A second course in LaTeX

Petri Laarne

Version 1.1b
June 16, 2025

Contents

Chapter 0

Introduction

These notes were created for a course called *Advanced LaTeX*, offered to PhD students at University of Helsinki in springs 2024 and 2025.[1] They attempt to give a somewhat unified view of what LaTeX is, how it works, and how it should be used in producing scientific literature. They are much larger than can be covered in a two-week intensive course – my hope is that they can work as a reference as well.

These notes owe a lot to *The LaTeX Companion* [4] by Frank Mittelbach and other LaTeX team members. The third edition of the book is a treasure trove for a serious LaTeXnician, but at three kilograms (printed version) it is not an investment for everybody. I have tried to distill some of the wisdom[2] into these notes (which are still not what you'd call lightweight).

Of course, the authoritative reference for each package is the documentation hosted on the Comprehensive TeX Archive Network (`ctan.org`).

0.1 What you need

I assume that you already know the basics of LaTeX – ideally, you have written a Bachelor's or Master's thesis or an article with it.

You need to have an up-to-date LaTeX environment. If you use a local installation, pause now for a moment and check whether the packages are up to date. If you use Overleaf, then the system is already good to go.

I encourage you to compare different editor programs. As programmers can testify, having a good editor makes coding much more pleasant. It is good to get familiar with the keyboard shortcuts your editor offers.

Since there are so many editors and they change much faster than base LaTeX, these notes do not discuss their use.

[1] I dislike the name of the course. As you will find out, we do not talk very much about advanced topics like TeX internals or package development. A better name would be *Professional LaTeX* or *A second course in LaTeX*, as these notes are named.

[2] While undoubtedly introducing some un-wisdom of my own.

Note to Overleaf users
As an exception, there are a few blocks like this that discuss Overleaf. I felt that Overleaf is worth mentioning, since so many people use it nowadays, and it has some differences to locally installed tools.

0.2 About this version

This is Version 1.1 of the notes, published at the end of the Spring 2025 course.

This version contains some fixes and additions to the 1.0 version, but it is still in need of a good copy editor. All red **TODO** notes that were there at the end of the 2024 course have just been turned into blue **FUTURE** notes, only visible in the working draft available on GitHub (see below).

Maybe someday I will have a chance to properly polish these notes. Any comments and suggestions are more than welcome.

0.3 See how it is made

These notes are licensed under the Creative Commons Attribution 4.0 license.[3] You are free to copy and redistribute them as you like. You can also modify and reuse them freely, under the condition that you indicate clearly the original author and whether you have modified the work. You can see the license terms for complete description.

The source code and latest PDF release for these notes are available on GitHub: `github.com/polsys/2nd-course-latex`.

You are invited to see how I built these notes – I tried to follow my own guidelines, but of course you can disagree with some choices. In the end, there is no one true path to TEXnical enlightenment.

If you spot a typo or an error, you can send a pull request with the correction, file a GitHub issue, or send me an email about it. You can find my up-to-date contact information at `petri.laarne.fi`.

I hope that you find these notes useful!

[3] `creativecommons.org/licenses/by/4.0/`

Chapter 1

Programming

What really is LaTeX?

It all starts with TeX, the typesetting system designed by Donald Knuth in late 70s and 80s. It had just become possible to use computers to control the chunky machines whose output could be fed into a printing press. Nowadays the machines are less chunkier and everybody just reads PDFs on screen, but the principles are the same.

The chunky machines take in sequences of instructions: move down one inch from top of page, select the 12-point Computer Modern font, write "Hello", move 5 points right, write "world"…

The original TeX is a language that expresses these instructions in a machine-independent way, and automates layout tasks like line breaking. Moreover, TeX is a powerful macro language. It is possible to implement very complex algorithms with it and to redefine any part of the language.

TeX is a powerful system for laying out text, but it puts a lot of responsibility on the writer. It will break lines and paragraphs for you, but you still need to handle the overall layout. Essentially, you are responsible for both the content and its presentation.

LaTeX (originally by Leslie Lamport, now maintained by several people) extends this core by introducing commands like `\section`: this command not only lays out the section title in a bigger font, but also adds some vertical whitespace around it, updates the current section number, and so on.

If you want to add a table of contents, you just add `\tableofcontents` in a suitable spot. LaTeX fills in the contents it gathered from your sectioning commands and makes sure the page numbers are right.

LaTeX *separates the content from how it is presented*. We will emphasize this philosophy on this course. By using standard constructs, packages, and our own custom commands, we can make the `.tex` source code *semantically* meaningful: something that you could read aloud. Any customization to the style is done in a separate place.

Warning
Even when using LaTeX, the old TeX is still there. It is possible to use plain-TeX commands in your documents, but *this is highly discouraged*. These commands sidestep all the enhancements brought by LaTeX, and might cause hard-to-diagnose problems. You might need them when writing more complicated packages, but then you are already solidly in the "you are free to shoot your own foot" territory. Some such commands are noted in these Warning boxes.

Unfortunately, many heritage templates and Stack Exchange answers still mix TeX, LaTeX, and pre-1994 LaTeX. Examples of the latter include the obsolete `\bf` and `\it` commands (e.g. writing `{\bf bold}` instead of `\textbf{bold}`).

New in LaTeX3
The "standard" LaTeX version is 2ε, released in 1994. The core commands have not changed since then. Evolution of the system happens in extension packages.

LaTeX3 is a long-running project (started already before 2ε, and re-activated around 2018) on rewriting the internals of LaTeX to be more extensible.

LaTeX3 introduces new commands for defining custom commands and environments. We will only discuss the version 2ε methods (which are still valid) here, but if you find yourself writing a package, you should consider learning about the new methods.[1]

We will talk about some new features and accessibility improvements brought by LaTeX3 below, starting already in Section 1.4.

1.1 How LaTeX processes files

The design of the TeX compiler is limited by the era it was conceived in. In the 1980s computer memory was measured in kilobytes, so the compiler stores as little information as possible.

In particular, TeX reads source files from top to bottom. Commands are evaluated immediately as they are encountered. TeX keeps track of text and content only for the current page. Once the page is full, it is "shipped out" as printing commands and cannot be changed anymore. We talk about the layout algorithm in Chapter 3.

As TeX never backtracks, how it is possible to create a table of contents, or any cross-references to upcoming sections at all? The table of contents is shipped out before TeX has a chance to know what to put there!

[1] `www.latex-project.org/help/documentation/usrguide.pdf`

The trick is simple: Whenever a `\section` or `\label` command is encountered, its name and position is recorded to an `.aux` file. LaTeX reads in this file before it starts processing the `.tex` file. This way it knows about what to expect.

Consequently:

1. On the first compilation, LaTeX does not know about upcoming references. An `.aux` file is generated.
2. On the next compilation, cross-references are resolved using the `.aux` file from the preceding compilation.
3. This means that all page number references point to *the state of the previous compilation*.
4. Since the resolution might change page numbering, it might be necessary to iterate. LaTeX gives the "`Label(s) may have changed. Rerun to get cross-references right.`" warning in this case.
5. For well-behaved documents, at most three compilations should give convergence.

Warning
Sometimes, a typo'ed command might end up messing the `.aux` file. In this case, fixing the error and recompiling will not be enough, since the `.aux` file feeds back incorrect data.

Usually, the `.aux` file from the recompilation is no longer erroneous, so a second recompilation is enough.

Rarely, the state might be so messed that the `.aux` file gets re-corrupted before the compilation halts. This manifests itself as weird compilation errors even when you have reverted your code back to a known good state. If this happens, delete the auxiliary files and recompile from scratch. Some LaTeX editors have even a handy button for this.

Since this is the only mechanism for creating forward references, the `.aux` file is not the only one of its kind. Some other temporary files that you can expect to see in your compilation folder:

`.bbl` The bibliography generated by the bibliography compiler. See below and Section 5.2. Some publishers, in particular arXiv, require you to submit this file together with your source code.

`.idx` Index created by the **makeindex** program (Section 5.3.1).

`.lof` List of figures.

`.log` This is the raw compiler log. LaTeX editors parse this into a more readable form, but long, multi-line error messages can be easier to read from the raw log.

`.lot` List of tables.

`.synctex.gz` This is used by some editors to synchronize the scrolling of PDF and source code.

`.thm` Theorems (ntheorem package).

`.toc` Table of contents.

This list is not exhaustive; several packages produce their own auxiliary files. They generally share the file name of the main `.tex` file.

Technical side note
These notes also use this mechanism. Code examples are written inside a special environment within the chapter `.tex` file. TeX outputs the example to a temporary file so that it can be read back *twice*: once to print out the code, and then again to show the results.

These auxiliary files can also be read and written by other programs. The prime example is the **bibtex** program. Processing a database of bibliographic entries is too complicated to do inside the TeX programming environment. Therefore, a separate program is used.[2] It works like this:

1. You write bibliographic data in a `.bib` file. See Section 5.2 for the syntax.
2. You write citation commands in the `.tex` source code.
3. When the LaTeX compiler encounters citations, it writes some special commands in the `.aux` file.
4. When the **bibtex** program is invoked, it reads the `.aux` file. It creates a combined list of all citations, and outputs the commands to typeset a bibliography into the `.bbl` file.
5. When LaTeX is run again, it reads the `.bbl` file and prints the bibliography.

The LaTeX compiler and **bibtex** are decoupled in the sense that you only need to run **bibtex** when you need to update the bibliography.

In fact, arXiv does not run any bibliographic compilers at all. They require you to submit the `.bbl` file created by your preferred compiler. (This probably has to do with two facts: 1. There is more than one bibliography compiler in wide use. 2. Some people use the same massive `.bib` database for all their articles, and probably don't want to share it.)

[2] The newer **biber** compiler used by biblatex package works in the same way.

1.1.1 Splitting your source into multiple files

The `\input` and `\include` commands can be used to include additional `.tex` files. This makes it possible to e.g. write each chapter of a long document in a separate file.

Good practices
While it would sound reasonable to split an article into smaller files, there are at least three reasons not to do so:

- If you are sharing files with collaborators, the complexity of keeping files in sync grows faster than linear in the number of files.
- Many journals prefer to have the submission in one `.tex` file for similar logistical reasons.
- There can also be some complexity in setting up your editor.

Therefore my recommendation is: only split the document if you are preparing a book. It is up to your pain-tolerance whether a thesis counts as one. These very notes are split into one file per chapter.[3]

When I started preparing this course, I asked some other TEXnicians for important pieces of advice to give. This was the piece of advice I got the most. Consider yourself warned.

Good practices
If you separate chapter contents into their own files, I highly recommend keeping the "main file" very short – consisting only of the preamble and `\include` commands.

There is a minor difference in the two commands. When `\input` is called with the name of a `.tex` file, the compiler acts as if the contents of the file were placed in that position. You could replace the input operation by copy-pasting the file contents.[4]

On the other hand, `\include` finishes the current page before processing file contents. This involves placing any queued floats (see Section 7.2). That makes it mostly suitable for including separate chapters. Every included file gets its own `.aux` file.

The `\includeonly` command temporarily restricts the compilation to a subset of the included files. This can be a neat way to reduce compilation

[3] And suffer from some associated problems. The "main file detection" heuristics of my editor do get confused by this chapter and the Beamer chapter where several `\documentclass` commands appear. It tries to compile the chapters as standalone files, failing spectacularly with 1000+ errors.

[4] Some environments do not allow `\input` inside them.

times on a long document. Since each file has its own `.aux` file, cross-references to uncompiled chapters might work (but be outdated!).

```
% In the preamble
\includeonly{intro,discussion}

% In the document body
\include{intro} % Input intro.tex
\include{methods} % Input methods.tex
\include{results} % Input results.tex
\include{discussion} % Input discussion.tex
```

Gotcha!
`\include` produces only a warning and not an error if it cannot find the specified file. Also, some packages work unreliably with `\include`.

Do not use `\includeonly` when preparing the final copy of your document, since some of the `.aux` files could be outdated.

Gotcha!
If you define commands, counters, or float types inside `\include`'d files, you will probably end up in pain whenever you use `\includeonly`. Define those things in the preamble.[5]

Regardless of which command you use to include files, the included `.tex` file must not contain a preamble – there already was one in the main file. Conversely, this means that the separated files cannot be compiled on their own.

There might be some editor-specific support for setting the main file of your project. This allows you to hit *Compile* while editing a chapter file, and the main file gets compiled instead.

Note to Overleaf users
You can set the main file in the settings menu at top left of the editor.

1.1.2 *Standalone files**

There is also a standalone document class. This is an odd beast that can be used for some advanced workflows. See its documentation on CTAN before proceeding further.

When loaded as a document class, it creates a page just large enough to contain its contents. For example, you could use it to output a TikZ picture

[5] Again an observation sponsored by these notes, since there are examples defining all of those objects.

or an equation to its own PDF. (If the content is just a TikZ picture, pass the `[tikz]` option to the class.)

When loaded as an ordinary package (`\usepackage{standalone}`), it instead provides the `\includestandalone` command. This can be used to include standalone .tex files like with `\input`.

The real power comes from the `mode` parameter to the package. Let's consider the following code:

```
% Preamble
\usepackage[mode=buildmissing]{standalone}

% Document
\begin{figure}
\includestandalone{complicated-picture}
\caption{A very complicated picture.}
\end{figure}
```

The `mode=buildmissing` option does the following:

- If `complicated.pdf` exists, it includes it with `\includegraphics`.
- If the PDF does not exist, it instead *compiles* `complicated.tex` into a PDF, and then includes the produced PDF.

That is, the picture and the rest of the document are compiled separately. (There is also `mode=buildnew` that can be useful for development.)

This can be useful for reducing compile times on complicated TikZ pictures, or a very specialized workflow: Some TikZ extension packages require **lualatex**, which is not supported by arXiv or many journals. With this trick, you can use **lualatex** for your writing flow, but send the pictures as PDFs for the journal. No code changes needed!

Technical side note
For this to work, your LaTeX compiler must be permitted to call external programs (namely, itself). Overleaf permits this by default, but otherwise you might need to tweak some editor settings or command line options.

1.2 Commands and environments

Good practices
When should you define your own commands? Generally, the less customization you have, the easier your manuscript is for the journal publication process and your coauthors. At the same time, good commands make the code semantically meaningful.

I personally find `\norm{...}` easier to read than `\left\|...\right\|`, and `\Prob` certainly better than `\mathbb P`, but others could find those an overkill. This is a hard balance to strike.

There is just one strict rule: never repurpose the name of an existing base LaTeX command. It will cause endless trouble when the journal style or some package tries to use the original command.

1.2.1 Defining commands

TeX is a macro language. This means that there are special source code tokens (*macros*), that are *expanded* into sequences of more tokens (including more macros, which are expanded recursively). The core system includes only a few commands for manipulating internal state and putting out text, and the rest is achieved via macro expansion.

