Modern

LaTeX

Matt Kline

Some crappy draft, typeset February 14, 2018.

The author apologizes for any typos, misattributions, or other flubs, and welcomes you to point them out on this book's Github repository at
`https://github.com/mrkline/latex-book`, or via email to
`matt <at> bitbashing.io`
The author does not have a checking account with the Bank of San Serriffe, but will happily purchase a beverage of your choice the next time we meet.

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

Contents

1. Typography and You

Modern life is a constant battle for your time—every day, dozens of ads, apps, clips, emails, sites, shows, and texts fight for a few short minutes of it. To put ideas into the world, nothing is more important than catching and holding the attention of your audience. When we write, we don't just reach readers with the words we use. We reach them with *typography*: how those words appear. Good design doesn't "look nice" only for the sake of art—it draws readers in.[1] It sets their expectations and establishes a subliminal brand for your work.[2]

> Typography is why this reminds you of terrible essays you wrote in school. Do you see many books that look like this? Why do you think that might be?

It is why

> **DO IT LATER**

seems oddly reminiscent of a certain shoe company's advertising, or how

MEN FROM THE PLANET EARTH
FIRST SET FOOT UPON THE MOON
JULY 1969, A. D.

If you care about more than *what* you tell your readers, but *how* you tell it, you should try LaTeX.[*] It's a program[†] for crafting written documents, like papers, presentations, or even the book you're reading right now. And unless you want to give Adobe large sums of money for InDesign or InCopy, you won't find any better typesetting software. By carefully handling subtle details, LaTeX provides high-quality layout with relatively little effort from you, the author. Modern versions can also take advantage of new[‡] advances in computer typography, giving you the same tools used by professional designers and publishers.

[*] Pronounced "lay-tech" or "lah-tech"

[†] And a markup language, and (sort of) a programming language, and a bit of a cult. But that's all for later.

[‡] By new, I mean "from the mid-1990s", but web browsers and desktop publishing software are only just starting to catch up.

LaTeX?

LaTeX is an alternative to “word processors” like Microsoft Word, Apple Pages, Google Docs, and LibreOffice Writer. Unlike these other applications, which work on the principle of *What You See Is What You Get* (WYSIWYG), a LaTeX document is written as a “plain” text file, using *markup* to specify how things should look. LaTeX then takes this source file and builds your document based on the formatting rules you give it. If you’ve done any web development, this is a similar process—just as HTML and CSS describe the page you want browsers to draw, your markup describes the appearance of your document to LaTeX.

```
\LaTeX{} is an alternative to ``word processors'' like
Microsoft Word, Apple Pages, Google Docs, and LibreOffice Writer.
Unlike these other applications, which work on the principle of
\introduce{What You See Is What You Get} \acronym{(wysiwyg)},
a \LaTeX{} document is written as a ``plain'' text file,
using \introduce{markup} to specify how things should look.
\LaTeX{} then takes this source file and builds your document
based on the formatting rules you give it.
If you've done any web development, this is a similar process---just
as \acronym{html} and \acronym{css} describe the page you want
browsers to draw, your markup describes the appearance of your
document to \LaTeX.
```

The LaTeX markup for the above paragraph

This might seem alien to you if you’ve never worked with a markup system before. However, it comes with a few advantages:

1. You can handle a document’s contents and its presentation separately. At the beginning of each document, you describe the design you want. LaTeX takes it from there, keeping fonts, sizes, layout, and other formatting considerations consistent across your whole text. Compare this to a WYSIWYG system, where you must constantly concern yourself with your work’s appearance as you write. If you changed the look of a caption, did you make sure to find all the other captions and do the same? If the program formats something in a way you don’t like, how hard is it to fix?

2. You can define your own rules, then tweak them to immediately change everywhere they’re used in your document. For example, the `\introduce` and `\acronym` commands you saw above are my own creations. The former *italicizes* text, and the latter sets words in SMALL CAPS with a bit of extra LETTERSPACING so the characters don’t look TOO CROWDED. If I wake up tomorrow and decide to introduce new terms ***with this look*** and set acronyms LIKE THIS, I just change the two lines that define those commands to update every place in the book that uses them.

3. Since the document source is plain text,
    - It can be read and understood with any text editor.
    - A document's structure is immediately visible and easily replicated.*
    - Content can be generated programmatically.
    - You can track changes with standard version control software.

Another guide?

One might wonder why the world needs another LaTeX guide. After all, it's been out for more than 30 years. A quick Amazon search finds nearly a dozen books on the topic. There are plenty of great resources online.

Unfortunately, most guides have two fatal flaws: they are long, and they are old. The first is anathema to newcomers—if somebody asks about LaTeX, throwing a 200+ page book at them isn't an encouraging start. The second matters because of how much the world of computer typesetting has changed since 1986.

When LaTeX was first released that year, none of the publishing technologies we use today existed. Adobe wouldn't introduce their Portable Document Format for another seven years. Digital typography was a new field, and desktop publishing was *just* getting off the ground.[3] This shows—badly—in most LaTeX guides. If you look for instructions to change your document's font, you'll likely get swamped with bespoke nonsense.†

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 to be a comprehensive reference. After all, you're a smart, resourceful individual who knows how to use a search engine. This book will:

1. Teach you the very basics of using LaTeX.
2. Point you to places where you can learn more.
3. Show you how to use modern technologies like OpenType and microtypography to create professional-quality documents.
4. End promptly thereafter.

