affect its perceived size. (Compare Garamond, Latin Modern, Futura, and Helvetica, all at 11 points.) Shown below are some common terms:



Type sits on the *baseline*, rises to its *ascender height*, and drops to its *descender height*. The *cap height* refers to the size of uppercase letters, and the *x-height* refers to the size of lowercase letters.

If the previous commands don't give you a size you need, you can create custom ones with `\fontsize`, which takes both a text size and a distance between baselines. This must be followed with `\selectfont` to take effect. For example, `\fontsize{30pt}{30pt}\selectfont` produces

*If you have typefaces with multiple optical sizes, LuaLaTeX and XeLaTeX can make good use of them! See chapter 9 for more on font selection.

large type with no additional space between lines

Note how without some extra space, or *leading*,* descenders from one line almost collide with ascenders and capitals on the next. Leading is important—without it, blocks of text become uncomfortable to read, especially at normal body sizes. Let your type breathe![†]

On your own

- Learn how to underline text with the `ulem` package.[‡]
- Use KOMA Script to change the size and style of your section headings.
- Learn the difference between italic and oblique type.
- Change the default text style (used by `\textnormal` and `\normalfont`) by redefining `\familydefault`.

*This term comes from the days of metal type, when strips of lead or brass were inserted between lines to space them out.[1]

[†]For a discussion of how much leading to use, see *Practical Typography*, as mentioned in Appendix B.

[‡]Other typographical tools—like italics, boldface, and small caps—are generally preferable to underlining, but it has its uses.

6. Punctuation

You would rather encounter a panda that eats shoots and leaves than one that eats, shoots, and leaves.[2] Punctuation is a vital part of writing, and there's more to it than your keyboard suggests.

Quotation marks

LaTeX doesn't automatically convert `"straight"` quotes into correctly-facing "curly" ones:

```
"This isn't right."
```

will get you

"This isn't right."

Instead, use ` for opening quotes and ' for closing quotes.*

```
``It depends on what the meaning of the word `is' is.''
```

quotes a former US president as,

"It depends on what the meaning of the word 'is' is."

*If your keyboard happens to have keys for "curly" quotes (“ ”), feel free to use those instead! And don't use " for closing double quotes. Not only does ``example" look a bit unbalanced, but " is used as a formatting command for some languages, like German. (See chapter 11 for more on international typesetting.)

Hyphens and dashes

Though they look similar, hyphens (-), en dashes (–), em dashes (—), and minus signs (−) serve different purposes.

Hyphens have a few applications:[3]

- They allow words to be split across the end of one line and the start of the next. LaTeX does this automatically.
- Compound words like *long-range* and *field-effect* use hyphens.
- They are used in phrasal adjectives. If I ask for "five dollar bills", do I want five $1 bills, or several $5 bills? *Five-dollar bills* makes it clearer that I want the latter.

Unsurprisingly, you get one by typing the hyphen character (-).

En dashes are for ranges such as "pages 4–12", and connected words, like "the US–Canada border". LaTeX places one wherever you enter two adjacent hyphens (--).

Em dashes separate clauses of a sentence. Other punctuation—like parenthesis and commas—play a similar role. Em dashes are typeset with three hyphens (---).

Minus signs are for negative quantities and mathematical expressions. They are similar in length to an en dash, but sit at a different height. Minus signs are set with `\textminus`, or with the hyphen character when in a math environment (see chapter 8).

Ellipses

A set of three dots which indicate a pause or omission is called an *ellipsis*. It is set with `\dots`.

```
I'm\dots{} not sure.
```

becomes

I'm… not sure.

Ellipses have different spacing than consecutive periods. Don't use the latter as a poor substitute for the former.

Spacing

As we discovered in chapter 3, LaTeX inserts additional space between periods and whatever follows them—presumably the start of the next sentence. This isn't always what we want! Consider honorifics like Mr. and Ms., for example. In these situations, we also need to prevent LaTeX from starting a new line after the period. This calls for a *non-breaking space*, which we set with a tilde.

```
Please call Ms.~Shrdlu.
```

produces proper spacing:

Please call Ms. Shrdlu.

In other occasions, like when we abbreviate units of measurement,* we want thinner spaces than our usual inter-word ones. For these, we use `\,` :

```
Launch in 2\,h 10\,m.
```

announces

Launch in 2 h 10 m.

On your own

- Learn more commands for spacing, such as `\:`, `\;`, `\enspace`, and `\quad`.
- Nest quotations, e.g., "She exclaimed, 'I can't believe it!'" more easily with `csquotes` package's `\enquote` command.

*There are also dedicated packages for doing so, like `siunitx`.