In LaTeX macros are defined with the `\newcommand` command.

```
\newcommand{\hello}{Hello world!}

\hello
```

Hello world!

Here `\newcommand` takes two arguments: the first is the name of the macro to define, and the second is what the macro expands to. In TeX things are grouped with the `{}` braces. (The first set of braces is technically unnecessary, but it looks cleaner in my opinion.)

Let us see another example where the command is used repeatedly, and also expanded inside other commands:

```
\newcommand{\hello}{Hello world}

--\hello, they whispered.\\
--I said, \MakeUppercase{\hello}!
```

–Hello world, they whispered.
–I said, HELLO WORLD!

Since the macros are expanded recursively until there is nothing more to expand, you can write things that are semantically very unclear, but comprehensible to a computer:

```
\newcommand{\hi}{^\infty}
\newcommand{\underscore}{_}
\newcommand{\lo}{\underscore{i=1}}

\[ \sum\lo\hi \]
```

$$\sum_{i=1}^{\infty}$$

The expansion goes something like this:

1. `\sum\lo\hi`
2. `\sum\underscore{i=1}\hi`
3. `\sum_{i=1}\hi`
4. `\sum_{i=1}^\infty`

So the end result is the same as if one had written the sum sensibly from the beginning.

Warning
In plain TEX macros are defined with `\def` and `\let`. If you see code using either, you have entered the "shoot your own foot" territory.[6]

The TEX compiler works in two basic modes: text mode and math mode. Some commands (for example, `\sqrt`) are only usable in math mode.

How to define a command that works *both* in text and math mode? You can use the `\ensuremath` command: in text mode it wraps its contents inside `$`, and in math mode it does nothing:

```
\newcommand{\magic}{\ensuremath{\sqrt 2}}

The number \magic{} satisfies
\[
\magic^2 = 2.
\]
```

The number $\sqrt{2}$ satisfies

$$\sqrt{2}^2 = 2.$$

1.2.2 Arguments to commands

To pass arguments to macros, we can indicate their number with an optional argument between the macro name and definition. These arguments can then be accessed in the macro definition with `#1`, `#2`, and so on.

[6] Since you asked, the difference between the two is whether the contents of the macro are expanded at usage or definition time. Neither protects you against overwriting existing macros.

```
\newcommand{\say}[2]{#1 says ``#2''!}
\say{Petri}{Hi}
```

Petri says “Hi”!

Arguments are often wrapped in the `{}` braces, but it is not actually necessary – in a very specific case. By default, LaTeX interprets a single letter or a number as an argument. The braces extend the argument to a longer stretch.

This means that all of the following are equivalent:

```
\[
\frac{1}{2}
= \frac 1 2
= \frac12.
\]
```

$$\frac{1}{2} = \frac{1}{2} = \frac{1}{2}.$$

Do note the last one – LaTeX command names can only consist of letters, so the number 1 is not interpreted as part of the command name but an argument. There is no requirement to separate the name and a following number. (I do find the last example hard to read, though.) Conversely, whitespace won't separate longer arguments; you really need the braces:

```
\[
\frac 12 34
\neq \frac {12} {34}.
\]
```

$$\frac{1}{2}34 \neq \frac{12}{34}.$$

However, commands are interpreted as single tokens. This happens regardless of whether the command would expand to several tokens.

```
\newcommand{\magic}{314 159}
\[ \frac \magic \pi \]
```

$$\frac{314159}{\pi}$$

Gotcha!
Some built-in macros swallow the whitespace that follows them. If a word space is actually needed, you can feed the command an empty group `{}`:

```
\LaTeX programming\\
\LaTeX{} programming
```

LaTeXprogramming
LaTeX programming

1.2.3 Groups and scopes

More precisely, the braces delimit a *group*. A group is handled as a single entity. Additionally, it serves as a *scope* for commands that change how all the following text is output. For example, the font-sizing commands like `\tiny` and `\Large` affect the size of all text that follows them. However, this effect only lasts until the group is closed with a `}`.

```
{\tiny I am small}
and I am normal
and {\Large I am large}
```

I am small and I am normal and
I am large

If you forget about this and just write `\tiny` without any scoping, you will have tiny text until the end of document (or next size command).

```
\tiny I am small and
{\Large I am large}
and I am tiny again
```

I am small and I am large and I am
tiny again

There is one place where the braces do not delimit a group: in the definition of commands. In the following example, `\mouse{Squeak}` is expanded into `\tiny Squeak` and not into `{\tiny Squeak}`. Therefore the effect persists onto the following line.

```
\newcommand{\mouse}[1]{Mouse: \tiny#1.}

\mouse{Squeak}\\
I: Oops.
```

Mouse: Squeak.
I: Oops.

The scope can be introduced here by adding `{}` into the command definition:

```
\newcommand{\mouse}[1]{{Mouse: \tiny#1.}}

\mouse{Squeak}\\
I: Alright.
```

Mouse: Squeak.
I: Alright.

Scopes also affect things like `\newcommand`: the command `\mouse` defined above only exists within the example block, and is not accessible beyond it.

1.2.4 Optional arguments

Some commands also have optional arguments. Instead of braces, these are passed within brackets `[]`. A basic example is the `\sqrt` command that produces not only square roots, but general roots:

```
$\sqrt{7}$,
$\sqrt[3]{7}$.
```

$\sqrt{7}$, $\sqrt[3]{7}$.

Gotcha!
LaTeX ignores whitespace between a command and its optional argument. This might sometimes lead to surprising behaviour. Here the second list item is interpreted as an optional argument that replaces the bullet symbol:

```
\begin{itemize}
\item Cookies
\item [Maybe cake?]
\item Drinks
\end{itemize}
```

- Cookies

Maybe cake?

- Drinks

Another example is the optional argument to `\cite` command, which indicates a precise position within the cited work: `\cite[Page 40]{TLC}` produces [4, Page 40]. (Here `TLC` is the bibliography key for *The LaTeX Companion*; we will discuss this more in Section 5.2.)

Gotcha!
LaTeX does not keep track of `[]` nesting. Inside an optional argument, LaTeX interprets the first `]` it sees as the closing bracket. This means that if you need to use `[]` *inside* an optional argument (say, as an optional argument to an inner command), you need to protect it with braces. Quite often, you might need even *two sets of braces*. Compare the two citations:

```
\cite[{{Page [40] maybe?}}]{TLC}\\
\cite[Page [40] maybe?]{TLC}
```

[4, Page [40] maybe?]
[**m**]aybe?]TLC

The first citation is shown correctly. In the second:

- the optional argument is interpreted to be "`Page [40`",
- the first `]` already closes the optional argument,
- the citation key is taken to be "`m`" (only a single letter since there are no braces),
- such a key does not exist so biblatex prints it as "[**m**]",
- the remaining "`aybe?]{TLC}`" is then normal body text.

The syntax for optional arguments is not always clear: sometimes they precede the real arguments, sometimes they follow them. Sometimes they consist of just one thing, sometimes they can be a list of things (see the discussion of `\usepackage` later in this chapter).

1.2.5 Replacing existing commands

Sometimes it is useful to overwrite an existing command. One common example is rewriting `\epsilon` to actually mean `\varepsilon`, since many consider the variant ε prettier than the regular ϵ.

It is not possible to write `\newcommand{\epsilon}{\varepsilon}`, since LaTeX rightly complains about `\epsilon` being already defined. You could be accidentally overwriting a command used elsewhere in the document. You have to make your intentions clear by using `\renewcommand`:

```
Old epsilon: $\epsilon$,\\
\renewcommand{\epsilon}{\varepsilon}
New epsilon: $\epsilon$.
```

Old epsilon: ϵ,
New epsilon: ε.

Again, the redefinition of a command lasts only for the current scope. Once the scope is closed, `\epsilon` again means the old regular symbol. To redefine a command within the entire document, you need to call `\renewcommand` in the document preamble.

Here is an example of `\newcommand` and `\renewcommand` with arguments. The two commands have matching syntax. Note that the new definition is *allowed* to have a different number of arguments, but it might be confusing!

```
\newcommand{\dual}[2]{(#1 \mid #2)}
Mathematicians: $\dual{a}{b}$.

\renewcommand{\dual}[2]
  {\langle#2 \mid #1\rangle}
Physicists: $\dual{a}{b}$.
```

Mathematicians: $(a \mid b)$.
Physicists: $\langle b \mid a \rangle$.

The `\DeclareCommandCopy` robustly copies the current definition of a command.[7] It is then possible to renew the definition, and still access the previous version. Let us illustrate this by swapping the meanings of `\epsilon` and `\varepsilon`:

```
Epsilon $\epsilon$ and varepsilon $\varepsilon$

\DeclareCommandCopy{\oldepsilon}{\epsilon}
\DeclareCommandCopy
    {\oldvarepsilon}{\varepsilon}
\renewcommand{\epsilon}{\oldvarepsilon}
\renewcommand{\varepsilon}{\oldepsilon}

Epsilon $\epsilon$ and varepsilon $\varepsilon$
```

Epsilon ϵ and varepsilon ε
Epsilon ε and varepsilon ϵ

New in LaTeX3
This command is part of the LaTeX3 programming interface. It requires a relatively recent (2020 or later) LaTeX version.

1.2.6 Defining environments

Environments encapsulate larger blocks of text. They also form implicit groups.

[7] Do not use raw TeX primitives to do this.

```
\begin{center}
\renewcommand{\epsilon}{\varepsilon}
Centered text and $\epsilon$
\end{center}

\begin{flushright}
Right-aligned text.
Here we have the old $\epsilon$.
\end{flushright}
```

Centered text and ε

Right-aligned text. Here we have the old ϵ.

To define an environment, we use the `\newenvironment` command. This command takes three arguments: the name of the environment and code to expand at the beginning and the end of the environment.

```
\newenvironment{cool}
    {A mathmo enters the lab.\par}
    {\par The mathmo leaves the lab.}

\begin{cool}
A massive explosion occurs.
\end{cool}
```

A mathmo enters the lab.
A massive explosion occurs.
The mathmo leaves the lab.

(If you're wondering about the `\par` commands, they are used to ensure that the first and last lines are their own paragraphs.)

There is also `\renewenvironment` that works like `\renewcommand`.

Since environments are groups, it is possible to change font characteristics for the duration of the environment:

```
\newenvironment{mouse}{\tiny}{}

Ordinary text.
\begin{mouse}
Very small and very squeaky text.
\end{mouse}
Again ordinary text.
```

Ordinary text. Very small and very squeaky text. Again ordinary text.

Note that here the end code is empty; the font properties are reset automatically as the group ends.

Gotcha!
If the begin/end code of your environment spans multiple lines, you need to be careful with line breaks. TEX happily copy-pastes things including whitespace characters, and whitespace is meaningful. The extra whitespace might cause LATEX to output an unintended empty space or even a paragraph break.

To avoid line breaks to be interpreted as line breaks, you can end each line in the environment definition with `%` – the comment character shallows the line break.

This same thing applies to empty lines before and after environments. If you don't want to start a new paragraph after the environment ends, do not leave an empty line between `\end{...}` and the following text. (For visual separation in the code, I prefer a line containing just `%`.)

1.3 Diagnosing errors

Macro languages were popular in the 1980s, when 640 kilobytes was enough memory for anyone. An unfortunate consequence of TEX's stability is that the compiler still operates under this worldview. It does not see any meaning in commands, it just expands them into more primitive commands. If your code is not correct, error messages can be extremely cryptic.

Note to Overleaf users
Some LATEX distributions are set up to ignore some errors. For example, Overleaf is quite good at ignoring missing `$` signs and other small typos. ArXiv and publishers are not as relaxed. You should really pay attention to red symbols in the source code margin and next to the *Recompile* button!

Gotcha!
Pay attention to warnings! Some packages are incompatible with each other, and might need to be loaded in a certain order. Often these packages give a warning when loaded in wrong order. The warning might be buried under error messages caused by the said incompatibility.

The compiler gives three forms of output: information, warnings and errors. Graphical LATEX editors usually only show the latter two. (Information is printed in the console output. It includes the current source file and page number that the compiler is processing, which can help diagnose a stuck compilation.)

Warnings are just as the name says: the document can still be compiled, but you should check whether you have gotten what you intended. Generally, the warning messages are very descriptive:

Overfull or underfull hbox.
A line could not be breaked in a nice way. See Section 3.1 for how to solve this.

Citation or label undefined.
LaTeX could not recognize a label that you tried to reference. If the label or bibliography entry was newly added, a recompilation might fix this. Otherwise you have forgotten to define it at all, or just forgotten how to spell it. See Sections 5.1 and 5.2.

Label(s) may have changed. Rerun to get cross-references right.
Indeed, you have added a new label since the last compilation. Just recompile the file.

Temporary extra page added at the end. Rerun to get it removed.
This can happen if the document becomes shorter than in the last compilation. LaTeX tries to place the processing that happens at `\end{document}` to the final page, but it does not know the final number of pages until it is too late (hence it looks at the previous compilation for a hint). As the message says, there is an extra page explaining the issue at the end. Recompile and it goes away.

Good practices
You should always check for warnings before submitting a document. It's a bit embarrassing to publish a document with [**?**] citations, after all.

Overfull and underfull hboxes are usually *not worth fixing* until the final typesetting phase, since very small changes to the text or style can drastically change the line breaking.

Errors, on the other hand, halt the compiler altogether. The compiler can try to ignore errors and recover itself, but at that point *it is guessing* what you tried to mean, and might get even more confused. Quite often the internal state is so messed up, TeX can only spew more errors.

Good practices
Recompile often. It is much easier to locate an error when you have modified only a small region of the code.

Always recompile after you modify a command definition or do a find-and-replace operation. These are risky.

If your document takes a long time to compile, find a way to speed it up (skip complicated TikZ pictures in draft mode, only include chapters you're working on, etc.). If recompilation breaks your flow, you won't do it often enough.

For some pointers for conditionally skipping parts of pictures, see page 34.

[100+ error messages]
Did you just modify the definition of a command? If so, then you probably introduced a typo there.

If not, this is a strong sign that TEX failed to recover from an error. Scroll to the *very first* error message and fix it. The later error messages are misleading, and will go away once the actual error is fixed.

If the cause of the error is a command used inside a sectioning command or table specification, you may have hit a robustness issue; see Section 1.7.2.

`\begin{equation}` *on input line 7 ended by* `\end{equation*}`
Exactly what is says: there is a mismatched begin/end combination. If the message says `\end{document}`, then the end command is missing altogether.

Missing `$` *inserted*
You have tried to use a math-mode command while in text mode. If you want to define a custom command that works in both modes, you can wrap mathematical symbols in it within `\ensuremath`.