Let's begin.

*Compare this to WYSIWYG systems, where it is often unobvious how certain formatting was produced or how to replicate it.

†Take all criticisms of LaTeX's past here with a grain of salt. After all, the fact that all of the technology around it became obsolete—multiple times—is a testament to its staying power.

2. Installation

You install LaTeX on your computer by downloading a *distribution*, which comes with:

1. LaTeX, the program—the thing that turns text files into nicely-typeset documents.[*]

2. A common set of LaTeX *packages*. Packages are pieces of code that can be placed in your documents to do all sorts of things, like add new commands or change their style. We'll see lots of them in action throughout this book.

3. Miscellaneous tools, such as editors.

Each major operating system has its own LaTeX distribution:

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

Windows has MikTeX. Install it from `https://miktex.org/download` as described by the tutorial. MikTeX offers the unique ability to automatically find and download additional packages if your document uses one that isn't currently on your computer.

Linux and BSD use TeX Live. Like most software, this is often provided through your OS's package manager. Use yours to install `texlive-core`, `texlive-luatex` and `texlive-xetex`.[†] As you work with LaTeX, you may need less-common packages, which are usually found in distro packages with names like `texlive-latexextra`, `texlive-science`, and so on.[‡]

Editors

LaTeX source files are regular text files, so you're free to create them with the usual choices of Vim, Emacs, Gedit, Sublime, Notepad++, and so on.[◊] However, there are also editors designed specifically for LaTeX, which often come with their own built-in PDF viewer. You can find a fairly comprehensive list on the LaTeX Wikibook, in its installation chapter. (See Appendix B.)

[*] We'll actually be installing multiple LaTeX programs, but we're getting to that.

[†] Some Linux distributions (like Arch) package all three into a single `texlive-core` package.

[‡] Of course, package names, and how LaTeX packages are split between distro packages, may vary from one Linux distribution to another. These names are some of the most common.

[◊] If you have never used any of these, you owe it to yourself to try a few. They're very popular among 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 a LaTeX distribution installed, let's try it out. Open up your editor of choice 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 places several different versions—or *engines*—on your machine, but for this entire book, we'll always use LuaLaTeX or XeLaTeX. These are the newest engines available—see Appendix A for an explanation of how they differ from the others.

If you are using a LaTeX-specific editor, it may contain a drop-down menu or some other configuration to select the engine you'd like to use, as well as a button to generate your document. Otherwise, from a terminal,[†] run the following:

```
$ xelatex hello.tex
```

Feel free to use `lualatex` instead—there are a few differences between the two, 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.
```

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

Hello, World!

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

[†] How to work a terminal emulator, making 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 examples just denotes a console prompt, and shouldn't actually be typed.

Congrats, you just created your first document!

Let's unpack what we did. All LaTeX documents begin with a `\documentclass` declaration, 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 your average document, `article` is probably a good choice. The next line, `% Say hello`, is a *comment*—anything placed after a percent symbol on a line is ignored by the engine, so we use `%` to leave notes about what's going on to anybody reading the document's source.[†] Finally, we use `\begin{document}` to tell LaTeX that what follows is our actual contents, and we use `\end{document}` to state that we are finished.

Let's cover some more basics.

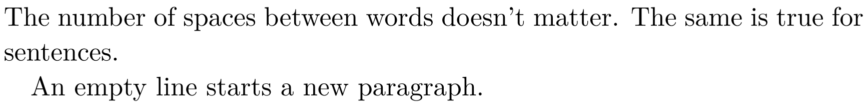
Spacing

LaTeX generally handles inter-word spacing for you, regardless of how many spaces[‡] you type. For example,

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

An empty line starts a new paragraph.
```

yields



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

An empty line starts a new paragraph.

Notice that LaTeX automatically follows typographic conventions, such as indenting new paragraphs and leaving more space after a period than it leaves between words. One quirk to be aware of is that comments "eat" all of the leading space on the subsequent line, such that

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

gives

> Thisis strange

[*] This last one is for slideshows. Kind of an odd name, no?
[†] Including, perhaps most importantly, a confused version of your future self!
[‡] Or tabs

Commands

LaTeX provides various commands to issue instructions to the engine, and you can define your own as well. Their names always begin with a backslash (\ ), contain only letters, and are case-sensitive.[*] Some commands require parameters, or *arguments*—`\documentclass`, for example, needs to know which class we want. Arguments are enclosed in subsequent pairs of braces, so if some command needed two arguments, we would type:

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

Many commands also take optional arguments, which are enclosed in square brackets and precede the mandatory ones. Say you want to inform LaTeX that your document will be printed as double-sided pages.[†] This is done with an optional argument to `\documentclass`:

```
\documentclass[twoside]{article}
```

Other commands take no arguments at all, such as `\LaTeX`, which prints the LaTeX logo. Know that these commands consume any space that follows them. For example,

```
\LaTeX is great, but it can be strange sometimes.
```

will give you

> LaTeXis great, but it can be strange sometimes.

You can fix this by adding an empty pair of braces to the command. Of course, the braces aren't needed 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.

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

[†] `twoside` introduces commands that only make sense in the context of double-sided printing, such as ones that skip to the start of the next odd page. It also allows you to have different margins for even and odd pages, which is useful for texts like this book.