- Discover the typographical origins of terms like *en*, *em*, and *quad*.
- Familiarize yourself with the difference between `/` and `\slash`.
- Add hyphenations for uncommon words using `\hyphenate` or `\-`.*

*LaTeX usually does a great job of automatically hyphenating words, based on a dictionary of patterns stored for each language. You should rarely need these commands.

7. Layout

Justification and alignment

LaTeX justifies text remarkably well. Instead of arranging it one line at a time—like most word processors, web browsers, and e-readers do—it considers every possible line break in a paragraph, then picks the ones that give the best overall spacing.[4] Combined with automatic hyphenation, which permits line breaks in the middle of words,[5] it can produce better paragraph layouts than almost any other software.

But sometimes we don't want justified text. If you would like it to be flush left, place it in a `flushleft` environment or add `\raggedright` to the current group. To center it, place it in a `center` environment or add `\centering` to the current group. And to flush it against the right margin, use a `flushright` environment or `\raggedleft`.

```
\begin{flushleft}
This text is flush left, with a ragged right edge.
Some prefer this layout since the space between words
is more consistent than it is in justified text.
\end{flushleft}

\begin{center}
This text is centered.
\end{center}

\begin{flushright}
And this text is flush right.
\end{flushright}
```

sets

This text is flush left, with a ragged right edge. Some prefer this layout since the space between words is more consistent than it is in justified text.

This text is centered.

And this text is flush right.

Lists

LaTeX provides several environments for creating lists: `itemize`, `enumerate`, and `description`. In all three, each item starts with an `\item` command. To create bulleted lists, use the `itemize` environment. With

```
\begin{itemize}
\item 5.56 millimeter
\item 9 millimeter
\item 7.62 millimeter
\end{itemize}
```

you get

- 5.56 millimeter
- 9 millimeter
- 7.62 millimeter

`enumerate` numbers its lists:

```
\begin{enumerate}
\item Collect underpants
\item ?
\item Profit
\end{enumerate}
```

which gives

1. Collect underpants
2. ?
3. Profit

The `description` environment starts each item with some emphasized label, then indents subsequent lines for that item:

```
\begin{description}
\item[Alan Turing] was a British mathematician who
    laid much of the groundwork for computer science.
    He is perhaps most remembered for his model of
    computation, the Turing machine.
\item[Edsger Dijkstra] was a Dutch computer scientist.
    His work in many domains---such as concurrency and
    graph theory---are still in wide use.
\item[Leslie Lamport] is an American computer scientist.
    He defined the concept of sequential consistency,
    which is used to safely communicate between tasks
    running in parallel.
\end{description}
```

gives us

Alan Turing was a British mathematician who laid much of the groundwork for computer science. He is perhaps most remembered for his model of computation, the Turing machine.

Edsger Dijkstra was a Dutch computer scientist. His work in many domains—such as concurrency and graph theory—are still in wide use.

Leslie Lamport is an American computer scientist. He defined the concept of sequential consistency, which is used to safely communicate between tasks running in parallel.

Columns

We often split pages into several columns, especially when printing on A4 or US Letter paper, since it provides more comfortable line widths with standard 8–12 pt text sizes.* You can either add the `twocolumn` option to your document class, which splits everything in two, or you can use the `multicols` environment from the `multicol` package:

```
One nice feature of the \texttt{multicol} package
is that you can combine arbitrary layouts.
\begin{multicols}{2}
This example starts with one column,
then sets the following section as two.
The \texttt{multicols} environment splits the text
inside it so that each column is about the same height.
\end{multicols}
```

arranges

> One nice feature of the `multicol` package is that you can combine arbitrary layouts.
>
> This example starts with one column, then sets the following section as two. The `multicols` environment splits the text inside it so that each column is about the same height.

Page breaks

Some commands, like `\chapter`, insert page breaks. You can add your own with `\clearpage`. When using the `twoside` document class option for double-sided printing, you can break to the front of the next page with `\cleardoublepage`.

*You will find different advice depending on where you look, but as a rule of thumb, aim for 45 to 80 characters (including spaces) per line. If a line is too long, readers have an uncomfortable time scanning for the start of the next one. If a line is too short, it doesn't have much inter-word spacing to adjust, which can lead to odd gaps or excessive hyphenation.

Footnotes

Footnotes are useful for references, or for remarks that readers might find helpful but aren't crucial to the main text. The `\footnote` command places a marker at its location in the body text, then sets its argument at the bottom of the current page:

```
I love footnotes!\footnote{Perhaps a bit too much\dots}
```

proclaims

> I love footnotes!*

On your own

- Control paragraph spacing with KOMA Script or the `parskip` package.
- Set the page size and margins with the `geometry` package.
- Customize list formatting with the `enumitem` package.
- Create tables with the `tabular` environment.
- Align text with tab stops using the `tabbing` environment.
- Customize footnote symbols and layout with the `footmisc` package and KOMA Script.
- Insert horizontal and vertical space with commands like `\vspace`, `\hspace`, `\vfill`, and `\hfill`.
- Learn what units LaTeX provides to measure space.
  (We've already mentioned a few here, like `pt`, `bp`, and `in`.)