Bad math environment delimiter
Conversely, do you have an extra `$` inside a display math environment?

Misplaced alignment tab character `&`
Extra alignment tab has been changed to `\cr`

You have a bit too many `&` characters per line, or are using an equation environment that does not support them at all.

Package amsmath Error: `\begin{split}` *won't work here.*
You need to wrap the environment inside `equation` or one of its friends.

Missing character: There is no … in font nullfont!
If this happens inside a TikZ picture, you have very likely mistyped a command or option. Check the code line shown above the error message.

1.4 Document structure

Let us then look at the basic structure of a `.tex` file. Everyone reading these notes is assumed to have seen such a file more than once, but we will spend some time on some nuances. Bear with me even if this sounds trivial!

Good practices
I would advocate for starting every project from an empty `.tex` file. The problem with "heritage" templates is that they accumulate a lot of unnecessary package dependencies and custom commands. I've seen files where the same package is loaded three times.

By starting from an empty file and only adding the customizations you need for the particular project, you help maintain your "code hygiene".

Let us start with the minimal example below.

```
1  \DocumentMetadata{}
2  \documentclass[a4paper]{article}
3
4  \fontencoding{OT1} % Read note below!
5  \usepackage[hidelinks]{hyperref}
6
7  \title{My first document}
8  \author{Firstname Lastname}
9  \date{\today}
10
11 \begin{document}
12
13 \maketitle
14
15 % Rest of content here...
16
17 \end{document}
```

We discuss the first line in a LaTeX3 note below. On the second line, we load a LaTeX document class. The document class determines the basic layout of your document, and it is where several basic commands like `\maketitle` are defined. We will look at some different document classes and arguments in the next subsection.

Lines 1–10 are collectively known as the *preamble*. This is the best place to define new commands and do other setup. Additional packages are also loaded here; see Section 1.4.2.

New in LaTeX3
What's interesting in the above example is the one package that is *not* loaded. The inputenc package used to be very important to load, since it told LaTeX about the encoding of non-English characters. Since 2018 LaTeX has followed the rest of the world and uses UTF-8 encoding by default.

What does this mean for you? If you create a new file in any reasonably modern LaTeX editor, you get the correct encoding and don't need to load inputenc.

However, if you modify a sufficiently aged file that contains non-English characters, you might stumble upon legacy encodings like `latin1` or `cp1252`. In such files, the inputenc package must be loaded with the proper optional argument.

The `\fontencoding` command tells LaTeX that Unicode characters will appear not only in the source code, but also the final output – thus the font must be interpreted in a compatible way. However, this command should

only be used with the traditional pdfTeX compiler; if you use the newer LuaLaTeX or XeLaTeX compilers, this command might cause subtly worse output. See page 63 for more explanation.

Trying to output any text before the `\begin{document}` line is an error. When the compiler reaches that line, a lot of stuff happens behind the scenes to prepare LaTeX for actually outputting pages.

Similarly, the `\end{document}` line is necessary. At that point, a lot of code is executed to make sure that everything is output properly. Any text written below that line is ignored. (I do not recommend writing anything there!)

New in LaTeX3
An important part of the LaTeX3 project is revising the PDF output to include accessibility information: for example, tagging section headings as headings and not just bold text in big font. Since this might cause issues with some packages, the new behaviour is opt-in.

If you write `\DocumentMetadata{}` before the `\documentclass` declaration, you opt in to these new features. This command also enables new functionality in e.g. the hyperref package.

1.4.1 Document classes

Good practices
Journals commonly define their own document classes, based on one of the standard classes. The discussion below only applies to the documents where you are in charge of the style. Always look at the journal instructions for preparing the published version of your article.

There are three document classes of interest included in base LaTeX:

article As the name suggests, this should be used for articles, short notes, and such. The `\maketitle` command does not create a separate title page, and the highest-level sectioning command is `\section`. There is no page break between sections.

report This class is suitable for e.g. a thesis, lecture notes, or a longer article. The present notes use this class. There is a separate title page, and the highest-level sectioning command is `\chapter`. Each chapter begins on a new page.

book This is the heaviest of all the classes. You get a lot of empty pages, just like in a real printed book. If you find yourself using this class, you have agreed to something monumental.

There is also a **letter** document class, which understandably sees little use nowadays. But it is still there if you need it!

Options passed to document classes are interesting in that *they are also passed to all packages*. If you pass `a4paper` to the document class, then you don't need to pass it again to the geometry package.

Some options are supported by all the basic document classes:

a4paper This is self-explanatory. By default, LaTeX assumes the American letter paper size.

10pt, 11pt, 12pt Sets the body font size. Fonts for section headers, footnotes, etc. are scaled accordingly. The default is 10 points.

oneside, twoside Whether the margins are equal or alternating between odd and even pages. For example this document (which is aimed for consuption on a screen) uses `oneside`, but a printed report should specify `twoside`.

openright, openany Whether chapters always begin on a right-hand side page (default for book) or any page (default for report). You should consider `openright` for a printed thesis and `openany` for electronic version. Not applicable to the article class where chapters are not supported at all.

notitlepage, titlepage Whether the title is set on a separate page.

fleqn Instead of centering, left-aligns display formulas.

twocolumn Sets the text in two columns.

A very common alternative document class is **amsart**. It is developed by the American Mathematical Society, and some prefer its style to that of **article**. It is a drop-in replacement for **article** and supports the same options.

Other common classes include those from koma-script (drop-in replacements for all standard classes) and memoir (intended for longer documents), which both provide a lot of flexibility. They both come with extensive documentation for those who wish to venture into that rabbit hole.

1.4.2 Loading packages

In the previous example, we loaded the hyperref package with:

```
\usepackage[hidelinks]{hyperref}
```

Options can be passed to packages inside the brackets. Here we pass the `hidelinks` option that suppresses the coloured boxes around clickable links. Some packages also take options with a `key=value` notation.

In addition to the options passed explicitly, all the options passed to the document class are also forwarded to the package.

Good practices
You should always load hyperref. It not only turns cross-references into clickable links, but also adds section headers to the PDF table of contents (usually found in the sidebar of any reader application). Your readers will appreciate this navigation aid.

CTAN, the *Comprehensive TEX Archive Network*, is the repository for LaTeX packages. It can be browsed at `www.ctan.org`.

Generally LaTeX distributions either contain the whole CTAN, or are able to download packages from there as needed. You should check out how you can keep your distribution up to date, as new versions of packages are continually released.

Most importantly, all major packages listed on CTAN have good documentation there. The documentation typically includes usage examples, a complete reference, and possible compatibility issues with other packages. For example, you can find the (very extensive!) documentation of hyperref at `www.ctan.org/pkg/hyperref`. They are also available at `www.texdoc.org`.

babel Support for languages other than English. See Section 3.1.

enumitem Customization of list structures. See Section 2.6.

graphicx Inclusion of image files. Replaces the old graphics package of early nineties LaTeX; do not confuse the two. See Section 7.1.

mathtools More mathematical environments and commands. Superset of the amsmath package, which is loaded by AMS document classes. See Chapter 4

xcolor Support for setting text and symbols in colour. Replaces the old color package included in early LaTeX. See Section 2.4.

Gotcha!
Quite a few packages modify the standard LaTeX commands and even the commands defined by each other. Sometimes it is important to load packages in the correct order. The core packages are usually compatible with each other, but you should always check the package documentation for possible conflicts.

Two notorious examples:

- babel should be loaded very early,
- cleveref must be loaded after hyperref and theorem packages.

1.5 Creating your own style file

Good practices
As discussed above, it is best to keep an article in one file. The same can be said also of a thesis.

You should only create a style file if you have multiple documents sharing commands and packages. For example, a style file is useful for a personal notes folder.

Internally, LaTeX packages live in `.sty` files. It is possible to move preamble code into such a file. If the file is in the same directory as the document,[8] it can then be accessed with `\usepackage`.

For example,

```
\usepackage{mystyle}
```

loads the file called `mystyle.sty`.

Inside the style file, there are a few important things. First off, the file is started with a special identification header:

```
\NeedsTeXFormat{LaTeX2e}[2020/02/02]
\ProvidesPackage{mystyle}[2024/05/20 mystyle]
```

These tell TeX that the file requires LaTeX released on 2 February 2020 or later, and that the file contains the version of `mystyle` published on 20 May 2024. (Keeping the latter number up to date is relevant only for package writers, but it is nice to keep in sync for your own reference.)

[8] Or on TeX's search path, but don't ask me about that.

After this, you can do everything that you could do in document preamble. Ideally, you would define your favourite commands here. One change is that loading packages is done with `\RequirePackage`.

As a complete example, here is a heavily shortened version of the style file for these notes.[9]

```
\NeedsTeXFormat{LaTeX2e}[2020/02/02]
\ProvidesPackage{latexcourse}[2024/03/05 latexcourse]

% A simple TODO command, available only when the 'draft' option is set
\DeclareOption{draft}{
    \newcommand{\todo}[1]{\textcolor{red}{\textbf{[TODO: #1]}}}
}
\ProcessOptions\relax

\RequirePackage{mathtools}
\RequirePackage{csquotes}
\RequirePackage[finnish,french,english]{babel}
\RequirePackage[hidelinks]{hyperref}

% Theorem styles for examples
\RequirePackage{amsthm}
\RequirePackage{thmtools}
\declaretheorem{theorem}
```

In this example, you could also see how a simple boolean option can be declared. It is enabled with `\usepackage[draft]{latexcourse}`. We do not discuss further package development topics here.

1.6 Counters

LaTeX keeps a lot of internal state in integer variables called *counters*.

```
We are in Chapter~\arabic{chapter}
and its Section~\arabic{section}.

\begin{enumerate}
\item This is item~\arabic{enumi}.
\item This is item~\arabic{enumi}.
  \begin{enumerate}
  \item Subitem~\arabic{enumii}.
  \item Subitem~\arabic{enumii}.
  \end{enumerate}
\end{enumerate}
```

We are in Chapter 1 and its Section 6.

1. This is item 1.
2. This is item 2.
   (a) Subitem 1.
   (b) Subitem 2.

[9] See the introduction for a link to the full source code!

The counters seen in the above example are automatically stepped by the sectioning and list commands. They can be manually manipulated with the `\setcounter` command. One such example is seen in Väisälä's topology textbook:

```
\setcounter{chapter}{-1}
\chapter{Prerequisites}

% The chapter name is printed as "0. Prerequisites"
% Note: the counter is set to -1 to compensate for \chapter stepping it
```

It is also possible to add/subtract a value to a counter:

```
\begin{enumerate}
\item Add one and step:
  \stepcounter{enumi}
\item Add seven and step:
  \addtocounter{enumi}{7}
\item Minus one + step:
  \addtocounter{enumi}{-1}
\item Same as above!
\end{enumerate}
```

1. Add one and step:
3. Add seven and step:
11. Minus one + step:
11. Same as above!

There are many ways to output the value of a counter:

```
Section~\arabic{section}\\
Section~\alph{section}\\
Section~\Alph{section}\\
Section~\Roman{section}\\
Section~\fnsymbol{section}
```

Section 6
Section f
Section F
Section VI
Section ‖

The last command `\fnsymbol` is used for footnote symbols. Note that the letter-based styles can be capitalized by using `\Roman` instead of `\roman` etc.

To format numbers as natural-language strings, you can use the fmtcount package. For example, its `\ordinalstring` command produces strings like "first", "second", and so on. There are several capitalization variants, as well as support for a few languages (including grammatical gender).[10] The package also provides commands for binary and hexadecimal number formatting.

To define your own counters, you can use `\newcounter`:

[10] Unfortunately, Finnish is not yet among them.

```
\newcounter{fact}
\newcommand{\axiom}[1]{\stepcounter{fact}%
    \textbf{Rule \thefact.} #1\par}

\axiom{Don't use plain \TeX.}
\axiom{See the above.}
```

Rule 1. Don't use plain TEX.
Rule 2. See the above.

As we saw above, the value of the counter is accessed by prefixing it with `\the`. The style of the counter can be changed by redefining this command:

```
We're at Fact number~\thefact.

\renewcommand\thefact{\Roman{fact}}

Now it's called Fact~\thefact.
```

We're at Fact number 2.
Now it's called Fact II.

If you want the counter to reset when another counter is stepped, you can pass an optional argument (*after* the counter name!) to `\newcounter`. An example use case is to reset the counter every time a new section is started (in which case the optional argument would be `[section]`):

```
\newcounter{exc}[fact]
\renewcommand{\theexc}{\roman{exc}}
\newcommand{\except}[1]{\stepcounter{exc}%
  {\tiny Exception~\theexc. #1}\par}

\axiom{Don't use plain \TeX.}
\except{Unless Don Knuth passes by.}
\except{Or you're a masochist.}
\axiom{See the above.}
\except{No exceptions.}
```

Rule 1. Don't use plain TEX.
Exception i. Unless Don Knuth passes by.
Exception ii. Or you're a masochist.
Rule 2. See the above.
Exception i. No exceptions.

Gotcha!
This only sees the effect of `\stepcounter` on the another counter; if you use `\setcounter` to change it, the dependent counter is not reset.

To customize the resetting of pre-existing counters, you can use the `\counterwithin` command. For example, to prefix the equation numbers with section numbers, you would put in the preamble

```
\counterwithin{equation}{section}
```

This effect can be undone with the `\counterwithout` command:

```
\counterwithout{equation}{section}
```

You can also pass an optional formatting command. If you like your equations in Roman numbers within section (like 1.iv), then you'd set

```
\counterwithin[\Roman]{equation}{section}
```

The `\counterwithin` command does essentially something like

```
% \counterwithin[\Roman]{equation}{section} =
\counterwithin*{equation}{section}
\renewcommand{\theequation}{\thesection.\Roman{equation}}
```

If you'd prefer not to have the within-counter as prefix (`\thesection.`), you can use the starred `\counterwithin*` form, or renew the format. This code numbers equations like (1), (2), …, and resets the numbering in each section:

```
\counterwithin{equation}{section}
\renewcommand\theequation{\arabic{equation}}
```

Technical side note
Counters are always in a global scope. That is, the definition of a counter does not disappear as the group is closed. If you would like to use the same counter name again in a later context, you must reset and reuse the previous counter.

Due to this, you should probably define your counters in the preamble.

Technical side note
The `\label` command creates a label referring to the "latest" updated counter. The usual `\stepcounter` command does not update this tag; instead, the LaTeX sectioning commands etc. use the `\refstepcounter` command. This command steps the counter and updates the "last stepped counter" tag.