Special characters and line breaks

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

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

Each has a corresponding command to print it in your document:

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

will produce

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

Regardless of what comes after them, you must always add braces to the caret (^) and tilde (~). This is a relic from days when these commands were also used to produce *diacritical marks*: once upon a time, users had to typeset "jalapeño" as `jalape\~no`. Today, we just type ñ into our source file.*

If you're wondering why we print \ with `\textbackslash` instead of just `\\`, it's 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 judiciously—deciding how to break paragraphs into lines automatically is one of LaTeX's best skills.

Environments

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

*The ease with which you can do this depends on the keyboard you're using, the language settings in your OS, and your editor. Later on, we'll talk much more about non-English languages and Unicode fun.

```
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}
```

quotes

> 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 command scope

Some commands change how LaTeX typesets the following text. `\itshape`, for example, *italicizes* everything that comes after it. To limit a command's effect to a certain region, we surround with braces.

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

is set as

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

The text within the braces forms a *group*, and all commands issued inside the group only take effect until it ends. Environments also form implicit groups:

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

produces

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

One other use of groups is to get around the spacing oddities of zero-argument commands: some prefer `{\LaTeX}` over `\LaTeX{}`.

4. Document Structure

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

Packages and the preamble

In the last chapter, you saw:

```
\documentclass{article}

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

The space between the `\documentclass` command and the start of the `document` environment is called the *preamble*. Here, we perform any setup we need—such as importing packages and defining commands—to control how our document will look. As mentioned briefly in Chapter 2, *packages* are bits of code that modify your document in interesting ways.

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

```
\documentclass{article}

\usepackage{metalogo}

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

should get you a PDF that reads

> XƎLaTeX and LuaLaTeX are neat.

Like other commands, `\usepackage` accepts optional arguments. The `geometry` package, for instance, takes your desired paper size and margins. For US letter paper with one-inch margins, you type:[*]

[*] Notice that command arguments can be spaced in whatever way is most convenient to you, so long as there are no empty lines between arguments.

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

Many packages are installed as part of your LaTeX distribution, Those you can't find locally are on the Comprehensive TeX Archive Network, or CTAN,* at `https://ctan.org`. The manuals for packages are also found there, so it should be your first stop when learning how to use one.

Titles, sections, and tables of contents

Authors often create hierarchy to help readers navigate their work. LaTeX provides seven different commands to break apart your document: `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph`, and `\subparagraph`. To use one, issue the command where you want that section to start, using the section's 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 may not be available depending on the document class you've chosen. Parts and chapters, for example, only appear in books. And don't go too crazy with these commands. Most works only need a few levels of hierarchy.

Sections† are automatically numbered—for example, the title of this chapter was produced with `\chapter{Document Structure}`, and LaTeX automatically determined that it was chapter 4. You can also have LaTeX automatically generate a table of contents with the `\tableofcontents` command.

*Curious readers may be wondering what TeX is, and how it differs from LaTeX. The short version is that TeX is the typesetting system that LaTeX is built on top of—the latter is a framework of commands for the former. (For example, `\documentclass` and friends are provided by LaTeX, but the TeX engine is what's actually laying out your document.) The long version is at the back of this book as Appendix A. We won't discuss how to use plain TeX here. That's for another book—the TeXbook.

†By this I mean all levels, not just `\section`.

What next?

As promised, this book isn't a comprehensive reference, but it *will* point you to places where you can learn more. We'll wrap up most chapters with a list of related topics you could explore next.

Consider learning how to:

- Automatically start your document with its title, the author's name, and the date using `\maketitle`.
- Control section numbering with `\setcounter{secnumdepth}` and "starred" section commands, e.g., `\subsection*{foo}`.
- Create auto-updating cross-references with `\label` and `\ref`.
- Use KOMA Script, a set of document classes and packages that make it easy to customize nearly every aspect of your document, from section heading fonts to footnotes.
- Include images using the `graphicx` package.
- Add hyperlinks to your PDF using the `hyperref` package.
- Split large documents into multiple 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 by default:

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

gives us

Oh my!

There are other tools at our disposal:

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

with

We can use **boldface** or SMALL CAPS as well.

Be judicious in your use of emphasis, especially boldface. It excels at drawing the reader's attention away from everything around it, so too much is distracting.

Meeting the whole (type) family

Italics, boldface, and small caps are just a few of the many styles available to you. 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**

The first form (which takes the text to format as an argument) should usually be preferred over the second (which affect the group in which they are issued), since the former automatically handle any spacing corrections needed around them.[*] However, there are cases where the second variety is the only option:

1. When formatting multiple paragraphs.
   (Command arguments cannot contain paragraph breaks.)

2. When defining the style of other commands.[†]

Sizes

The default text size is controlled by your document class. It is usually ten points,[‡] but this can be adjusted by passing additional arguments to `\documentclass`.[◊] To scale text relative to this size, use the following commands:

`\tiny`	Example Text
`\scriptsize`	Example Text
`\footnotesize`	Example Text
`\small`	Example Text
`\normalsize`	Example Text
`\large`	Example Text
`\Large`	Example Text
`\LARGE`	Example Text
`\huge`	Example Text
`\Huge`	Example Text

There is some subtlety here that you may not have noticed. LaTeX's default type family, Latin Modern, comes in multiple *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 you magnify 5 point type and place the result next to normal 10 point type, the differences are immediately noticeable.

[*] For example, *italic type* amidst upright type should be followed by a slight amount of extra space, called an "italic correction".