*Perhaps a bit too much...

8. Mathematics

LaTeX excels at typesetting mathematics, both inside body text ($x_n^2 + y_n^2 = r^2$) and on their own lines:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

The former is typed inside `$...$` or `\(...\)`, and the latter within `\[...\]`. In these math environments, the rules of LaTeX change:

- Most spaces and line breaks are ignored. Spacing decisions are made for you based on typographical conventions for mathematics. `$x+y+z$` and `$x + y + z$` both give you $x + y + z$.
- Empty lines are not allowed—each formula occupies a single "paragraph".
- Letters are automatically italicized, as they are assumed to be variables.

To return to normal "text mode" inside a formula, use the `\text` command and friends. Standard formatting commands work in these blocks. From

```
\[ \text{fake formulas} = \textbf{annoyed mathematicians} \]
```

we get

$$\text{fake formulas} = \textbf{annoyed mathematicians}$$

Examples

Math typesetting is the raison d'être of LaTeX,* but we could take dozens of pages to just cover the basics. Given the breadth of modern mathematics, there are *many*

*Well, TeX

different commands and environments. You owe it to yourself to find some real references and learn what LaTeX is capable of. But before moving on, let's see some examples of what it can do.

1.
```
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
```

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

2.
```
e^{j \theta} = \cos(\theta) + j \sin(\theta)
```

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)$$

3.
```
\begin{bmatrix}
x' \\
y'
\end{bmatrix} =
\begin{bmatrix}
\cos \theta &  -\sin\theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
```

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

4.
```
\oint_{\partial \Sigma} \mathbf{E} \cdot
\mathrm{d}\boldsymbol{\ell}
= - \frac{\mathrm{d}}{\mathrm{d}t}
    \iint_{\Sigma} \mathbf{B} \cdot \mathrm{d}\mathbf{S}
```

$$\oint_{\partial\Sigma} \mathbf{E} \cdot \mathrm{d}\boldsymbol{\ell} = -\frac{\mathrm{d}}{\mathrm{d}t} \iint_{\Sigma} \mathbf{B} \cdot \mathrm{d}\mathbf{S}$$

On your own

- Number equations for later reference with the `equaiton` environment.
- Automatically size parenthesis and braces to fit their contents with `\left` and `\right`.
- Learn some of the many helpful features of the `amsmath` package from the American Mathematical Society, such as the `align` environment for lining up equivalent equations.

9. Fonts

Digital fonts have completely changed since LaTeX was created decades ago. LaTeX originally used METAFONT, a format Donald Knuth designed specifically for TeX. As time went on, support for PostScript* fonts was added. Today, LuaLaTeX and XeLaTeX support the modern font formats you will find on your computer: TrueType and OpenType.†

TrueType was developed by Apple and Microsoft in the late 1980s. Most fonts that come pre-installed on your system are probably in this format. TrueType files generally end in a `.ttf` extension.

OpenType was first released by Microsoft and Adobe in 1996. Improvements over TrueType include its ability to embed various features, such as alternative glyphs and spacing options, into a single file. OpenType files usually end in an `.otf` extension.

Changing fonts

By default, LuaLaTeX and XeLaTeX use Latin Modern, an OpenType rendition of LaTeX's original type family, Computer Modern. While these are high-quality fonts, they are not the only ones you ever want to use. For other fonts, we turn to the `fontspec` package:

```
\documentclass{article}

\usepackage{fontspec}
```

*One of Adobe's original claims to fame, PostScript is a language for defining and drawing computer graphics, including type. It remains in widespread use today.

†Mac versions of LaTeX also support Apple's AAT, but let's limit this discussion to more common formats.

```
\setmainfont[Ligatures=TeX]{Source Serif Pro}
\setsansfont[Ligatures=TeX]{Source Sans Pro}
\setmonofont{Source Code Pro}

\begin{document}
Hello, Source type family! Neat---no? \\
\sffamily Let's try sans serif! \\
\ttfamily Let's try monospaced!
\end{document}
```

should produce something like*

> Hello, Source type family! Neat—no?
> Let's try sans serif!
> Let's try monospaced!