1.7 *Further topics**

1.7.1 Control flow

TEX does have syntax for basic control flow like *if* statements, but the ifthen package makes it much nicer to use. The `\ifthenelse` command takes three arguments: an expression, what to do if it is true, and what to do if it is false.

```
\ifthenelse{\thepage>10}
    {We're past page~10.}
    {We haven't passed it yet.}
\ifthenelse{\thepage>50}
    {We're also past page~50.}
    {Yet page~50 is still ahead.}
```

We're past page 10. Yet page 50 is still ahead.

The syntax allows basic comparisons of counters with `<`, `=`, and `>`. There is also the `\isodd` command that is useful with page numbers. Lengths (Section 3.2.1) can be compared by wrapping the comparison inside `\lengthtest`. See the package documentation for all expressions.

It is also possible to define and test boolean variables that have value equal to `true` or `false`:

```
\newboolean{MyBool}
\ifthenelse{\boolean{MyBool}}
    {It is true!}{It is false!}

\setboolean{MyBool}{true}
\ifthenelse{\boolean{MyBool}}
    {It is true!}{It is false!}
```

It is false!
It is true!

The package also provides a `\whiledo` command that provides a primitive loop.

```
\newcounter{mynumber}

\whiledo{\themynumber<6}{%
\stepcounter{mynumber}
\arabic{mynumber} is \Roman{mynumber}\\}
```

1 is I
2 is II
3 is III
4 is IV
5 is V
6 is VI

However, in many cases the `\foreach` command provided by pgffor is easier. Since the pgffor package is part of TikZ, we discuss it along TikZ in Section 8.4.

1.7.2 Robustness

The design of TEX as a macro language poses some challenges. One is controlling the time and place where macros are expanded.

Some commands like `\section` store their contents in the auxiliary file: the section title is also used for setting the table of contents. Commands like `\ifthenelse` cannot be used in this context. They are called *fragile* commands.

The commands that expand to text are generally robust. Similarly, most LATEX commands are nowadays robust.

It is very rare to see a robustness issue, but its telltale signs are very confusing error messages caused by using a command inside a sectioning command, figure caption, or table specification. If you really need to use a command there, you can try prefixing its use with `\protect`.

1.7.3 More on command and style definitions

LATEX3 provides a completely revamped system for defining commands. If you want optional arguments or starred forms, you should use the new system. For environments, it offers a clearer syntax for parameters and even capturing the environment contents. It is also a bit more robust in places.

The new system is documented in "*LATEX for authors*" distributed with LATEX. It can be accessed by the name `usrguide` at `www.texdoc.org` or with the `texdoc` command-line utility.

Chapter 2

Styling text

2.1 The not so minor characters

This is not a style guide, but let us still first talk about punctuation. There are some characters that look very similar but are actually different.

Good practices
Stylish use of punctuation makes your text look so much more professional. Some (but not nearly all) journals employ copy editors who have a keen eye for these issues.

The hyphen -, coded as `-`
Use this in hyphenated compound words (quasi-invariant) and when a single person's name has many parts (Eric Vanden-Eijnden).

The en dash –, coded as `--`
Slightly longer than the humble hyphen, this is used for numeric intervals (2017–2021) and separating several persons' names (Cauchy–Schwarz inequality).

The em dash —, coded as `---`
The longest of them all, used generally only for parenthetical remarks.[1]

The apostrophe ', coded as `'`
Used in English to indicate omission of letters (I've) and possessive form (Grönwall's inequality).

The single quotes '', coded as `` ` `` `'`
Note this: when a single word is 'quoted', the opening and closing quotes look slightly different. On a Finnish keyboard, you get the opening character by pressing Shift+`.

[1] Parenthetical remarks are usually written—such as here—with an em dash and no space, or – like here – with en dash and spaces, but please consult a style guide.

The double quotes “”, coded as `` ''
This effect is even more exaggerated with “double quotes”; if you use the " symbol on your keyboard, you get ”not as nice” quotes.

2.1.1 Special characters

Most characters reserved for LaTeX commands can be produced by prefixing them with a backslash: % is produced by `\%`, # by `\#`, $ by `\$`, and & by `\&`. The only exception is \, which is given by `\textbackslash`.

There is also a whole lot of special characters and symbols accessible through special commands: for example, © via `\textcopyright{}`. To find these, I highly recommend the Detexify tool.[2]

Since LaTeX is nowadays Unicode-native, you can write letters like åäö directly into your source code. If you can't find the specific letter on your keyboard, or the journal you're submitting to uses age-old tools, the old commands for producing accents are still there. These are useful for e.g. author names in bibliographies. You can find the full list online;[3] some common examples are here:

```
Curriculum Vit\ae\\
\r{A}bo Akademi\\
Wac\l{}aw Sierpi\'{n}ski\\
Bernt \O{}ksendal\\
Besan\c{c}on\\
\v{S}koda\\
Dotless \i
```

Curriculum Vitæ
Åbo Akademi
Wacław Sierpiński
Bernt Øksendal
Besançon
Škoda
Dotless ı

TeX also does some typographical tweaks on its own. Look at the ‘ffi’ in ‘efficient’ – the letter shapes have been merged. This is called a *ligature*.

Ligatures improve legibility, but if you need to suppress them, you can stick a `\/` between the letters. This is sometimes practiced for English words like ‘shelfful’, where there is a word boundary between the f's.

```
Efficient\\
Ef\/ficient\\
Ef\/f\/icient
```

Efficient
Efficient
Efficient

---

[2] `detexify.kirelabs.org/classify.html`
[3] `en.wikibooks.org/wiki/LaTeX/#Special_Characters`

2.1.2 Dingbats and logos

There are several packages providing symbol characters. One notable example is fontawesome6, which packages the version 6 of the Font Awesome symbol font.[4] Once the package is loaded, symbols can be set with names prefixed with `\fa`:

```
\faBook{} Reading suggestions\\
\faMugSaucer{} (Have a cup at hand)\\

There are also brand logos like
\faGit~Git and \faLinux~Linux,
but remember that they are trademarks.\\

Font size commands are supported:
{\tiny\faPoop}~{\faPoop}~{\Large\faPoop}
```

Reading suggestions
(Have a cup at hand)

There are also brand logos like Git and Linux, but remember that they are trademarks.

Font size commands are supported:

The full list of symbols is included in the documentation. The package includes the free set of symbols, but supports a larger set if you happen to own the commercial version of the font. There are also alternative versions of some symbols.

What about emoji? It is possible to include emoji in LaTeX documents. Emoji are just special Unicode characters; the problem is whether the font supports them. The inclusion of emoji will be discussed below in Section 6.3.4.

2.2 Style and size

2.2.1 Styles

LaTeX provides three independent attributes for modifying the appearance of text. These are:

Family The basic shape of the font. There are three families by default: serif, sans serif, and `typewriter`. In these notes, the sans serif font is used to highlight package names, and the typewriter font is used for LaTeX code.

[4] As of May 2025 this is the newest version. There are also LaTeX packages corresponding to versions 5 (fontawesome5) and 4 (fontawesome). Note that some icons have been renamed between versions.

Series This is the combination of weight and width of the font. Generally, one only needs to care about the normal and **bold** series. Those into graphical design know that this is only the tip of the iceberg; the methods in Section 6.3 can create more series.

Shape Usual choices include upright, *italic*, and SMALL CAPITALS. Technically, small caps is not a shape variant but an independent style altogether, but we are simplifying things here.

These can be freely combined (although non-default fonts might not support all combinations):

```
\textsf{\textit{italic sans serif}}\\
\textbf{\texttt{bold typewriter}}
```

italic sans serif
bold typewriter

The following table shows the most common font commands. These come in two forms: the first takes the text as its argument, and the second applies the change to all the following text (until end of scope; see p. 17).

Command form	Declarative form	Output
`\textrm{...}`	`{\rmfamily ...}`	Serif family (default)
`\textsf{...}`	`{\sffamily ...}`	Sans serif
`\texttt{...}`	`{\ttfamily ...}`	`Typewriter`
`\textbf{...}`	`{\bfseries ...}`	**Bold series**
`\textit{...}`	`{\itshape ...}`	*Italic shape*
`\textsc{...}`	`{\scshape ...}`	SMALL CAPITALS

Technical side note
Prefer the command forms. They are easier to parse for automatic tools like **latexdiff**, and moreover `\textit` does a tiny typographic tweak called *italic correction* at the interface between italic and upright styles:

```
\textit{Super}charged\\
{\itshape Super}charged
```

*Super*charged
*Super*charged

There are two important points to remember: First, you should keep the semantics and the styling separate. For example, these notes define a

\pkg command to highlight package names; there are no \textsf commands sprinkled around the code.[5]

Another is that you should not use \textit to emphasize text. It is much better to use \emph: first, it is semantically meaningful, and second, it supports nesting:

```
Sometimes you need to
\emph{emphasize} a point.
\emph{And sometimes your emphasis
contains \emph{an even more important}
point to consider.}
```

Sometimes you need to *emphasize* a point. *And sometimes your emphasis contains* an even more important *point to consider.*

If you need a declarative form (for use in an environment), it is \em.

2.2.2 But what about underlining?

LaTeX does not provide a command to underline text. Many typography enthusiasts consider underlining bad style, an unfortunate byproduct of mechanical typewriters where you could not (easily) change the font.

If you still want to underline things, there is the ulem package. By default, it rewrites \emph to use (nested) underlining instead of italic, so you might want to load it as \usepackage[normalem]{ulem}, which keeps \emph as it usually is.

In addition to \uline, this package also provides the \sout command that lets you strike out text. There are also several styles of underlines:

```
\uline{Underlined}\\
\uuline{Doubly so}\\
\sout{Struck out}\\
\uwave{Waves}\\
\dashuline{Dashes}\\
\dotuline{Dots}
```

Underlined
Doubly so
~~Struck out~~
Waves
Dashes
Dots

2.2.3 Size

Coming from any graphical word processor or layout program, you have seen fonts being measured in *points*. LaTeX does support it (see Section 6.3 where we talk about font customization), but again it works against separating content from presentation.

[5] Moreover, this command adds an entry to the index. Thus the code is not only semantically meaningful but also benefits the reader.

Instead, LaTeX offers a range of sizes relative to the default document font passed as the class option (Section 1.4.1). These are as follows:

`\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

You mostly need these when making figures/tables/source code listings or customizing heading styles. In a manuscript for a journal you probably should not mess around with font sizes. Putting huge text in the middle of a paragraph breaks the line spacing and might cause you enemies in graphical design circles.

2.2.4 Verbatim text

If you need to display source code in your work, the listings package is your friend. This package offers a lot of customization, so you should check out its documentation on CTAN. We content ourselves with a very *meta* example, displaying the first lines of this lecture note itself:

```
\lstset{basicstyle=\tiny\ttfamily,
  commentstyle=\color{gray},
  breaklines=true,
  prebreak=\mbox{\tiny$\searrow$}}

\lstinputlisting
  [language={[LaTeX]TeX}, lastline=11]
  {2nd-course-in-latex.tex}
```

```
% The \DocumentMetadata command is ↘
      used to activate LaTeX3 features
\DocumentMetadata{lang=en}
\documentclass[a4paper, 11pt]{report}

\usepackage{latexcourse}
\usepackage{tocbibind}
\makeindex

\addbibresource{2nd-course-refs.bib}

\hypersetup{href/protocol=https://,
```

It is possible to either embed the source code within your `.tex` file by using the `lstlisting` environment, or to input the contents directly from a source code file like above.

The package offers some support for making listings float like figures, or alternatively you could define your own float environment (see Section 7.2.2). Computer science journals typically have their own instructions for this.

The `lstlisting` environment is a special variant of the `verbatim` environment. All contents inside this environment are printed verbatim in the typewriter font: LaTeX commands and special characters are printed as is. (Except of course for the `\end{verbatim}`. It would be silly not to evaluate that.)

```
\begin{verbatim}
Printed as is, including
  whitespace.
\LaTeX & $math$
\end{verbatim}
```

```
Printed as is, including
  whitespace.
\LaTeX & $math$
```

There is also the `\verb` command for short spans of verbatim text. This command has a special syntax: it must be followed by a special character (or a number, but not `*`). This special character ends the verbatim span. Obviously, you need to choose a character that does not appear in the text! There is also a starred form that makes the whitespace visible.

```
\verb+\LaTeX{}+ \\
\verb|$1 + 1$| \\
\verb*|$1 + 1$|
```

```
\LaTeX{}
$1 + 1$
$1␣+␣1$
```

These notes use a lot of `\verb|...|` in the code (and the preceding was coded as `\verb+\verb|...|+`, which in turn was coded as…).

The `verbatim` environment is a more lightweight way to display source code, but it offers no customization at all. The fancyvrb package provides a lot of that customization. However, the listings package additionally provides automatic syntax highlighting for many common programming languages, so it should be your first choice.

One neat use of verbatim environments is to comment out large blocks of text. This example requires fancyvrb and its capitalized `Verbatim` environment:

```
Some text.
\begin{Verbatim}[lastline=0]
You cannot see what's happening here.
It's as if it didn't exist!
\end{Verbatim}
Some more text.
```

Some text.

Some more text.

This can be further simplified with the `\DefineVerbatimEnvironment` command, which creates a customized environment based on `Verbatim`:

```
\DefineVerbatimEnvironment
    {CommentOut}{Verbatim}{lastline=0}

Did you hear something?
\begin{CommentOut}
Nothing to see here.
\end{CommentOut}
I thought I heard something.
```

Did you hear something?

I thought I heard something.

Technical side note
Verbatim environments are implemented with dark TEX magic: essentially, they need to override the entire TEX parser to do their work. This causes a lot of limitations: for example, you cannot use `\verb` as an argument to a command. There are ways to override these issues; see `\SaveVerb` and `\UseVerb` in `fancyvrb` documentation.

2.3 Spaces, line breaks, and paragraphs

TEX ignores consecutive whitespace characters. It does not matter if you put one or two spaces between words, or even start a new line – only one space is printed. For a description of the spacing algorithm, and the ways to produce visual space of fixed space, you should go to Chapter 3.

Good practices
I personally like to put every sentence on a new source code line. I find it easier to scan, and it also goes well with Git version control software: it highlights changed lines between revisions, so there is a one-to-one correspondence between changed lines and changed sentences.

An empty line starts a new paragraph. Since TEX handles paragraph spacing, you should almost never need manual line breaking commands. In some esoteric corner cases (like macro definitions), you can also use the `\par` command to cause a paragraph break.

Gotcha!
Be careful with line breaks in your macro definitions; you might accidentally end up causing paragraph breaks. If your macro definition spreads over many lines, you should tell TEX to ignore the line breaks by ending each line with `%`.

For customization of the space between paragraphs and indentation, again look at Chapter 3 and specifically Section 3.3.2. If you need to suppress indentation of a single paragraph, you can use `\noindent` at the beginning.

```
\setlength\parindent{2em}
\setlength\parskip{0pt}

This is a long example paragraph.

This too is a long example paragraph.

\noindent
This paragraph is exceptional,
as it is not indented.
```

This is a long example paragraph.