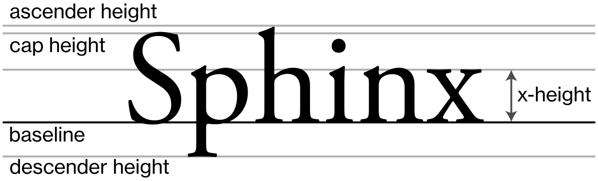
[†] For example, this book's section headers are styled with `\setkomafont{section}{\Large\itshape}`.

[‡] 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 a "big point", or `bp`.

[◊] The standard LaTeX classes accept `10pt`, `11pt`, or `12pt`. KOMA Script classes accept arbitrary sizes with `fontsize=<size>`.

Since this requires much more work from the type designer, many digital typefaces—even professional ones—lack this feature.[*]

But points and optical sizes don't tell the whole story. Each typeface has its own proportions, which make a huge difference in perceived size. (Compare Garamond, Latin Modern, and Helvetica, all at 10 pt.) Shown below are some common terms:



Type sits on the *baseline*, rises to the *ascender height*, and drops to the *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 aren't enough, you can create custom sizes with `\fontsize`, which takes both the desired point size and the distance between baselines. `\fontsize` must be followed with `\selectfont` to take effect. For example, `\fontsize{30pt}{30pt}\selectfont` gives:

large type with no extra space between lines

Note that without extra spacing, or *leading*,[†] descenders from one line nearly collide with ascenders and capitals on the line below. Leading is important—without it, blocks of text become uncomfortable to read, especially at normal body sizes.
Let your type breathe.[‡]

What next?

- Learn how to underline text using the `ulem` package.[◊]
- Use KOMA Script to change the size and style of your section headings.
- Learn the difference between italic and oblique type.

[*] If you are fortunate enough to have a typeface with multiple optical sizes, XƎLaTeX and LuaLaTeX can make good use of them! See Chapter 9 for more on font selection and OpenType features.

[†] This term comes from the days of metal type, when strips of lead were inserted between lines to give them extra spacing.

[‡] For a discussion on optimal leading, see *Practical Typography*, listed 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 much rather encounter a panda that eats shoots and leaves than one that eats, shoots, and leaves.[4] 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.”

Hyphens and dashes

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

Hyphens have a few uses:[5]

- They allow a word to be split between the end of one line and the start of the next. LaTeX usually handles this automatically.
- Some compound words use hyphens, like *long-range* and *field-effect*.

[*] Don't use `"` for closing double quotes. Not only does ``` ``example" ``` look a bit unbalanced, but `"` is used as a formatting command when typesetting certain languages, like German. (See Chapter 11 for more on international typesetting.)

- They are used in phrasal adjectives. If I ask for “five dollar bills”, do I want five $1 bills, or several $5 bills? It’s clearer that I mean the latter when set as *five-dollar bills*.

  In LaTeX, they are produced with the hyphen key (`-`).

En dashes indicate ranges such as “pages 4–12”, or separations like the “US–Canada border”. They are set with two adjacent hyphens (`--`).

Em dashes can be used to separate clauses of a sentence. Other punctuation marks—like parenthesis or commas—play a similar role. They are set with three adjacent hyphens (`---`).

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

Ellipses

A set of three dots used to indicate a pause or omission is called an *ellipsis*. It is set using `\ldots`.

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

becomes

I’m… not sure.

Spacing

As we discovered in our first example, LaTeX automatically inserts extra space between periods and whatever follows them—presumably the start of the next sentence. Sometimes, this isn’t what we want! Consider honorifics like Mr. and Ms., for example. In situations like these, 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, such as when we abbreviate units of measurement,[*] we want spaces that are thinner than usual inter-word ones. For these, we use `\,` :

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

announces

> Launch in 2 h 10 m.

What next?

- Add hyphenations for uncommon words using `\hyphenate` or `\-`.[†]
- Learn other commands for spacing, such as `\:`, `\;`, `\enspace`, and `\quad`.
- Use the `csquotes` package's `\enquote` to simplify nested quotations, e.g., "She exclaimed, 'I can't believe it!'"
- Discover the typographical origins of terms like *en*, *em*, and *quad*.
- Familiarize yourself with the difference between `/` and `\slash`.

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

[†] LaTeX usually does a good job of automatically hyphenating words, based on a dictionary of patterns stored for each language. You should rarely need these tools.

7. Layout

Justification and alignment

LaTeX is extraordinarily good at justifying text. Instead of considering each line individually—as most word processors and web browsers do—it examines all possible line breaks in a given paragraph, then chooses ones that will give the best overall spacing.[6] Combined with its ability to automatically hyphenate words, which allows line breaks in many more places,[7] this produces some of the best paragraph layouts available.

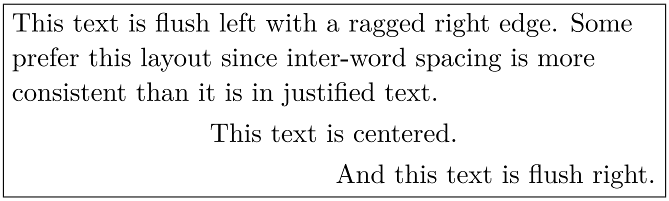
But sometimes we don't want our text justified. If you would like it to be "flush left" with a ragged right side, place it in a `flushleft` environment or add `\raggedright` to the current group. To center it, place it in a `center` environment or add `\centering` to the current group. And to flush it 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 inter-word spacing
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 inter-word spacing is more consistent than it is in justified text.