The `Ligatures=TeX` option lets you use the punctuation shortcuts from chapter 6 (`--` for en dashes, ``` `` ``` and `''` for curly quotes, etc.) instead of forcing you to enter the corresponding characters, which probably aren't on your keyboard. You usually don't want these substitutions with monospaced type, though. Text that uses it—such as code—is often meant to be printed verbatim. `"Hello!"` should not become "Hello!".

Selecting font files

`fontspec` usually finds the files you need for a given typeface, especially if you just want the basic set of upright, *italic*, **bold**, and ***bold italic*** fonts. But typefaces can have many more than that. The version of Futura in this book, for example, comes in light, book, **medium**, **demi**, **bold**, and **extra bold** weights. Each of these weights has an oblique font, too. A typeface could have other variations, like SMALL CAPITALS† or multiple optical sizes (see chapter 5).

*Assuming, of course, that you have Adobe's open-source fonts installed.[6]

†OpenType allows some styles, like small caps, to be placed in the same file(s) as the "main" glyphs for a given weight. If your font supports this, `fontspec` will automatically switch to them whenever you use `\textsc` or `\scshape`. But for TrueType fonts, and for OpenType fonts that don't leverage this feature, you will have to specify separate files.

We might want to hand-pick weights to achieve a certain look, or to better match the other fonts in our document.* Continuing to use Futura as an example, say we want "book" as our default weight and "demi" for bold. Assuming the font files are named:

- `Futura-Boo` for upright book weight
- `Futura-BooObl` for *oblique book weight*
- `FuturaSC-Boo` for SMALL CAPS, BOOK WEIGHT
- `Futura-Dem` for **upright demi(bold)**
- `Futura-DemObl` for ***oblique demibold***

Our setup might resemble:

```
\usepackage{fontspec}
\setmainfont[
    Ligatures=TeX,
    UprightFont = *-Boo,
    ItalicFont = *-BooObl,
    SmallCapsFont = *SC-Boo,
    BoldFont = *-Dem,
    BoldItalicFont = *-DemObl
]{Futura}
```

Note that instead of typing out `Futura-Boo`, `Futura-BooObl`, and so on, we can use `*` to insert the base name.†

*Compare how the light, book, **and medium weights** of Futura look next to the rest of the type on this page.

†This is a place where XeLaTeX and LuaLaTeX differ. The former uses system libraries—such as FontConfig on Linux—to find font files. The latter has its own font loader, based on code from FontForge.[7] Because the two look for files in different ways, the expected name of a font might differ between the two engines. See the `fontspec` package manual for details.

Scaling

Creating a cohesive design with multiple fonts is tricky, especially since they might look completely different at the same point size. `fontspec` can help by scaling fonts to match either the x-height or the cap height* of your main font with `Scale=MatchLowercase` or `Scale=MatchUppercase`, respectively. But one way to sidestep this issue is to use fewer typefaces in the first place. Just one or two, used carefully, can produce amazing results.

OpenType features

As mentioned at the start of the chapter, OpenType fonts provide many features that can be switched on and off. In LaTeX, we do this with optional arguments to `\setmainfont` and friends. Features can also be set for the current group with `\addfontfeature`. Let's examine some common ones.

Ligatures

Many typefaces use *ligatures*, which combine multiple characters into a single glyph.† OpenType groups them into three categories:

Standard ligatures remedy spacing problems between certain characters. Consider lowercase f and i: in many typefaces, these combine to form the ligature fi—this avoids awkward spacing between f's ascender and i's dot (fi). Other common examples in English writing include ff, ffi, fl, and ffl. Standard ligatures are enabled by default.

Discretionary ligatures, such as ct, are offered by some fonts. They're disabled by default but enabled with `Ligatures=Discretionary`.

Historical ligatures are ones which have fallen out of common use, such as those with a *long s* (e.g., ft). These are also disabled by default but can be enabled with `Ligatures=Historic`.

*Refer back to chapter 5 for an explanation of these heights.

†Ligatures fell out of style during the 20th century due to limitations of printing technology and the increased popularity of sans serif typefaces, which often lack them. Today they are making a comeback, thanks in no small part to their support in OpenType.

Multiple options can be grouped together. Say you want discretionary ligatures. In the likely event that you also want `Ligatures=TeX`, you would enable both with `Ligatures={TeX,Discretionary}`. Ligatures can also be disabled with corresponding `*Off` options. If you want to stop using discretionary ligatures for some passage,

```
{\addfontfeature{Ligatures=DiscretionaryOff}...}
```

does the trick.

Some words arguably look better without ligatures—shelfful is a classic example.[8] You can manually prevent a ligature by inserting a zero-width space, e.g., `shelf\hspace{0pt}ful`. Or, since life is too short, you can let the `selnolig` package handle these cases for you.