This too is a long example paragraph.

This paragraph is exceptional, as it is not indented.

If you really need a manual line break without starting a new paragraph, you can use either the `\\` or the `\\*` command. The difference between the two is that the starred form prevents a page break between the lines. (That is, you probably need to use the latter.) These commands take an optional argument for the space after the line break:

```
Yours sincerely,\\*[8pt]
P.~Laarne\\*
\TeX nical assistant
```

Yours sincerely,

P. Laarne
TEXnical assistant

2.4 Colour

Due to historical and technical reasons, TEX does not natively support colours. The xcolor package adds this support. Many other packages load xcolor automatically and support the colour syntax (for example tikz).

Gotcha!
Be careful with the spelling: the color package is the original 90s implementation, and its use is discouraged.

Like with the font commands, there are two forms of the colour specification: `\textcolor` that colours only its (second) argument, and `\color` that colours everything until the current scope ends. There are three basic ways and several advanced ones to define colors:

By name `\textcolor{blue}{...}` prints the text in blue. By default, only a few basic names are loaded. If you load the package with `[svgnames]` option, you get a lot more colours such as DarkOrange. Check out the Wikipedia article "Web colors" or the package documentation for a full list. The capitalization is significant!

By RGB value `\textcolor[rgb]{0.24,0.7,0.44}{...}` sets the intensities of red, green and blue pixels to 24 %, 70 % and 44 % respectively, yielding something like this shade described as medium sea green. In this model black is `{0, 0, 0}` and white `{1, 1, 1}`.

By CMYK value This one is only for those into printing presses; the optional argument is `[cmyk]` and syntax is similar to the RGB model. The four parameters correspond to cyan, magenta, yellow and key (black) inks on white paper.

By grayness `\textcolor[gray]{0.6}{...}` where 0 is black and 1 is white; for example, 60% lightness.

By HSB value Here the keyword is `[hsb]` and the parameters are hue, saturation, and brightness.

By HTML code This takes the six-letter hexadecimal string colour string, such as `\color[HTML]{2ABC4D}`.

By wavelength Science nerds can use `\textcolor[wave]{410}` to approximate the light at 410 nm wavelength.

The package also offers a lot of ways to mix colours together. As you probably learned in primary school, this generally procudes mainly shades of greyish brown, so we do not talk about it here.[6] The only exception is the syntax `\textcolor{blue!60}{...}` which blends blue with 40 % white.

It is possible to define custom colour names with `\definecolor`. This can be useful e.g. when creating TikZ pictures with recurring elements. Here we follow the University of Helsinki style guide and define the brand colour of the Faculty of Science:

```
\definecolor{Sciency}{cmyk}{0,0.46,1,0}
\textcolor{Sciency}{Science!}
```

Science!

---

[6] Another reason is that the result depends on the specific colour model. Taking averages of RGB values can yield a very different result to averaging equivalent HSB values. There is no 'correct' way to do this; in fact, trying to understand what colour *really is* leads you to a rabbit hole of mathematics, human perception, and questioning the reality.

Good practices
Never use colour as the sole means of indicating information. Many people read papers in black-and-white prints or on ebook reader with grayscale screen. Red-green colourblindness, the most common form of colour vision deficiency, affects about 1 in 12 males.

Even then, pay attention to the colour contrast. Low-contrast text might be impossible to read e.g. on screen in bright sunlight, or for a person with a vision deficiency.

2.5 Footnotes

Footnotes are produced with the imaginatively named `\footnote` command. The command itself needs no explanation, but we remark on a few points on placement and styling of footnotes.

Gotcha!
Whitespace immediately preceding the footnote command is interpreted as a word space as usual. If you put a footnote on a separate code line, the line break is interpreted as a space. This causes separation between the preceding word or punctuation.

The easy fix is to put a `%` before the whitespace, since it causes the whitespace and line break to be swallowed.

```
An ugly space.
\footnote{First note!}\\

Tightly together.%
\footnote{Second note!}
```

An ugly space. [a]

Tightly together.[b]

[a] First note!
[b] Second note!

Footnotes cannot be placed everywhere. For example, footnotes cannot be nested: the footnote mark appears but the text is placed nowhere. Similarly, they cannot be used with most table environments (but see Section 7.3.2 for more on this!).

In these cases the footnote mark can be placed with `\footnotemark` and the footnote text defined outside the environment with `\footnotetext`. Do note that now you are responsible for keeping the footnote symbols and texts in sync!

```
Some text.%
\footnote{A footnote.\footnotemark}
\footnotetext{A nested footnote.}
```

Some text.[a]

[a] A footnote.[b]
[b] A nested footnote.

Another potentially surprising behaviour is that `minipage` environments have independent footnote numbering. You can see this with the examples in these lecture notes; their footnotes are marked alphabetically whereas standard footnotes are numeric. You can use `\footnotemark` to create a "standard footnote" reference inside the environment, and define the footnote contents with `\footnotetext` immediately after the environment.

If you want to customize the style and placement of footnotes, the footmisc package should be your first stop. It offers a range of footnote symbols, possibility for per-page footnotes, and even placement of the footnotes in page margin instead. In two-page layout, the ftnright package can be used to gather footnotes in the right column.

Generally, footnotes are avoided in mathematical literature. In some fields, such as law and linguistics, it is common to have extremely long and nested footnotes. In those cases the manyfoot and bigfoot packages can be used; they are similar with slight differences, but I cannot comment on those.

Finally, there is the enotez package for printing end notes or chapter notes.

2.6 List structures

LaTeX provides three kinds of list structures by default:

`itemize` A bulleted list.

`enumerate` A numbered list.

`description` This style of list.

The optional argument to the `\item` command provides the content for the "bullet symbol":

```
\begin{itemize}
\item[$\clubsuit$] Club
\item[$\spadesuit$] Spade
\item[$\heartsuit$] Heart
\item[$\diamondsuit$] Diamond
\end{itemize}
```

- ♣ Club
- ♠ Spade
- ♡ Heart
- ♢ Diamond

List elements may contain multiple paragraphs, and lists can be nested. As we discuss in Section 1.6, the running numbers can be accessed with the `enumi`, `enumii`, and `enumiii` counters (corresponding to levels of nesting).

The enumitem package offers a lot of customization for the standard environments. It adds an optional argument with key-value syntax to the standard environments.

```
\begin{enumerate}
\item Default list
\item Nothing to see here
\end{enumerate}

Some text in between.

\begin{enumerate}[resume,noitemsep]
\item Numbering continued!
\item Tighter spacing!
\end{enumerate}

\begin{description}[labelwidth=2cm]
\item[Label] Extra space
\item[Long label] Aligned!
\item[Very long label] Uh-oh.
\end{description}
```

1. Default list
2. Nothing to see here

Some text in between.

3. Numbering continued!
4. Tighter spacing!

Label Extra space

Long label Aligned!

Very long label Uh-oh.

You can also use the package to customize labels. Note that you need to use a starred form of the counter command (these were defined on page 31).

```
\begin{enumerate}[noitemsep,
  format=\bfseries,
  label={Rule~\arabic*.}]
\item Don't use plain \TeX.
\item See the above.
\end{enumerate}
```

Rule 1. Don't use plain TEX.
Rule 2. See the above.

Instead of repeating yourself at every list environment, you can use the `\setlist` command to customize all of the lists:

```
\setlist[enumerate]{noitemsep}

\begin{enumerate}
\item Tightly\dots
\item \dots{}spaced.
\end{enumerate}

Some text

\begin{enumerate}
\item Same\dots
\item \dots{}here.
\end{enumerate}
```

1. Tightly…
2. …spaced.

Some text

1. Same…
2. …here.

It is also possible to define your own customized list environments. Here the starred `label*` argument means: "append this to the label of the previous list level". We then use the `\setlist[rules,1]` command to further customize the first level.

```
\newlist{rules}{enumerate}{2}
\setlist[rules]{noitemsep,
    format=\bfseries,
    label*=\arabic*.}
\setlist[rules,1]{noitemsep,
    label={Rule~\arabic*.}}

\begin{rules}
\item Don't use plain \TeX.
\item See the above.
\begin{rules}
\item Unless Don Knuth passes by.
\end{rules}
\end{rules}
```

Rule 1. Don't use plain TEX.
Rule 2. See the above.

Rule 2.1. Unless Don Knuth passes by.

2.7 Sectioning commands

You are already familiar with the basic sectioning commands `\section`, `\subsection`, and their starred variants (that produce no section number). Additionally, the `book` and `report` document classes define `\chapter` as the top level. Typically, chapters begin on a new page (right-hand page if `openright` class option is set).

There are also the finer-level `\subsubsection` and `\paragraph` commands, but journals usually discourage more than three levels of sectioning, and even `\subsubsection` is usually too fine-grained.

Good practices
Always use the standard sectioning commands. They are semantically meaningful, they take care of numbering and cross-references, and they are used to build a PDF navigation bar (when hyperref is loaded).

Gotcha!
If your section title contains mathematical symbols and you have hyperref loaded (as you do, right?), you will probably get a warning. This is because the PDF navigation bar format does not support mathematical symbols.

To solve this issue, you can use the `\texorpdfstring` command. It takes two arguments: the first is the one to show in the document, and the second is the (non-mathematical) alternative to show in PDF navigation.

```
\section{The \texorpdfstring{$\phi^4$}{phi4} measure}
```

This code displays "The ϕ^4 measure" in the document but "The phi4 measure" in the navigation bar.

You can pass a shorter version of the section title as an optional argument. It will be used in the table of contents and page headers (if enabled).

```
\section[Principia summarized]{Philosophi\ae{} Naturalis
                               Principia Mathematica summarized}
```

Appendices are quite common in mathematical papers. The `\appendix` command resets the top-level numbering and further sections/chapters are numbered alphabetically.

```
\section{Introduction} % Section 1
\section{Methods} % Section 2
\section{Results} % Section 3

\appendix

\section{Derivation of results} % Section A
\subsection{Difference quotients} % Section A.1
```

The `book` document class also provides the `\frontmatter`, `\mainmatter`, and `\backmatter` commands. These are only relevant for heavy-duty book authoring; your PhD thesis should probably use the `report` class.

Pages in the front matter are numbered with Roman numerals and chapters are unnumbered. The `\frontmatter` command should appear immediately after `\begin{document}`. After `\mainmatter`, pages and chapters are numbered usually. Appendix numbering can be enabled with the `\appendix` command as before. Finally, bibliographies and other closing stuff can be put after `\backmatter`.[7]

2.7.1 Customizing headings

To customize heading styles, it is easiest to use the titlesec package. It provides two ways to customize headings: a "simple" and an "advanced" interface.

Gotcha!
If you use the memoir or koma-script document classes, this package is not usable. The document classes provide their own mechanisms instead.

The simple interface is used with the `\titleformat*` command. It takes two arguments: the sectioning command and formatting commands to apply to it. The formatting commands should use the declaration form.

For example, to set centered subsection headers with a large small caps typeface, you would put in the preamble:

```
\usepackage{titlesec}
\titleformat*{\subsection}{\centering\Large\scshape}
```

The simple interface does not let you customize the layout of headings, and it does not permit customization of chapter headings (which spread on two lines by default). For these cases, there is the advanced interface. It uses the unstarred `\titleformat` command, which takes up to seven parameters:

- The heading command to modify.
- (Optional.) The shape of the heading. The default is `hang`, which works like the default for sections. For centered version, `block` is preferable. The `display` option puts the content on two lines like the default for chapters; the `frame` option additionally sets a frame. Finally, `runin` places the heading within the following paragraphs; a special case is `wrap`, which additionally wraps the heading. There are some more styles, which you can find in the documentation.

[7] On the very last page of a finely set book, you might find a *colophon*. Google what it means, and you're on your way to becoming a typography enthusiast.

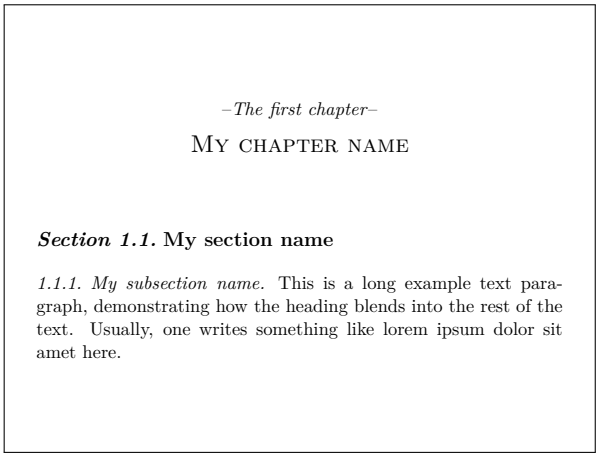
- Formatting code applied to the whole title. This might also include some commands in vertical mode.
- Formatting of the heading number. If you want to prefix the number with a string, do it here. Since chapters are called either "Chapter" or "Appendix", you can get the correct string with `\chaptertitlename`.
- Length of the (vertical or horizontal, depending on shape) separation between label and heading.
- Code to execute before the heading text.
- (Optional.) Code to execute after the heading text.

Inside this command, there are a few special considerations. To center or right-align text, you should use `\filcenter` and `\filouter`/`\filinner` instead of the standard commands. These special commands respect the separation parameters.

Let us explore these through a couple examples. Many more possibilities are shown in the package documentation.

In the next example, chapter titles are centered with `\filcenter`. The chapter number is stated as an ordinal with the fmtcount package. Section titles are customized to include the word "Section" and a full stop. Finally, subsection titles are changed to the run-in format. The `\wordsep` command gives the natural inter-word distance for the font, and the optional end-code is used to add a full stop.

```
\titleformat{\chapter}[display]{\filcenter}
    {--\emph{The \ordinalstring{chapter} \MakeLowercase{\chaptertitlename}}--}
    {4pt}{\Large\scshape}

\titleformat{\section}[block]{\large}
    {\bfseries\itshape Section~\thesection.}{4pt}
    {\bfseries}

\titleformat{\subsection}[runin]{}{\itshape \thesubsection.}
    {\wordsep}{\itshape}[.]
```

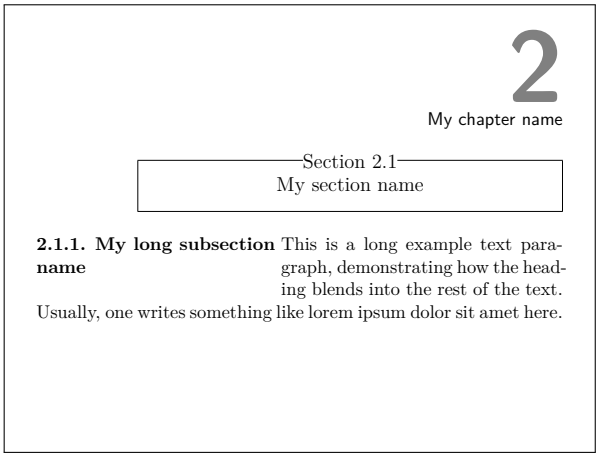


The `\titlespacing` command adjusts the spacing around the title. It takes three or four arguments:

- Addition to the left margin. For the `wrap` shape, it gives the maximum width of the heading.
- Vertical separation above the heading.
- Vertical separation below the heading.
- (Optional.) Addition to the right margin.