This text is centered.

And this text is flush right.

Lists

LaTeX provides three environments for lists: `itemize`, `enumerate`, and `description`. In all three, each item starts with an `\item` command. The `itemize` environment uses bullets. 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

The `enumerate` environment numbers your list:

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

produces

1. Collect underpants
2. ?
3. Profit

The `description` environment starts each item with some emphasized text, or *label*:

```
\begin{description}
\item[Alan Turing] was a British mathematician who laid much
    of the groundwork for the field of computer science.
    He is perhaps most remembered for his model of
    general-purpose computing, the Turing machine.
\item[Edsger Dijkstra] was a Dutch computer scientist.
    His contributions in many subdomains---such as concurrency
    and graph theory---are still in wide use today.
\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 on multiple processors.
\end{description}
```

gives us

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

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

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 on multiple processors.

Columns

We often split documents—or parts of them—into multiple columns. This is especially common when printing on A4 or US letter paper, since it allows more comfortable line widths at standard 8–12 pt text sizes.[*] You can either add the `twocolumn` option to your document class, which splits everything into two columns, or you can use the `multicols` environment from the `multicol` package:

```
One nice feature of \texttt{multicol} is that you can
combine arbitrary layouts.
\begin{multicols}{2}
In this example we start with one column,
then create a short section with two.
The package ensures that the text inside is split
so that each column is about the same height.
\end{multicols}
```

is split into

One nice feature of `multicol` is that you can combine arbitrary layouts.

[*] You'll see different advice depending on where you look, but a general rule of thumb is to design layouts to have fewer than 90 characters (including spaces) per line. Shorter lines keep readers from needing to scan very far to find the start of the next line, which produces a more comfortable experience.

In this example we start with one column, then create a short section with two. The package ensures that the text inside is split so that each column is about the same height.

Page breaks

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

Footnotes

Footnotes are a great tool for comments and asides which readers might find useful, but aren't crucial to the main text. The `\footnote` command places a marker at its location in the body text, then inserts its argument as a footnote at the bottom of the current page:

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

proclaims

> I love footnotes!*

What next?

- Control paragraph spacing, either using the relevant KOMA Script options, or with 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.
- Choose footnote symbols and layout with KOMA Script and the `footmisc` package.
- Insert horizontal and vertical space with commands like `\vspace`, `\hspace`, `\vfill`, and `\hfill`.
- Learn what units LaTeX provides for specifying spacing.
  (We've already mentioned a few here, such as `pt`, `bp`, and `in`.)

*Perhaps a bit too much…

8. Mathematics

LaTeX is excellent at typesetting mathematics, both inline with body text, e.g., $x_n^2 + y_n^2 = r^2$, and as standalone formulas:

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

The former is created with `$...$` or `\(...\)`, and the latter with `\[...\]`. Inside these math environments, the rules of LaTeX change:

- Most spaces and line breaks are ignored, and the engine usually makes spacing decisions 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 must occupy 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. Other formatting commands mentioned in Chapter 5 work as well. From

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

we get

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

Examples

Typesetting mathematics is arguably the raison d'être of LaTeX,* but the topic is so broad that giving it decent coverage would take up half of this book. Given how wide the field of mathematics is, there are *many* different commands and environments. Here, more so than any other topic, you owe it to yourself to find some real references and learn what the system is capable of. Before moving on, though, let's show some examples of what LaTeX can do.

*Well, TeX

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

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

9. Fonts

Digital typography has changed almost entirely in the past thirty years. Originally, LaTeX used METAFONT, a system designed by Donald Knuth specifically for TeX. As time went on, support for PostScript[*] fonts was added. Today, LuaLaTeX and XeLaTeX offer support for the two formats you're likely to encounter on your computer: TrueType and OpenType.[†]