Figures

When setting figures,* you have two choices to make: lining versus text, and proportional versus tabular. *Lining* figures, sometimes called *titling* figures, have heights similar to capital letters:

A B C D 1 2 3 4

Text, or *oldstyle* figures, share more similarities with lowercase letters:

Sitting cross-legged on the floor... 25 or 6 to 4?

Either choice is fine for body text, but don't mix capital letters with text figures. "F-15C" looks odd, as does "V2.3 Release".

The terms *proportional* and *tabular* refer to spacing. Tabular figures are set with a uniform width, so that 1 takes up the same space as 8. As their name suggests, this is great for tables and other scenarios where figures line up in columns:

Item	Qty.	Price
Gadgets	42	$5.37
Widgets	18	$12.76

*_Figures_ is typography-speak for what we might also call *digits* (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

Proportional figures are the opposite—their spacing is, well... *proportional* to the width of each figure. They look a bit nicer in body text: 1837 looks more natural here than 1837 does.

You select figures with the following options:

```
Numbers=  Lining / Uppercase
          OldStyle / Lowercase
          Proportional
          Tabular / Monospaced
```

Like ligature options, these can be combined: proportional lining figures are set with `Numbers={Proportional,Lining}`, and tabular oldstyle figures are set with `Numbers={Tabular,OldStyle}`. Each option also has a corresponding `*Off` variant.*

Finally, some fonts provide *superior and inferior* figures, which are used to set ordinals (1st, 2nd 3rd, ...), fractions ($^{25}/_{624}$), and so on. They have the same weight as the rest of the font's characters, offering a more consistent look than shrunken versions of full-sized figures. (Compare the examples above to their imposters: 1st, 2nd, 3rd, and $^{25}/_{624}$. Notice how this second set is too light compared to the surrounding type.) Superior figures are typeset with `VerticalPosition=Superior`, and inferiors are set with `VerticalPosition=Inferior`.

On your own

- Learn how `fontspec` can choose optical sizes based on point size, either automatically from ranges embedded in OpenType fonts, or manually using `SizeFeatures`.
- Experiment with letter spacing—or *tracking*—with the `LetterSpace` option. Extra tracking is unnecessary in most cases, but can be useful to make SMALL CAPS a little more READABLE.

*This is especially useful since different fonts have different defaults. Some fonts use lining figures by default and enable text figures with `Numbers=OldStyle`. Others default to text figures and require `Numbers=Lining`.

10. Microtypography

Microtypography improves text's legibility with small, subliminal tweaks. It is

> [...]the art of enhancing the appearance and readability of a document while exhibiting a minimum degree of visual obtrusion. It is concerned with what happens between or at the margins of characters, words or lines. Whereas the macro-typographical aspects of a document (i.e., its layout) are clearly visible even to the untrained eye, micro-typographical refinements should ideally not even be recognisable. That is, you may think that a document looks beautiful, but you might not be able to tell exactly why: good micro-typographic practice tries to reduce all potential irritations that might disturb a reader.[9]

In LaTeX, microtypography is controlled with the `microtype` package. Its use is automatic—for the vast majority of documents, you should add

```
\usepackage{microtype}
```

to your preamble and move on. But let's take a quick look at what the package actually does.

Character protrusion

By default, LaTeX justifies lines between perfectly straight margins. This is an obvious default, but falls victim to an annoying optical illusion: lines ending in small glyphs—like periods, commas, or hyphens—seem shorter than lines that don't.* `microtype` compensates by *protruding* these glyphs into the margins.

*Many other optical illusions come up in typography. For example, if a circle, a square, and a triangle of equal heights are placed next to each other, the circle and triangle look smaller than the square. For this reason, round or pointed characters (like O and A) must be slightly taller than "flat" ones (such as H and T) for all to appear the same height.[10]

Font expansion

To give paragraphs more even spacing and fewer hyphenated lines, `microtype` can stretch characters horizontally. You might think that distorting type like this would be immediately noticeable, but you're reading a book that does it on every page! This effect, called *font expansion*, is applied *very* slightly—by default, character widths are altered by no more than two percent.*

This feature isn't currently available for XeLaTeX. If you want to use it, you need LuaLaTeX.

On your own

As always, see the package manual for ways to tweak these features. `microtype` has a few other tricks, but several only work on older LaTeX engines.† Those we care about—such as letterspacing—can be handled with `fontspec` or other packages.

*Of course, you can use package options to change this limit, or disable the feature entirely.
†i.e., pdfTeX

11. Typographie Internationale

Surprisingly, languages besides English exist. You may want to write with them.