In the following, we move the chapter heading closer to the top of page, indent the framed section heading, and specify a wrapped subsection heading:

```
\titleformat{\chapter}[display]{\filinner\sffamily}
    {\bfseries\textcolor{gray}{\fontsize{60pt}{60pt}\selectfont\thechapter}}
    {0pt}{}
\titlespacing{\chapter}{0pt}{12pt}{12pt}

\titleformat{\section}[frame]{\large}
    {Section~\thesection}{8pt}
    {\filcenter}
\titlespacing{\section}{5em}{4pt}{12pt}

\titleformat{\subsection}[wrap]{\bfseries}{\thesubsection.}{0.5em}{}
\titlespacing{\subsection}{5cm}{8pt}{\wordsep}
```



Warning
Be careful with heading customization, especially with the run-in and wrap styles. Long headings might break in surprising and suboptimal ways.

Chapter 3

Page layout and whitespace

3.1 How TEX sets lines

The line and paragraph shaping algorithm of TEX has long been touted as excellent. The algorithm breaks lines and hyphenates words in a way that tries to keep the inter-word spacing visually pleasing. It works on paragraph level: changing a word may cause the preceding lines to be set differently, as TEX tries to find the globally best solution.

Technical side note
The algorithm has quite a few parameters that *can* be modified… but most likely *should* not. You can find them in the TEXbook [2].

Warning
The following discussion is TEXnical, and mostly for giving you an idea of how things work. If you find yourself needing to manipulate boxes directly,

1. reconsider what you are doing,
2. go read the TEXbook.

TEX lays out a line just like a human typesetter would do with metal type. The basic unit is *box*, a rectangular piece of content. TEX alternates between a horizontal mode, where the boxes are set in line, and a vertical mode, where boxes are set on top of each other.[1] Boxes are nested to produce complicated layouts.

[1] To be precise, there are two horizontal modes: the one where lines are broken automatically, and the one where they are not. Similarly, there are two vertical modes that differ in page breaking behaviour. Finally, there are the inline and display math modes, bringing the total to six.

Technical side note
We can play an old-school typesetter with the `\hbox` and `\vbox` commands. (These are low-level TEX commands – see below for LATEX versions!)

```
\vbox{\hbox{\hbox{Above1}\hbox{Above2}}\hbox{Below}}
```

Above1Above2
Below

A key concept in typesetting is the *baseline*. The bottoms of letters lay on the baseline. Some letters like 'p' have *descenders* that drop below the baseline. The *height* of a letter is the vertical extent above baseline; the *depth* that below baseline.

The height of each line is given by the *baseline skip*: the vertical distance between baselines. It is visually pleasing when lines are spread evenly, but TEX will happily give extra space when content like $\frac{1}{2}$ needs it, even though it looks bad (just look at it!). Moreover, the baseline skip might vary between pages, since TEX might try to squeeze one more line onto a page to get a better layout.

Back to boxes. I can invent three reasons to create boxes manually:

- overriding hyphenation,
- drawing a frame around something,
- doing some placement tricks.

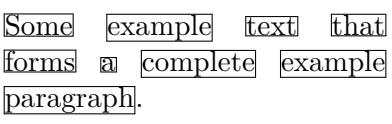
We will look at each of these use cases. You can create your own boxes with the `\mbox` command, and framed boxes with the `\fbox` command. These commands are both defined in LATEX.

```
\fbox{Framed!}
```

Framed!

The `\fboxsep` length parameter determines the padding between the box and the frame; by default it is 3 pt. There is also a `\fboxrule` parameter for the width of the line (0.4 pt by default). We can use these parameters to illustrate how TEX lays out words (ignoring hyphenation):

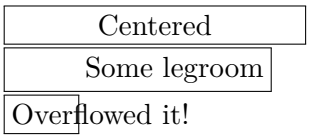
```
\setlength\fboxsep{0pt}

\fbox{Some} \fbox{example} \fbox{text}
\fbox{that} \fbox{forms} \fbox{a}
\fbox{complete} \fbox{example}
\fbox{paragraph}.
```



The `\makebox` and `\framebox` commands behave otherwise similarly, but they accept optional width and alignment parameters. By default, the text is centered. Inside these commands and their arguments, a special `\width` command gives the total width of the contents.

Do note that lines are not automatically broken inside these boxes. You can thus overflow the size:

```
\framebox[4cm][c]{Centered}\\
\framebox[1.5\width][r]{Some legroom}\\
\framebox[1cm][l]{Overflowed it!}
```



Technical side note

There are a few *niche* use cases for producing special layouts with boxes. You most likely should not use these in an article.

The first is that you can take advantage of the overflow with zero-width boxes. Here we use a right-aligned zero-width box to push stuff into the margin:

```
Some text\\
\makebox[0pt][r]{(!) }Important text\\
Some more text
```

Some text
(!) Important text
Some more text

Another is to raise or lower a box for dramatic effect:

```
My feeling \raisebox{-\totalheight}{sank.}
```

My feeling $_{\text{sank.}}$

The `\totalheight` command gives the height of the line: it is the sum of `\height` (of the contents above baseline) and `\depth` (of the contents below baseline). We will talk about baseline a bit later below.

What if you want to put multiple lines of content into a box, with automatic hyphenation? Enter paragraph boxes. The `minipage` environment essentially creates a page within page. You need to tell it the width of the box, and optionally whether the top, center, or bottom line is aligned with the baseline:

```
Some text
\begin{minipage}[b]{2.7cm}
interrupted by a rambling
note, followed by
\end{minipage}
some
\begin{minipage}[t]{2cm}
more text, albeit not
very usefully set either.
\end{minipage}
```

interrupted by a rambling note,
Some text followed by some more text, albeit not very usefully set either.

Let us also show a more practical example of how this could be used:

```
\begin{minipage}[c]{5cm}
\textbf{Curriculum Vit\ae}\\
Cira the dog\\
Born 2009\\
Expert eater of everything
\end{minipage}
\hfill
\begin{minipage}[c]{4cm}
\includegraphics[bb=2.8cm 1.4cm 7cm 6cm, clip, width=\textwidth]
    {pictures/TheDogs.jpg}
\end{minipage}
```

Curriculum Vitæ
Cira the dog
Born 2009
Expert eater of everything

Finally, let us talk about invisible boxes.

The `\phantom` command can be used to reserve space. You can see it in action in Section 4.1.2.

```
There is a \phantom{phantom} in here.\\
There is a phantom in here.
```

There is a in here.
There is a phantom in here.

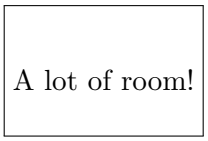
The `\rule` command creates a filled box, but it can also be used to create a *strut*: an invisible "support beam" that takes up some vertical space. The command takes the width and height of the rule, optionally preceded by the position above baseline:

```
Thick rule on baseline: \rule{1cm}{3pt}\\
Raised a bit: \rule[6pt]{1cm}{3pt}
Below baseline: \rule[-3pt]{1cm}{3pt}
```

Thick rule on baseline:
Raised a bit: Below baseline:

If we set the width to zero, then the space is reserved but no rule is drawn. Here we reserve 0.5 cm below the baseline and 1 cm above it:

```
\fbox{\rule[-0.5cm]{0pt}{1.5cm}%
A lot of room!}
```



This can be used e.g. to create some vertical clearance inside a table cell.

3.1.1 Hyphenation

By default TEX hyphenates words according to a set of English rules. It is very good at hyphenating general text, and can be trusted at it.[2] Sometimes, especially with technical compound words, it might still need some help.

To hint the possible breaking points, use the `\-` command. This disables automatic hyphenation for the particular word, so it will only be broken at the hinted points. Conversely, if a word should not be broken across lines, it can be wrapped in an `\mbox`. Finally, to prevent a line break between two words, the tilde `~` symbol produces a *non-breaking space*.

Compare these two passages (to illustrate the automatic hyphenation, they are slightly different). The breaking override commands cause the lines to be underfull, but at least for names the result is syntactically more correct.

[2] As a non-native English speaker, I don't even know what the rules are.

```
Quantum electro\-chromo\-dynamics
presented by P.~Laarne,
based on \mbox{Chatterjee}.\\

Quantum electrochromodynamics
presented by one P. Laarne,
reading about Chatterjee.
```

Quantum electrochromodynamics presented by P. Laarne, based on Chatterjee.

Quantum electrochromodynamics presented by one P. Laarne, reading about Chatterjee.

Good practices
If you add a hyphenation hint to a word, I suggest adding `\-` commands at all reasonable hyphenation points. That might save you some effort when the line breaking changes.

Let us then talk about the issues. An *overfull hbox* means that TEX could not break the lines without overflowing into the margin. Conversely, an *underfull hbox* results from a line having too little content, and thus unacceptably large spaces between words. There are a few ways to fix these issues:

- If the offending line is visually not too bad, just accept it.
- Slightly change the wording or word order to create better hyphenation opportunities.
- Manually tweak the hyphenation of a long word within the paragraph.
- If the page layout causes suboptimal paragraph shaping, it could also be tweaked, but this is an extreme action (see Section 3.3.2).

There is just one hard rule:

Warning
Only fix over-/underfull issues at the very end of publication process, when you have applied the final style file and all the content fixes are done.

Since the TEX layout algorithm is global, adding just one letter can change the breaking of a line, which can change the length of the paragraph, which can change the page layout, which then propagates to the next pages. Result: You fixed a typo, and all the fine-tuning you had done on the following pages was just lost.

I have often seen a one-word change increase the length of a paper by half a page. *The LATEX Companion* [4] production notes mention a 1 % change to a typewriter font size that broke 50 already fine-tuned pages.

By default LaTeX produces justified text that extends from the left to right margin. It is possible to let the line lengths vary (as most word processors do by default). This is done by the `\raggedright` command, or more locally inside a `flushleft` environment.

```
Here the text is spread evenly
between the two margins.
Spaces between words can vary a lot.\\

\raggedright

Now the lines are broken
once they no longer fit more text.
Spaces between words are more equal.
```

Here the text is spread evenly between the two margins. Spaces between words can vary a lot.

Now the lines are broken once they no longer fit more text. Spaces between words are more equal.

Using ragged lines is a design decision, and only applicable to documents where you are in control. There seems to be quite a lot of disagreement of the merits of justified and ragged text.

I personally used ragged text in the bibliography of my MSc thesis, as the justification made it a mess of overfull and underfull lines. Some journals have also made this choice. Your mileage may vary.

3.1.2 Language

The standard method for producing LaTeX documents in a language other than English, or even in multiple languages, is the babel package. It does several things:

- Applies the language-specific hyphenation rules;
- Translates words produced by LaTeX commands such as "Chapter", "References", and dates produced by `\today`;
- Might do some typographical fine-tuning.

The list of languages is passed as an option when babel is loaded. The last language on the list is taken to be the default language for the document. In these notes, the magic command is

```
\usepackage[finnish,french,english]{babel}
```

The full list of languages (and regional variants!) can be found in the babel documentation. Some European languages have undergone major orthography revisions in the past decades; for modern German the language code is `ngerman` and for Norwegian nynorsk it is `norsk`.

The currently active language is changed with `\selectlanguage`. The `otherlanguage` environment can be used for a short passage in a different language. Below there are two visible changes: obviously the language is different, but do note also the small space before ! in the French version.

```
This file was compiled on \today.
``Awesome!''\\

\begin{otherlanguage}{french}
Cet fiscier est compilé le \today.
\frquote{Voilà!}
\end{otherlanguage}
```

This file was compiled on June 16, 2025. “Awesome!”

Cet fiscier est compilé le 16 juin 2025. « Voilà ! »

The hyphenation rules are also changed, but the effect might be negligibly small. Overall, the hyphenation algorithm of TEX is optimized for English, so the long Finnish words might need your help.

```
Pitkähköjen yhdyssanamuotojen tavutus
oikeinkirjoitussääntöjen mukaisesti\\

\begin{otherlanguage}{finnish}
Pitkähköjen yhdyssanamuotojen tavutus
oikeinkirjoitussääntöjen mukaisesti
\end{otherlanguage}
```

Pitkähköjen yhdyssanamuotojen tavutus oikeinkirjoitussääntöjen mukaisesti

Pitkähköjen yhdyssanamuotojen tavutus oikeinkirjoitussääntöjen mukaisesti

---

Finally, since non-English languages also contain non-English characters, we need to talk about font encodings.

Gotcha!
This here is the most important place where the traditional pdfTeX compiler and the Unicode-native LuaTeX and XeTeX compilers differ. You *should always* use the fontenc package with pdfTeX, unless your document only contains the basic English characters. On the other hand, you *should not* use fontenc with LuaLaTeX or XeLaTeX, since they use a better internal encoding by default.

Since 2018, LATEX has accepted Unicode input by default. However, this only applies to the input: characters are then transformed into an internal

representation and further into individual glyphs from a font file.[3] When TEX and LATEX were developed, they had to solve the problem of having many (mathematical) glyphs, but Unicode was back then far from a universal standard.

There are several of these internal representations. The original `OT1` encoding only has 128 characters. Characters like 'ä' need to be composed of ¨`a` set on top of each other. The `T1` encoding is a more recent encoding that supports Western European languages. Cyrillic and Greek characters still need further encodings.

The basic use is:

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

The LuaTeX and XeTeX engines use Unicode for their internal representation, so this command is no longer necessary. Furthermore, it might cause missing/incorrect characters or bad hyphenation, so it should be removed altogether.

Good practices
Unfortunately, most journals (and coauthors) still use pdfLaTeX in their production process, so you might just need to accept the minor issues with newer compilers.

3.2 Units of measure and whitespace

TEX supports specifying distances in a variety of units. The most useful are:

`cm` Centimetres

`in` Inches (1 in=2.54 cm)

`pt` Points (1 in=72.27 pt); usually used for font sizes[4]

`ex` Approximate height of 'x' in current font

`em` Approximate width of 'M' in current font

`mu` Math unit (1 em=18 mu); used for spacing in math mode

[3] To be precise, several characters can be combined into one glyph: see the ffi ligature in 'efficient'.

[4] There is also the `bp` (big point) used by PDF format, where there are exactly 72 bp to an inch.