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

OpenType was first released in 1996 by Microsoft and Adobe. One major improvement over TrueType is its ability to embed multiple versions of glyphs in a single font file. We'll see this feature in action through much of this chapter. OpenType files generally 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, you may want to use others for your document. For this, we turn to the `fontspec` package:

```
\documentclass{article}

\usepackage{fontspec}
\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[‡]

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

[†] Mac versions of LaTeX also support Apple's AAT, but we'll limit ourselves here to a discussion of the more ubiquitous formats.

[‡] Assuming, of course, that you have Adobe's open-source fonts installed.[8]

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

The `Ligatures=TeX` option allows you to use the standard punctuation shortcuts mentioned in Chapter 6 (e.g., creating en dashes with `--` or curly quotes with ``` `` ```) instead of having to type the the corresponding characters, which probably aren’t on your keyboard. You often don’t want these for monospaced type, though, since the text you set in it—such as code—is usually meant to be displayed verbatim. You don’t want `"Hello!"` to turn into `“Hello!“`.

Selecting font files

Typefaces come packaged as multiple files for their various weights and styles—a typical set includes upright, *italics*, **bold**, and ***bold italics***. `fontspec` can generally deduce the appropriate file names given the name of the typeface.[*] However, many typefaces come in more than two weights—some versions of Futura, for example, come in light, book, **medium**, **demi**, **bold**, and **extra bold**. Sometimes small caps are stored as separate files as well.[†]

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

- `Futura-Boo` for our upright book weight
- `Futura-BooObl` for our *oblique book weight*
- `FuturaSC-Boo` for small caps, book weight
- `Futura-Dem` for **upright demi(bold)**
- `Futura-DemObl` for ***oblique demibold***

[*] This is one of the places XeLaTeX and LuaLaTeX differ in a way that’s noticeable to the casual user. The former generally uses system libraries—such as FontConfig on Linux—to locate files from a typeface’s name. The latter has its own font loader, based on code from FontForge.[9] The expected name of a font might differ between the two engines—refer to the `fontspec` manual for details.

[†] OpenType allows small caps to be placed in the same file(s) as the other glyphs. If your font supports this, you don’t need to do anything—`fontspec` will dutifully switch to them whenever you use `\textsc` or `\scshape`. But for TrueType, and for OpenType fonts that don’t take advantage of this feature, you’ll have to load a separate file as shown here.

[‡] Compare how the light, book, **and medium weights** of Futura look compared to surrounding type on this page.

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.

Scaling

Using different typefaces to create a cohesive experience is tricky, especially since—as mentioned in Chapter 5—different typefaces might look completely different at the same point size. `fontspec` can help a bit here by scaling fonts to match either the x-height or the cap height of your main font with `Scale=MatchLowercase` or `Scale=MatchUppercase`, respectively.*

OpenType features

As mentioned above, one of OpenType's defining features is the ability to store multiple variations of a typeface's glyphs in a single file and allow users to switch between them. All of these features are specified as optional arguments to `\setmainfont` and friends. They can also be set for the current group with `\addfontfeature`. Let's touch on a few common ones.

Ligatures

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

*One way to sidestep this issue is to have fewer typefaces in your design. Even just one or two, used carefully, can produce amazing results.

†Ligatures fell out of style somewhat during the 20th century due to limitations of printing technology—including early computer typesetting—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. **Trivia:** glyphs such as the ampersand (&) and the German Eszett (ß) evolved from ligatures.

Standard ligatures are enabled by default, and remedy spacing problems a typeface might otherwise have. Consider the lowercase letters f and i. In many serif typefaces, these combine to form the ligature fi, which avoids awkward spacing between f's ascender and i's dot (fi). Other common examples in English writing include ff, ffi, fl, and ffl.

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

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

In the likely event that you also want to use `Ligatures=TeX`, multiple options can be grouped together, e.g., `Ligatures={TeX,Discretionary}`. Ligatures can also be disabled using corresponding `*Off` options. If you wanted to temporarily disable discretionary ligatures,

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

does the trick.

Some words are arguably typeset better without ligatures—a classic example is shelfful.[10] You can manually prevent the insertion of ligatures with an empty group, e.g., `shelf{}ful`, or use the `selnolig` package to handle most of these cases automatically.

Figures

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

A B C D 1 2 3 4

Oldstyle, or *text* figures, share more similarities with lowercase letters:

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

For body text, either choice is fine, but oldstyle figures shouldn't be combined with capital letters. “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, such that 1 takes up the same space as 8. As their name suggests, this is great for tables and other scenarios where figures must line up with the ones above and below them:

*[*Figure*] here refers to what some might call a *numeral* or *digit*—i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Typographers generally prefer the first term to the other two.

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

Proportional figures are the opposite—their spacing is, well... *proportional* to the width of the figure. They are usually preferred in body text, where 1837 looks a bit nicer than 1837.

You select figures with the following options:

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

These traits can be mixed and matched: you get proportional lining figures with `Numbers={Proportional,Lining}`, and tabular oldstyle ones with `Numbers={Tabular, OldStyle}`. And, like ligatures, each option has a corresponding `*Off` variant.*

Finally, some fonts provide *superior and inferior* figures, which are useful for ordinals (1st, 2nd 3rd, ...), fractions (25⁄624), and so on. These have the same weight as their full-sized counterparts, which looks much better than shrinking the font's normal figures. (Compare the above to 1st, 2nd, 3rd, and 25⁄624. Notice how this second set is too light compared to the surrounding type.)

What next?

- Learn about more OpenType features supported through `fontspec`, such as stylistic sets and alternatives.

- See how `fontspec` chooses fonts based on point size when multiple optical sizes are available, either automatically via OpenType, 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 bit more READABLE.

*This is especially useful since fonts select figures in different ways. In some, the default figures are lining and oldstyle figures are enabled with `Numbers=OldStyle`. To return to lining figures, `Numbers=Lining` doesn't work, but `Numbers=OldStyleOff` does.

10. Microtypography

Microtypography is the use of small, subliminal tweaks to improve a document's legibility. In other words, 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.[11]

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