Unicode

Digitizing written language is a complicated topic that has evolved significantly since LaTeX's inception. Today, most software uses Unicode to represent text. Briefly,

- A Unicode text file is a series of *code points*. Each represents a character to be drawn, an accent or diacritical mark to combine with an adjacent character, or formatting information, such as an instruction to print subsequent text right-to-left.
- One or more of these code points combine to represent a *grapheme cluster* or *glyph*, the shapes within fonts that we informally call "characters".

Приве́т नमस्ते

How many characters do you see? How many code points?

- Modern font formats contain tables which map code points to the glyphs the file contains.

LuaLaTeX and XeLaTeX are Unicode-literate and play well with Unicode text files.* Make sure that the fonts you select contain the glyphs you need—many only support Latin languages.

*LuaLaTeX accepts UTF-8 files. XeLaTeX also accepts UTF-16 and UTF-32.

Polyglossia

When your document contains languages besides English, consider using the `polyglossia` package. It will automatically:

- Load language-specific hyphenations and other conventions.
- Switch between user-specified fonts for each language.
- Translate document labels, like "chapter", "section", and so on.
- Format dates according to language-specific conventions.
- Format numbers in languages that have their own numbering system.
- Use the `bidi` package for documents with languages written right to left.
- Set the script and language tags of OpenType fonts that have them.

To use `polyglossia`, specify your document's main language, along with any others it uses. Some languages also take regional dialects as an optional argument:

```
\usepackage{polyglossia}
\setdefaultlanguage[variant=american]{english}
\setotherlanguage{french}
```

Once set up, `polyglossia` defines environments for the requested languages. Each automatically applies their language's conventions to the text within. French, for example, places extra space around punctuation, so

```
Dexter cried,
\begin{french}
«Omelette du fromage!»
\end{french}
```

gives

> Dexter cried, « Omelette du* fromage ! »

*Yes, it's *omelette au fromage*. Direct all complaints to Genndy Tartakovsky.

On your own

- See the `polyglossia` manual for language-specific commands.
- Look into the `babel` package as an alternative to `polyglossia`.*
- Try typesetting Japanese or Chinese with the `xeCJK` or `luatex-ja` packages.

*`polyglossia` has better support for OpenType font features via `fontspec`. However, it is newer and has a few known bugs. `babel` is a fine substitute if you run into trouble.

12. When Good Type Goes Bad

With luck, you're off to a solid start with LaTeX. But as with any complicated tool, you will eventually run into trouble. Here are some common problems and things you can try to fix them.

Fixing overflow

When LaTeX can't break a paragraph into well-spaced lines, it gives up and overflows into the margin. You can sometimes remedy this with some "emergency stretch". If you add `\emergencystretch=<width>` to the preamble, LaTeX will try to set troublesome paragraphs a second time, stretching or shrinking the total space in each line by up to the provided width.* If that still doesn't help, tweak the wording of problematic paragraphs. This can be frustrating, but the alternative is for LaTeX to create spacing that is too loose—where words have large gaps between them—or too tight, wherewordsareawkwardlycrammedtogether.

Avoiding widows and orphans

Good layouts avoid *widow* and *orphan* (also called *club*) lines: ones that get separated from the rest of their paragraph by a page boundary. LaTeX tries to prevent these, but its page-splitting algorithm is much more primitive than its paragraph-splitting one.† You can make LaTeX try harder to avoid orphans and widows with:

*LaTeX has pretty sane defaults for how much it stretches and shrinks spacing. You probably don't want to make `<width>` larger than an em or two.

†This is because 1980s computers didn't have enough RAM to do so. Seriously—Knuth wrote at the time, "The computer doesn't have enough high-speed memory capacity to remember the contents of several pages, so TeX simply chooses each page break as best it can, by a process of 'local' rather than 'global' optimization."[11]

```
\widowpenalty=<penalty>
\clubpenalty=<penalty>
```

`<penalty>` is a value between 0 and 10000. When these values are maximized, LaTeX is never allowed to leave orphans or widows, at *any* cost.* This can produce some really odd layouts, so be sure to review your pages if you choose large penalties.

Handling syntax errors

If you confuse LaTeX—say, by issuing commands that don't exist, or forgetting to end an environment—it will print an error message,† then display an interactive prompt starting with `?`. Here you can enter instructions for how to proceed. Once upon a time, when computers were thousands of times slower and LaTeX took that much longer to re-run, this was more useful. Today, we probably just want to quit, then try again once we've fixed our document. To exit the prompt, type `X`, then press Enter. Better yet, you can tell LaTeX to give up as soon as it finds trouble by running your engine with the `-halt-on-error` flag:

```
$ lualatex -halt-on-error myDocument.tex
```

Good luck!