Wherever a length is used, it must be supplied with a unit: for example, `1.5cm`.

In horizontal mode, a space is created with the `\hspace` command. The `\quad` command produces an *em space* (1 em), `\qquad` doubly so, and `\enspace` half of it. In math mode, there are some more commands: see Section 4.3.1.

```
An em space is this big:\\
xxx\hspace{1em}xxx
```

An em space is this big:
xxx xxx

It is possible to make a length *stretchy*, letting it expand or shrink within some limits. Here the syntax `1cm plus 1cm minus 0.6cm` means that the space can be anything between 0.4 cm and 2.0 cm:

```
\newcommand\spa
  {\hspace{1cm plus 1cm minus 0.6cm}}

Every\spa word\spa space\spa in\spa this\spa
long\spa example\spa paragraph\spa has\spa
this\spa weird\spa stretchy\spa space.
Perhaps\spa anybody\spa should\spa not\spa
set\spa any\spa text\spa like\spa it.
```

Every word space in this long example paragraph has this weird stretchy space. Perhaps anybody should not set any text like it.

An extreme case of stretchy spaces is given by `\hfill`: it expands to take all the available space on the line.

```
Left\hfill center\hfill right
```

Left center right

Internally, `\hfill` is implemented as `\hspace{\stretch{1}}`. With the `\stretch` command it is possible to distribute the space in uneven ratio:

```
Left\hspace{\stretch{1}} off-center\hspace{\stretch{2}} right
```

Left off-center right

There are also the `\hrulefill` and `\dotfill` commands that fill the space with something:

```
Left\hrulefill center\dotfill right
```

Left_____center . . . . . right

The same general ideas work for vertical spacing. One can request exact vertical space with `\vspace` or a stretch space with `\vfill` (below, the latter does nothing since the "page" streches according to the example content):

```
Something

\vspace{1cm}

Something else

\vfill

Bottom of the page
```

Something

Something else
Bottom of the page

Vertical space at page boundaries is eliminated. If you need to suppress that, for instance when designing a title page, you can use `\vspace*` (see the example on page 74).

Instead of manual spacing commands within text, you should use the semantically more meaningful `\smallskip`, `\medskip`, and `\bigskip` commands.

However, there is a significant catch. Remember that TEX alternates between the horizontal and vertical mode. Using vertical commands in the horizontal mode might cause some surprises.

```
An example.
\bigskip
Inside this paragraph,
there is a vertical spacing command.
Can you see where it got applied?
```

An example. Inside this paragraph, there is a vertical spacing command. Can you see where it got applied?

3.2.1 Length variables

Like counters, TEX supports special length variables. TEX and LATEX supply quite a lot of read-only variables that can be used in length expressions.

Let us use `\textwidth` as our example. It gives the width of the text area on the page. We can print its value with the `\the` command (mostly useful for debugging!), and put a multiplicative factor in front of it:

```
The text width is \the\textwidth.
Here's a line of that width:

{\color{blue}\rule{\textwidth}{1pt}}

And of half that width: {\color{blue}\rule{0.5\textwidth}{1pt}}
```

The text width is 360.0pt. Here's a line of that width:

And of half that width:

Importantly, this length changes depending on the context. Inside a `minipage` environment, the width of the text area is different. Let us do the same example again, but now the output is placed inside a smaller minipage:

```
The text width is \the\textwidth.
Here's a line of that width:

{\color{blue}\rule{\textwidth}{1pt}}

And of half that width:
{\color{blue}\rule{0.5\textwidth}{1pt}}
```

The text width is 143.9978pt. Here's a line of that width:

And of half that width:

Out of the box, the only calculation possible with lengths is multiplication, like `0.5\textwidth` used above. The calc package has traditionally been used when a complete expression language is needed.

New in LaTeX3
LaTeX3 now also provides similar functionality through the `\dimeval` and `\inteval` commands:

```
Text height: \the\textheight\\
Paper height: \the\paperheight\\
Padding:
\dimeval{\paperheight - \textheight}\\
About this many lines:
\inteval{\textheight / \baselineskip}
```

Text height: 595.80026pt
Paper height: 845.04684pt
Padding: 249.24658pt
About this many lines: 44

Be warned that the syntax does have some peculiar corner cases.

It is usually enough to provide length expressions to commands, but sometimes it is useful to define custom length variables. Such a variable is initialized with `\newlength`, similar to how counters are created. It can then be set with `\setlength`, and used by prefixing the name with `\`:

```
\newlength\Mylen
\setlength\Mylen{1.75em}

My\hspace{\Mylen}space.
```

My space.

Technical side note
Like counters, length variables live in the global scope. Once created, they never go away.

There are the useful `\settowidth`, `\settoheight`, and `\settodepth` commands that measure a piece of text and store its size in the variable:

```
\settowidth\Mylen{Hello world!}

The text ``Hello world!''
is {\the\Mylen} wide.

\settowidth\Mylen{\large Hello world!}

In \verb|\large| font it takes
the whopping \the\Mylen.
```

The text “Hello world!” is 57.82695pt wide.
In `\large` font it takes the whopping 62.01604pt.

3.3 Page geometry

Let us then move on to the layout of a complete page. To see the frame of a page, you can load the showframe package or pass the `showframe` option to the geometry package. The result is shown in Figure 3.1.

The most basic customization tools are the options passed to the document class. For further changes to the geometry, or a larger range of paper sizes, it is easiest to turn to the geometry package.

Just loading the geometry package changes the layout: by default, the package applies narrower margins than LaTeX does by default. Further customization is done by optional arguments to the package, or later in the preamble with the `\geometry` command.

You only need to specify the options you specifically want to customize – the package computes the remaining parameters.[5] For all the details, look at the package documentation. Some of the more useful options are:

`inner` The width of the inner margin. In `oneside` document mode, this corresponds to the left margin.

`outer` The width of the outer margin. In `oneside` document mode, this corresponds to the right margin.

`textwidth` The width of the text area.

`textheight` The height of the text area.

`lines` Alternatively to the above, the desired text height in lines (in the current default font).

`headheight` The height of the header area.

`headsep` Distance between the header and text area.

`footskip` Distance between the text area and the footer.

`paper` This accepts an extended range of paper sizes: `a0paper` through `a6paper`, and the same for B and C series. Custom sizes can be specified through `paperwidth` and `paperheight`.

Remark
University of Helsinki doctoral theses are printed on B5 paper. The suggested font size is 10 pt in that case.

[5] Of course, it is possible to give an incompatible set of measurements.

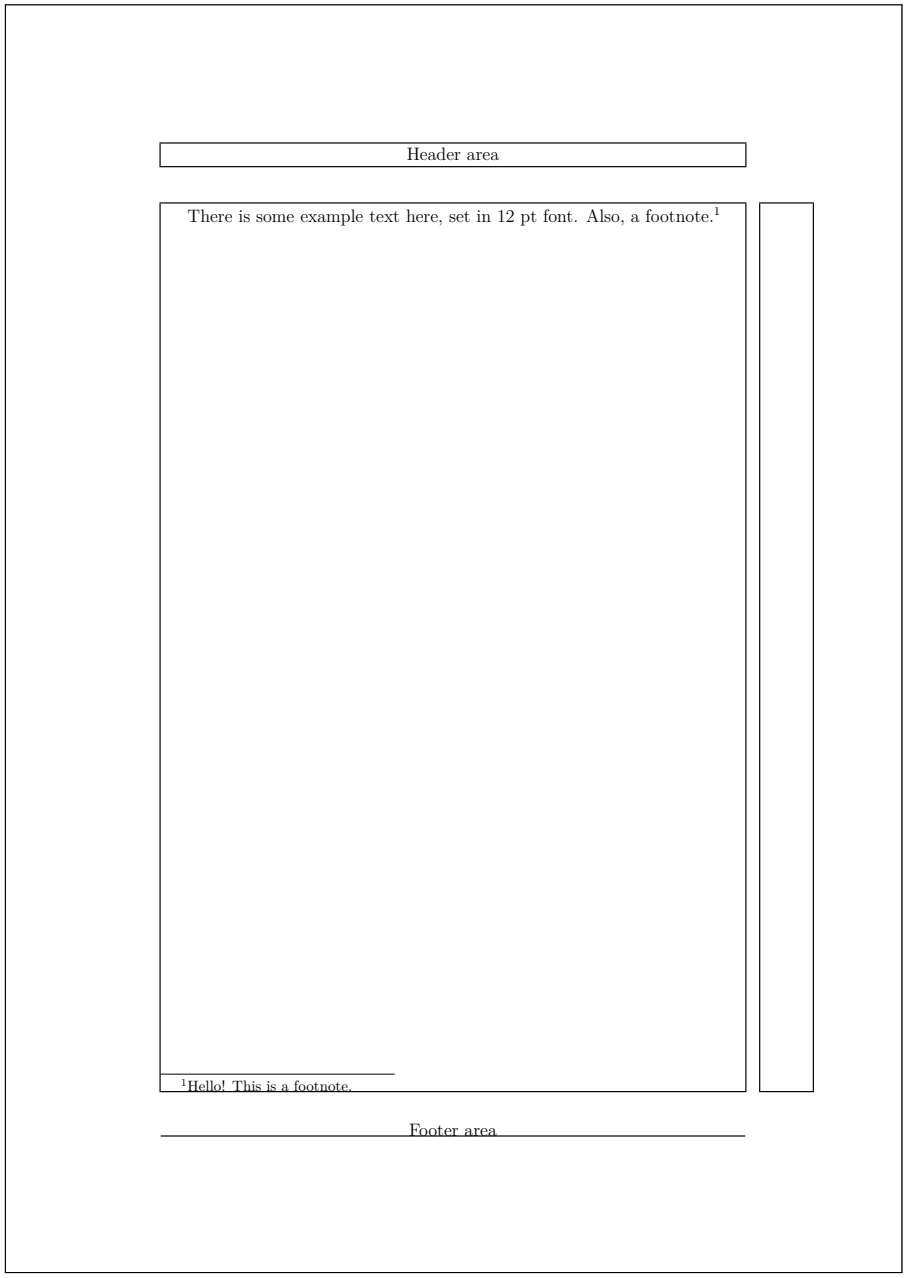


Figure 3.1: The geometry of an A4 page. The header is set to be 15 pt tall.

Warning
The raw TEX and LATEX lengths are available as well, and many a Stack Exchange answer suggests modifying them directly. There is nothing wrong with it, but geometry gives a nicer interface that avoids some surprising things.

Technical side note
As an example of a surprise, the coordinate system of TEX puts the origin at 1 inch to the right and below of the top-left corner of the paper. That means that some of the internal length parameters are negative.

Of the read-only length variables provided by LATEX, `\textwidth` and `\textheight` are some of the most useful ones. It is very common to set a figure to be a certain portion of the text width.

3.3.1 Landscape pages

LATEX has several restrictions to changing the layout mid-document. Some parameters can be customized with the `\newgeometry` command, which ends the current page, outputs all the outstanding floats, and starts a new page with the new layout.

Sometimes, it is useful to produce single landscape pages, e.g. to hold a wide table. For this, you can use the pdflscape package and its `landscape` environment. The contents of this environment are set on a new landscape page (or several). See also Section 7.2.3 for automatic placement of floats on landscape pages.

Technical side note
The pdf in pdflscape stands for the fact that the environment even tells the PDF reader application to rotate the page on your screen. This is its sole improvement over the older lscape package.

3.3.2 Paragraphs and page breaks

Other two commonly customized parameters are `\parindent` and `\parskip`. As the names suggest, they control the indentation and distance between paragraphs.

By default, paragraphs are indented, with no extra vertical separation. If you prefer non-indented, vertically separated paragraphs, the commands are:

```
\setlength\parindent{0pt}
\setlength\parskip{4pt}

Some text forming an example paragraph.
Just enough to get it to a few lines.

Another example paragraph,
with quite a few words.
```

Some text forming an example paragraph. Just enough to get it to a few lines.

Another example paragraph, with quite a few words.

If you want to remove the indentation of a single paragraph only, you can put `\noindent` at the beginning of the paragraph.

Let us then move to page breaks. The `\clearpage` command ends the current page, then produces enough pages to set all the outstanding floats (figures and tables), and finally starts a new page. The `\cleardoublepage` variant ensures that the new page has odd number, which puts it on the right-hand side of a twoside layout. This is done automatically at the end of chapter in the report document class.

Generally, you should not otherwise interfere with the page breaking algorithm, but sometimes it is necessary for a good layout. The `\pagebreak` command forces a manual page break.

However, this command *is dangerous*. If the preceding text changes, it is possible that the `\pagebreak` command now produces a mostly-empty page!

Instead, it might be better to use the needspace package. The identically named command approximates the remaining space on the page, and starts a new page if there is not enough. For example, to ensure that there is space for two more lines, one can say

```
\needspace{2\baselineskip}
```

Finally, there is an extreme command for fine-tuning the final layout of a page. The addlines package can be used to add or remove lines to the current spread. (The package applies the extension to both sides of a spread in twoside mode; in oneside mode it only affects the single page. This distinguishes the package from raw LaTeX `\enlargethispage` command.)

Warning
Only do fine-tuning to the page layout at the very end of editing process – even the tiniest change can cause a massive cascade of layout changes!

Quite often, you can do completely without these tweaks.

3.4 Headers and footers

3.4.1 Built-in LaTeX styles

By default, LaTeX provides three page styles:

`empty` Nothing in the header or footer.

`plain` Centered page number in the footer.

`headings` Header contains current chapter/section and subsection, and the page number.

For articles and reports, `plain` is the default; for books, `headings`. The choice is taken with `\pagestyle` in the preamble. It is also possible to override the style temporarily with `\thispagestyle`. (The `titlepage` environment described below does this automatically.)

We do not discuss here the base LaTeX mechanisms for styling headers, since the fancyhdr package described below does a much better job at it. However, it is worth understanding the chapter and section names appearing in the `headings` style.

There are two marks: left mark and right mark. What they should be thought of is *main mark* and *sub-mark*, but unfortunately here the semantics and presentation are not separated. The `\markboth` command sets both marks, and `\markright` only the sub-mark. Essentially, the sectioning commands do something like

```
\chapter{Chapter name} -> \markboth{Chapter name}{}
\section{Section name} -> \markright{Section name}
```

The right-hand-sides are internally implemented with the `\chaptermark` and `\sectionmark` commands,[6] and by renewing them it is possible to change the format of marks. For example, to remove the automatic upper-casing one can use

```
\renewcommand{\chaptermark}[1]{\markboth{\chaptername~\thechapter. #1}{}}
\renewcommand{\sectionmark}[1]{\markright{\thesection. #1}}
```

To access the marks, you can use `\leftmark` and `\rightmark`.