```
\usepackage{microtype}
```

to your preamble and carry on—but let's take a brief look at what the package does.

Character protrusion

By default, LaTeX justifies lines between perfectly straight left and right margins. This is the obvious choice, but falls victim to an annoying optical illusion: lines ending in small punctuation—like a period or a comma—seem shorter than lines that don't.* `microtype` *protrudes* punctuation and other smaller characters into the margins to compensate.

Font expansion

To assist LaTeX's justification algorithm, `microtype` can stretch or shrink characters horizontally to produce a layout with more even spacing and fewer hyphenated lines. You might think that distorting character shapes this way would be immediately noticeable, but you're

*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 made slightly taller than "flat" ones (such as H and T) for all to appear the same height.[12]

reading a book that does this on every page! Font expansion is applied *very* slightly—by default, character widths are changed by no more than two percent.*

Note that this feature isn't currently available with XƎLATEX. You'll need to use LuaLATEX if you want to take advantage of it.

What next?

As always, see the package manual for ways to tweak these features. `microtype` is capable of a few other tricks, but several only work on older LATEX engines.† Those we do 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, many languages besides English exist. You may want to write with them.

Unicode

Digitizing human language is a complicated topic that has evolved significantly since LaTeX's inception, but today, computers usually represent text with Unicode. Briefly,

- A Unicode text file is made of a series of *code points*, each of which can represent a character to be drawn, an accent or other diacritical mark to combine with an adjacent character, or some non-printing character, such as an instruction to print subsequent text right-to-left.

- One or more of these code points combines to represent a *grapheme cluster* or *glyph*, the shapes within a font that we informally call "characters".

Привéт   नमस्‌ते

How many characters do you see? How many code points are they built from?

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

LuaLaTeX and XeLaTeX use these tools to render documents from Unicode input files.* Make sure that the fonts you select contain the needed glyphs—many only contain ones for Latin languages.

The polyglossia package

If your document contains anything besides English, you should try the `polyglossia` package. It will automatically:

- Load language-specific hyphenation patterns and other typographical conventions.

- Switch between user-specified fonts for each language.

- Translate document labels, like "chapter", "section", and so on.

*LuaLaTeX accepts UTF-8 files, while XeLaTeX is a bit more flexible and also accepts UTF-16 and UTF-32.

- 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 the current font, if applicable.

When using the package, specify the main language of your document, along with any other languages you use. Some languages also take regional dialects as an optional argument:

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

`polyglossia` will define an environment for each loaded language. These automatically apply that 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 ! »

What next?

- 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.

[*] Yes, it's *omelette au fromage*. Direct all complaints to Cartoon Network.

[†] `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 Typesetting Goes Bad

With luck, you're off to a good start with LaTeX. But as with any complicated tool, you'll eventually run into trouble. Here are a few common problems and what you can try to fix them.

Fixing overflow

When LaTeX cannot fit a line into a paragraph with good spacing, it gives up, overflowing that line into the margin. You can sometimes remedy this by adding `\emergencystretch=<width>` to the document's preamble. When a paragraph can't be broken to LaTeX's satisfaction, this command makes it try one last time, stretching or shrinking the space in each line by up to the provided width.* If that doesn't work, try tweaking the paragraph's wording. 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, where words are uncomfortably crammed together.

Avoiding widows and orphans

Typesetters do their best to avoid *widow* lines—which appear separated from the rest of their paragraph at the start of the following page. They also avoid *orphan*, or *club* lines, which are "left behind" on the previous page while the rest of the paragraph begins the next. LaTeX takes some effort to avoid these, but unfortunately, the algorithm it uses to split pages is much more simplistic than the one it uses to split paragraphs into lines.† You increase the "penalty" for doing so in order to make LaTeX try harder to avoid it‡ with:

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

*LaTeX has pretty sane default limits to how much it stretches and shrinks spacing in a paragraph. 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: "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." [13]

‡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.

`<penalty>` is a value between 0 and 10000. Be sure not to set these penalties *too* high—at their maximum, LaTeX is never allowed to leave orphans and widows, at *any* cost. This may cause odd layouts to be chosen. 300 is probably a sane starting point.

Handling syntax errors

If you confuse LaTeX—say, by issuing commands that don't exist, or forgetting to end a group or environment—it will print an error message,* then drop into an interactive prompt starting with `?`. Here you can enter various 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 likely just want to quit and try again once we've fixed our document. You do so by typing `X`, then pressing 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
```

*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 metaprogramming errors a run for their money. As you continue to use LaTeX, you'll start to get a feel for what sorts of mistakes cause these rare, but enigmatic messages.

A. A Brief History of LaTeX

Most programmers should be familiar with Donald Knuth, the man who coined the term *analysis of algorithms* in 1968 and pioneered many of the computer science fundamentals we use today. Knuth is perhaps most famous for his ongoing magnum opus, *The Art of Computer Programming*.

When the first volume of TAOCP was released that same year, it was printed the way most books had been since the turn of the century: with *hot metal* type. Each individual letter was cast from molten lead, then arranged into its line. These lines were clamped together to form pages of the book, which were finally inked and pressed against paper. By March of 1977, Knuth was ready for a second run of TAOCP, volume 2, but he was horrified when he received the proofs. Hot metal typesetting was an expensive, complicated, and time-consuming process, so publishers had replaced it with phototypesetting, which works by projecting images of characters onto film. The new technology, while much cheaper and faster, didn't provide the same level of quality Knuth had come to expect.[14]

The average author would have resigned themselves to this change and moved on, but Knuth took great pride in print quality, especially for the mathematics in his books. Around this time, he discovered an exciting new technology: digital typesetting. Instead of working with metal or film, letters and shapes were built from tiny dots, often packed together at over 1,000 per inch. Inspired by this burgeoning tech and frustrated with the current state of affairs, Knuth set off on one of the greatest yak shaves[*] of all time. For years, he paused all work on his books to create his own digital typesetting system. When the dust settled in 1978, Knuth had the first version of TeX.[†]

It's hard to appreciate how much of a revolution TeX was, especially looking back from a time where anybody with a copy of Word 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 format, DVI. Scalable fonts were uncommon at the time, so he created a system, METAFONT, to rasterize his characters into dots on the page.[‡] Perhaps most importantly, Knuth and his students designed algorithms to automatically hyphenate and justify lines of text into beautifully-typeset paragraphs.[◊]

[*] 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*.[15]

[†] TeX is pronounced like the first syllable of "**tech**nician". It is from the Greek τέχνη, for *art* or *craft*.[16]

[‡] Instead of constructing characters from lines and curves, as most TrueType and OpenType fonts do, METAFONT uses strokes of virtual pens. By tweaking the parameters of these pens and strokes, one can build entire font families from the same base designs.