*When considering a given layout, LaTeX assigns penalties, or "badness", to anything that arguably makes a document look worse. It chooses whichever layout it can find with the least badness.

†Usually this contains a succinct summary of the problem and the number of the line(s) it occurred on. Occasionally, LaTeX gets *really* confused and emits something so cryptic it gives C++ template errors a run for their money. As you use LaTeX, you will start to get a feel for what sorts of mistakes cause these rare, but enigmatic messages.

A. A Brief History of LaTeX

Donald Knuth is a father of computer science, famous for pioneering the *analysis of algorithms* and for his ongoing magnum opus, *The Art of Computer Programming*.

When the first volume of TAOCP was released in 1968, it was printed the way books had been since the turn of the century: with *hot metal* type. Letters were cast from molten lead, then arranged into lines. These lines were clamped together to form pages, which were inked and pressed against paper.

In March of 1977, Knuth was ready for a second run of TAOCP, volume 2, but he was horrified when he received proofs. Working with hot metal was expensive, complicated, and time-consuming, so publishers—including Knuth's—switched to phototypesetting, a process that projects characters onto film instead. The new technology, while cheaper and faster, didn't provide the quality Knuth expected.[12]

The average author would have resigned themselves to this change and moved on, but Knuth took great pride in his books' typography, especially when it came to their mathematics. Inspired by his recent introduction to the growing field of digital typesetting, Knuth set off on one of the greatest yak shaves* of all time. Setting aside his other work, he hammered on this problem for more than a year. When the dust settled in 1978, Knuth introduced the world to TeX.†

It's hard to appreciate how much of a revolution TeX was, especially looking back from today, where anybody with a web browser can be their own desktop publisher. Adobe's PDF wouldn't exist for another decade, so Knuth and his graduate students devised their own device-independent document format, DVI. Scalable fonts were uncommon, so he created METAFONT to rasterize glyphs into dots on the page. Perhaps most importantly, Knuth and his students designed algorithms to automatically hyphenate and justify text into beautifully-typeset paragraphs.‡

LaTeX, short for Lamport TeX, was developed by Leslie Lamport in the early 1980s as a set of commands for common document layouts. He introduced it with his guide, *LaTeX: A Document Preparation System*, in 1986. Development

*Programmers call seemingly-unrelated work needed to solve their main problem "yak shaving". The phrase is thought to originate from an episode of *The Ren & Stimpy Show*.[13]

†The name "TeX" comes from the Greek τέχνη, meaning *art* or *craft*.[14]

‡These same algorithms went on to influence the ones Adobe uses in its software today.[15]

continues today, both in the form of user-provided packages for TEX and LATEX, and as improvements to the TEX typesetting program itself. There are four versions, or *engines*:

TEX is the original system by Donald Knuth. Knuth stopped adding features after version 3.0 in March 1990—subsequent releases contain only bug fixes. With each release, the version number asymptotically approaches π by adding an additional digit. The most recent version, 3.141592653, came out in 2021.

pdfTEX is an extension of TEX that provides direct PDF output, native support for PostScript and TrueType fonts, and micro-typographic features discussed in chapter 10. It was originally developed by Hàn Thế Thành as part of his PhD thesis for Masaryk University in Brno, Czech Republic.[16]

XƎTEX is a further extension of TEX that adds native support for Unicode and OpenType. It was originally developed by Jonathan Kew in the early 2000s, and gained full cross-platform support in 2007.[17]

LuaTEX is similar to XƎTEX in its native Unicode and modern font support. It also embeds the Lua scripting language into the engine, exposing an interface for package and document authors. It first appeared in 2007, developed by a core team of Hans Hagen, Hartmut Henkel, Taco Hoekwater, and Luigi Scarso.[18]

Building TEX today is an… interesting endeavor. When it was written in the late 1970s, there were no large, well-documented, open-source projects for computer science students to study, so Knuth set out to make one. As part of this effort, TEX was written in a style he calls *literate programming*: opposite most programs—where documentation is interspersed throughout the code—Knuth wrote TEX as a book, with the code inserted between paragraphs. This mix of English and code is called `WEB`.*

Unfortunately, modern systems don't have good tooling for the 1970s dialect of Pascal that TEX was written in, so present-day distributions use another program, `web2c`, to convert its `WEB` source into C code. pdfTEX and XƎTEX are built by combining the result with other C and C++ sources. As an alternative to this complicated approach, the LuaTEX authors hand-translated Knuth's Pascal into C. They have used the resulting code since 2009.[19]

*Knuth also released a pair of companion programs named `TANGLE` and `WEAVE`. The former extracts the book—as TEX, of course—and the latter produces TEX's Pascal source code.