[◊] These same algorithms went on to influence the ones Adobe uses in its software today.[17]

LaTeX, short for Lamport TeX, was developed by Leslie Lamport in the 1980s as a set of commands for common document layouts. It was introduced in 1986 with Lamport's guide, *LaTeX: A Document Preparation System*. Other typesetting systems based on TeX also exist, the other most popular today being ConTeXt.

Development continues, both in the form of user-provided packages for TeX and LaTeX, and on improvements to the TeX typesetting program itself. Currently, 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, and all subsequent releases have contained only bug fixes. With each release, the version number asymptotically approaches π by adding an additional digit. The most recent version, 3.14159265, came out in January 2014.

pdfTeX is an extension of TeX that provides direct PDF output (instead of TeX's DVI), 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.[18]

XeTeX 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 and inclusion in the TeX Live distribution in 2007.[19]

LuaTeX is similar to XeTeX 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, and is developed by a core team of Hans Hagen, Hartmut Henkel, Taco Hoekwater, and Luigi Scarso.[20]

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 students to study, so Knuth set out to make TeX into 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 interspersed between paragraphs. This mix of English and code is called `WEB`.†

Unsurprisingly, most modern systems don't have good tooling for the late 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 XeTeX are built by combining the result with other C and C++ sources. Instead of following this complicated process, the LuaTeX authors hand-translated Knuth's Pascal into C and have been using the resulting code since 2009.[21]

*Specifically, PostScript Type 1

†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 at the start, this book is incomplete. To keep things short, major LaTeX features—like figures, captions, and graphics—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 TeX Stack Exchange, at `https://tex.stackexchange.com/`

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`

For typography

We've spent most of our time here focusing on questions of *what* you can do with LaTeX, and little on *how* you should use it to create quality typography. 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

The Elements of Typographic Style, by Robert Bringhurst

Detail in Typography, by Jost Hochuli

Notes

1. Matthew Butterick, "Typography for Docs" (presented at the Write The Docs Conference, April 8, 2013), `https://www.youtube.com/watch?v=8J6HuvosP0s`

2. Erik Spiekermann, "Type is Visible Language" (presented at Beyond Tellerrand, Düsseldorf, Germany, May 19–21, 2014), `https://www.youtube.com/watch?v=ggQpDu63kk0`

3. *Graphic Means: A History of Graphic Design Production*, directed by Briar Levit (2017)

4. Lynne Truss, *Eats, Shoots & Leaves* (New York, 2003)

5. Butterick, "Hyphens and dashes", *Practical Typography*, `https://practicaltypography.com/hyphens-and-dashes.html`

6. Donald E. Knuth and Michael F. Plass, *Breaking Paragraphs Into Lines* (Stanford, 1981)

7. Franklin Mark Liang, *Word Hy-phen-a-tion by Com-put-er* (Stanford, 1983), `http://www.tug.org/docs/liang/`

8. Adobe's open-source typefaces are freely available at `https://github.com/adobe-fonts`

9. *LuaTeX Reference* (Version 1.0.4, February 2017), 10

10. Knuth, *The TeXbook*, (Addison-Wesley, 1986), 19

11. R Schlicht, *The microtype package* (v2.7a, January 14, 2018), 4

12. Jost Hochuli, *Detail in typography* (Éditions B42, 2015), 18–19

13. Knuth, *The TeXbook*, 110

14. Knuth, *Digital Typography* (Stanford, 1999), 3–5

15. "yak shaving", *The Jargon File*, `http://www.catb.org/~esr/jargon/html/Y/yak-shaving.html`

16. Knuth, *The TeXbook*, 1

17. 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 via Knuth's collaborations with Zapf. *hz* was later acquired by Adobe and used in part when creating their paragraph formatting systems for InDesign.

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

19. Jonathan Kew, "XƎTEX Live", *TUGboat* 29, no. 1 (2007)

20. `http://www.luatex.org`

21. 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 Garamond Premier by Robert Slimbach. His revival is based on 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 Matthias Tellen's `mononoki`, a typeface designed to work well on both low-resolution computer monitors and in high-resolution print.

Captions are set in Neue Haas Grotesk, a Helvetica restoration by Christian Schwartz. Other digitizations of the classic Swiss typeface are based on fonts made for Linotype and phototypesetting machines, resulting in digital versions with all the compromises and kludges from those past two 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. Originally released in 1927 by Paul Renner, 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* language in the Unicode standard.

Finally, Latin Modern—the OpenType version of Knuth's Computer Modern used throughout the book—as well as TeX Gyre Termes—the free alternative to Times Roman seen on page 1—are from the digital type foundry of Grupa Użytkowników Systemu TeX, the Polish TeX Users' Group. An overview of their excellent work can be found at the following locations:

`http://www.gust.org.pl/projects/e-foundry/latin-modern`

`http://www.gust.org.pl/projects/e-foundry/tex-gyre`.