B. Additional Resources

For LaTeX

As promised from the start, this book is incomplete. To keep it short, major LaTeX features—like figures, captions, tables, graphics, and bibliographies—haven't been discussed. Use some of these resources to fill in the gaps:

The LaTeX Wikibook, at https://en.wikibooks.org/wiki/LaTeX

The Not So Short Introduction to LaTeX,
available at https://www.ctan.org/tex-archive/info/lshort/english/

The ShareLaTeX knowledge base, at https://www.sharelatex.com/learn

The TeX Stack Exchange, at https://tex.stackexchange.com/

For typography

We've spent most of our time here focusing on *what* you can do with LaTeX, and little on *how* you should use it to create well-designed documents. Read on:

Practical Typography, by Matthew Butterick.
Available (for free!) at https://practicaltypography.com

Stop Stealing Sheep & Find Out How Type Works, by Erik Spiekermann

Thinking With Type, by Ellen Lupton

Shaping Text, by Jan Middendorp

The Elements of Typographic Style, by Robert Bringhurst

Detail in Typography, by Jost Hochuli

Notes

1. Jan Middendorp, *Shaping Text* (Amsterdam, 2014), 71
2. Lynne Truss, *Eats, Shoots & Leaves* (New York, 2003)
3. Matthew Butterick, "Hyphens and dashes", *Practical Typography*, https://practicaltypography.com/hyphens-and-dashes.html
4. Donald E. Knuth and Michael F. Plass, *Breaking Paragraphs Into Lines* (Stanford, 1981)
5. Franklin Mark Liang, *Word Hy-phen-a-tion by Com-put-er* (Stanford, 1983), http://www.tug.org/docs/liang/
6. Adobe's open-source typefaces are freely available at https://github.com/adobe-fonts
7. *LuaTEX Reference* (Version 1.0.4, February 2017), 10
8. Knuth, *The TEXbook*, (Addison-Wesley, 1986), 19
9. R Schlicht, *The microtype package* (v2.7a, January 14, 2018), 4
10. Jost Hochuli, *Detail in typography* (Éditions B42, 2015), 18–19
11. Knuth, *The TEXbook*, 110
12. Knuth, *Digital Typography* (Stanford, 1999), 3–5
13. "yak shaving", *The Jargon File*, www.catb.org/~esr/jargon/html/Y/yak-shaving.html
14. Knuth, *The TEXbook*, 1
15. Several sources (http://www.tug.org/whatis.html, https://tug.org/interviews/thanh.html, http://www.typophile.com/node/34620) mention TEX's influence on

the *hz*-program by Peter Karow and Hermann Zapf, thanks to Knuth's collaborations with Zapf. *hz* was later acquired by Adobe and used when creating InDesign's paragraph formatting systems.

16. Hàn Thế Thành, *Micro-typographic extensions to the TEX typesetting system* (Masaryk University Brno, October 2000)

17. Jonathan Kew, "XeTeX Live", *TUGboat* 29, no. 1 (2007)

18. http://www.luatex.org

19. Taco Hoekwater, *LuaTEX says goodbye to Pascal* (MAPS 39, EuroTEX 2009), https://www.tug.org/TUGboat/tb30-3/tb96hoekwater-pascal.pdf

Colophon

This guide was typeset with LuaLaTeX in Robert Slimbach's Garamond Premier, a revival of roman type by 16th century French punchcutter Claude Garamond. Italics are inspired by the work of Garamond's contemporary Robert Granjon.

Monospaced items are set in Drive Mono, designed by Elliott Amblard and Jérémie Hornus at Black Foundry.

Captions use Neue Haas Grotesk, a Helvetica restoration by Christian Schwartz. Earlier digitizations of the ubiquitous Swiss typeface are based on fonts made for Linotype and phototypesetting machines, resulting in digital versions with needless compromises and kludges from the two previous generations of printing technology. Schwartz based his work on Helvetica's original drawings, producing a design faithful to the original cold metal type.

URW Futura makes a few guest appearances. Designed by Paul Renner and first released in 1927, Futura has found itself almost everywhere, from advertising and political campaigns to the moon. Douglas Thomas's recent history of the typeface, Never Use Futura, is a fantastic read.

Various bits of non-Latin text are set in Noto, a type family by Google that covers *every* glyph in the Unicode standard.

Finally, Latin Modern—the OpenType version of Knuth's Computer Modern used throughout the book—and TeX Gyre Termes—the free alternative to Times Roman seen on page 1—are the work of Grupa Użytkowników Systemu TeX, the Polish TeX Users' Group. Overviews of these excellent projects can be found on the GUST website:
http://www.gust.org.pl/projects/e-foundry/latin-modern
http://www.gust.org.pl/projects/e-foundry/tex-gyre.