New in LaTeX3
LaTeX3 introduces a completely new marking mechanism. We do not discuss it here, but its brief documentation does give the essential details.[a]

[a] `www.latex-project.org/help/documentation/ltmarks-doc.pdf`

[6] These commands exist for all sectioning levels, not just chapters and sections.

3.4.2 Page numbering

To customize just the page numbering, you can renew `\thepage`. For example, `\renewcommand{\thepage}{\roman{page}}` would print page numbers in Roman numerals.

Gotcha!
There is also a `\pagenumbering` command. It takes a formatting style as its sole argument, for example `Roman` for uppercase Roman numerals or `arabic` for the usual numbers. You can also pass `gobble` to suppress page numbering.

But here's the catch: the command also resets the page numbering! If you are use this command after the first page of a two-side layout, this might cause confusion.

If you want to write something like "Page m of n", where n is the total number of pages, the easiest way is to load the lastpage package. It creates a label called `LastPage` on the last page of the document, and this label can then be referred to as usual:

```
This is page
\thepage{} of \pageref*{LastPage}.
```

This is page 74 of 205.

(Here we used the starred `\pageref*` command that does not produce a clickable link. It is of course fine, if a bit odd, to provide a link to the last page.)

Do note that this method gives the *page number* of the last page, not the *total number* of pages. The two might not be equal if several page numbering schemes have been in use.

3.4.3 Customized title pages

If the output of `\maketitle` is not sufficient for your artistic tastes, you can of course set the title page by yourself. The `titlepage` environment creates an empty page with no decorations.[7] The title page of these notes was created with the following code:

[7] It also sets the page number to 1, which is usually correct. If you intend the title page to be on left-hand side in twoside mode, then this gets you in trouble.

```
\begin{titlepage}
\vspace*{\stretch{1}}

\hrulefill

\begin{flushright}
\textbf{\LARGE A second course in \LaTeX}\\[1em]
\Large Petri Laarne\\[1em]
Draft version\\
\today
\end{flushright}

\hrulefill

\vspace*{\stretch{2}}
\end{titlepage}
```

3.4.4 Getting fancy with fancyhdr

The fancyhdr package provides a nice interface for customizing the headers and footers. It is taken into use by loading the package and setting the `fancy` page style. Its model of the page consists of six positions for text:

Left header	Centre header	Right header

[Page content]

Left footer	Centre footer	Right footer

The `\fancyhf` command is the basic workhorse of the package: it takes an optional placement specifier and the header content. The placement consists of up to three letters: `E` or `O` for even and odd pages, one of `LCR` for left-centre-right, and `H` or `F` for header and footer. The `\fancyhead` and `\fancyfoot` commands imply the header/footer specifier.

These are maybe best illustrated by an example:

```
\pagestyle{fancy}

% No specifier, so the command applies to all positions
% This clears any pre-existing content
\fancyhf[]{}

% Page number to the centre footer of all pages
\fancyfoot[C]{\thepage}

% Some text to the margin-side header
% (Left on even pages, right on odd pages)
\fancyhead[LE, RO]{\textbf{An important document}}
```

To access the current chapter and section name, you can use the marks described in Section 3.4.1. The following code replicates the default LaTeX `headings` page style:

```
\pagestyle{fancy}

% Clear all
\fancyhf[]{}

% Page number to outer edge
\fancyhead[LE,RO]{\thepage}

% Mark to inner edge
\fancyhead[RE]{\leftmark}
\fancyhead[LO]{\rightmark}
```

Remark
The package gives a warning if the header height is not large enough. The warning even gives a suggested value for the `\headheight` length, which you can pass to geometry (or do a raw `\setlength`, if not using the package).

You can customize the rule separating header and text by redefining the `\headrulewidth` length, and correspondinly `\footrulewidth`. It is also possible to customize the separation from header text and the line style, but for these we refer to the package documentation.[8]

[8] One has to be a bit careful with vertical spacing.

Chapter 4

Mathematics layout

Plain TEX already provides a lot of facilities for typesetting mathematics, and this is further extended by LATEX and the amsmath packages. The amsmath package is essentially as old as LATEX. It is automatically loaded by the Americal Mathematical Society document classes like `amsart`, so you might not even need to load it.

The package works well but has not been significantly updated since the nineties. Some additions and bug fixes are collected in the mathtools package. If you use a non-AMS document class, you can load mathtools instead of amsmath to get slightly improved typesetting.

Technical side note
The amsmath package used to be tied to the AMS document classes, but it is nowadays maintained by the LATEX core team. It is one of the packages guaranteed to be present on every LATEX distribution.

4.1 Equation environments

Standard LATEX provides the `\[ ... \]` syntax for creating a mathematics display (as opposed to inline mathematics with `$ ... $`). This environment does not support equation numbering or line breaks; for those, amsmath provides a lot of options.

Warning
Plain TEX used the `$$ ... $$` syntax for display mathematics. Do not use it – the LATEX-style environment has some hooks and accessibility features that the TEX syntax does not have.

4.1.1 Single numbered equation

The basic equation environment of amsmath is `equation`. It sets its contents on a single line and numbers the equation:

```
\begin{equation}
i \partial_t u - \Delta u = -u^3.
\end{equation}
```

$$i\partial_t u - \Delta u = -u^3. \quad (4.1)$$

If you'd rather have an unnumbered equation, you can use the `equation*` environment. It is essentially equivalent to `\[ \]` of LaTeX.

To customize the equation number, you can use the `\tag` command.

```
\begin{equation}
i \partial_t u - \Delta u = -u^3.
\tag{NLS}
\end{equation}
```

$$i\partial_t u - \Delta u = -u^3. \quad \text{(NLS)}$$

The tag also appears in cross-references to this equation. The tag is read in text mode, so any mathematical symbols need to be wrapped with `$`. For example, you would write `\tag{$\ast$}` to produce $(*)$.

If you want to change whether equations are numbered globally or per section, use the `\counterwithin` mechanism described in Section 1.6:[1]

```
\counterwithin{equation}{section}
```

4.1.2 Single equation on many lines

You can break a long expression into multiple lines with the `multline` environment. Only one equation number is produced (none if you use `multline*`). Line breaks are specified with `\\`:

```
\begin{multline}
a^5 + 5 a^4 b \\
+ 10 a^3 b^2 + 10 a^2 b^3\\
+ 5 a b^4 + b^5
\end{multline}
```

$$\begin{aligned} a^5 + 5a^4 b \\ + 10a^3 b^2 + 10a^2 b^3 \\ + 5ab^4 + b^5 \end{aligned} \quad (4.2)$$

The first line is left-aligned, the last line is right-aligned, and the rest are centered.

[1] The amsmath package also defines `\numberwithin`, which is functionally identical to `\counterwithin`. It does not matter which one you use.

If you need to control the alignment, you can use the `split` environment. This environment needs to be put inside `equation` (or `equation*`). The alignment point is denoted by `&`:

```
\begin{equation}
\begin{split}
(a+b)^2
&= (a+b)(a+b)\\
&= a^2 + 2ab + b^2.
\end{split}
\end{equation}
```

$$\begin{aligned}(a+b)^2 &= (a+b)(a+b)\\ &= a^2 + 2ab + b^2.\end{aligned} \tag{4.3}$$

Sometimes, you need to align with things that are not really there. Let us produce a slightly different version of the previous example:

```
\begin{equation}
\begin{split}
&\mathrel{\phantom{=}} (a+b)^2\\
&= (a+b)(a+b)\\
&= a^2 + 2ab + b^2.
\end{split}
\end{equation}
```

$$\begin{aligned}&(a+b)^2\\ &= (a+b)(a+b)\\ &= a^2 + 2ab + b^2.\end{aligned} \tag{4.4}$$

The `\phantom` command reserves enough space on the first line for =, even though it is not printed there; the `\mathrel` command ensures that the spacing is that surrounding = as well. This ensures that the expressions are aligned. The same effect can not be attained with moving the alignment symbol, since the TEX spacing algorithm (described below) cannot see across the `&`:

```
Bad spacing after $=$ sign:
\begin{equation}
\begin{split}
&(a+b)^2\\
= &(a+b)(a+b)\\
= &a^2 + 2ab + b^2.
\end{split}
\end{equation}
```

Bad spacing after = sign:

$$\begin{aligned}&(a+b)^2\\ =&(a+b)(a+b)\\ =&a^2 + 2ab + b^2.\end{aligned} \tag{4.5}$$

4.1.3 Many equations, many lines

The `gather` environment collects equations, each centered.

```
\begin{gather}
(a+b)^2 = a^2 + 2ab + b^2,\\
a^2 - b^2 = (a+b)(a-b).
\end{gather}
```

$$(a+b)^2 = a^2 + 2ab + b^2, \tag{4.6}$$
$$a^2 - b^2 = (a+b)(a-b). \tag{4.7}$$

This environment also supports `split` on one or more sub-equations:[2]

```
\begin{gather}
(a+b)^2 = a^2 + 2ab + b^2,\\
\begin{split}
(a+b)^3 &= a^3\\
&\mathord{\phantom{=}} + 3a^2 b + 3ab^2 + b^3.
\end{split}
\end{gather}
```

$$(a+b)^2 = a^2 + 2ab + b^2, \tag{4.8}$$
$$\begin{aligned}(a+b)^3 &= a^3 \\ &\quad + 3a^2b + 3ab^2 + b^3.\end{aligned} \tag{4.9}$$

You can customize the tag of each individual equation with `\tag`, and also suppress an individual tag with `\notag`. These commands can appear anywhere before the `\\` that ends the particular equation. Again, the `gather*` environment suppresses all tags except those explicitly created with `\tag`.

```
\begin{gather}
(a+b)^2 = a^2 + 2ab + b^2,\\
(a+b)^3 = a^3 + 3a^2 b + 3a b^2 + b^3, \notag\\
a^2 - b^2 = (a+b)(a-b). \tag{$\ast$}
\end{gather}
```

[2]Now we wrap the phantom equals sign in `\mathord`, which makes the + sign act as binary plus, not unary plus.

$$(a+b)^2 = a^2 + 2ab + b^2, \tag{4.10}$$
$$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3,$$
$$a^2 - b^2 = (a+b)(a-b). \tag{$*$}$$

If you need to have several aligned equations, you can use the `align` environment. In comparison to `split`, each line is now numbered individually:

```
\begin{align}
(a+b)^2
&= (a+b)(a+b)\\
&= a^2 + 2ab + b^2.
\end{align}
```

$$(a+b)^2 = (a+b)(a+b) \tag{4.11}$$
$$= a^2 + 2ab + b^2. \tag{4.12}$$

The same remarks about `\tag` and `\notag` with `gather` apply here. The `align` environment also supports more than one alignment point.

Warning
There is also an `eqnarray` environment provided by base LaTeX. It is far less sophisticated and generally uglier than anything you can achieve with the environments presented here, so I would avoid it.

If you want to number the equations as subequations, it can be done by wrapping the environment inside `subequations`:

```
\begin{subequations}
  \begin{align}
    (a+b)^2
    &= (a+b)(a+b)\\
    &= a^2 + 2ab + b^2.
  \end{align}
\end{subequations}
```

$$(a+b)^2 = (a+b)(a+b) \tag{4.13a}$$
$$= a^2 + 2ab + b^2. \tag{4.13b}$$

If your equation environment is very long, it might be beneficial to allow page breaks. The display environments presented here do not allow page breaks by default. By putting a `\displaybreak` command before the `\\`, you indicate that a page break *is allowed*.

The `\allowdisplaybreaks` command permits page breaks in displays everywhere in its scope. If you put it inside an environment, it applies to that environment only; if you put it in the preamble, it applies to all environments.

4.1.4 Cases

To group several expressions with vertical brackets, you can use the `cases` environment inside an equation environment. This environment supports a single alignment `&`:

```
\begin{equation}
f(n) = \begin{cases}
    n^2, & n > 0,\\
    -n, & n \leq 0.
\end{cases}
\end{equation}
```

$$f(n) = \begin{cases} n^2, & n > 0, \\ -n, & n \leq 0. \end{cases} \tag{4.14}$$

If you mostly have text following the alignment character, you can use the starred environment. It interprets the right-hand side of `&` in text mode:

```
\begin{equation}
f(n) = \begin{cases*}
    n^2+1, & if $n$ is even,\\
    n+7, & if $n$ is odd.
\end{cases*}
\end{equation}
```

$$f(n) = \begin{cases} n^2+1, & \text{if } n \text{ is even}, \\ n+7, & \text{if } n \text{ is odd}. \end{cases} \tag{4.15}$$

They also have a right-aligned cousin `rcases`:

```
\begin{equation}
\begin{rcases}
i \partial_t u - \Delta u = -u^3,\\
\partial_{tt} u - \Delta u = -u^3
\end{rcases}
\text{ some dispersive PDE}
\end{equation}
```

$$\left.\begin{aligned} i\partial_t u - \Delta u &= -u^3, \\ \partial_{tt} u - \Delta u &= -u^3 \end{aligned}\right\} \text{ some dispersive PDE} \tag{4.16}$$

However, do note that there is some space put before the `&` alignment position. If you want to collect several aligned equations inside a brace, it might be nicer to fake the output with help from the `aligned` environment and a suitably sized `{` character:

```
Not nice:
\begin{equation}
\begin{cases}
    \partial_{tt} u - \Delta u &= 0,\\
    u(0) &= X,\\
    \partial_t u(0) &= Y.
\end{cases}
\end{equation}

Much nicer:
\begin{equation}
\left\{
\begin{aligned}
    \partial_{tt} u - \Delta u &= 0,\\
    u(0) &= X,\\
    \partial_t u(0) &= Y.
\end{aligned}
\right.
\end{equation}
```

Not nice:

$$\begin{cases} \partial_{tt} u - \Delta u &= 0, \\ u(0) &= X, \\ \partial_t u(0) &= Y. \end{cases} \tag{4.17}$$

Much nicer:

$$\left\{ \begin{aligned} \partial_{tt} u - \Delta u &= 0, \\ u(0) &= X, \\ \partial_t u(0) &= Y. \end{aligned} \right. \tag{4.18}$$

Here the `\left`–`\right` pair is used to produce a correctly sized brace.

If you want to number lines in the `cases` environment individually, check out the cases package.

4.1.5 Matrices

Like `cases`, matrices are not equation environments on their own, but can appear inside one. There are a few variants depending on how you like them:

```
\begin{gather*}
\begin{pmatrix}
    1 & 2\\
    3 & 4
\end{pmatrix}\\
\begin{bmatrix}
    1 & \cdots & 1\\
    0 & \ddots & \vdots\\
      & \cdots & 1
\end{bmatrix}\\
\begin{vmatrix}
    a & b\\
    c & d
\end{vmatrix}
\end{gather*}
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\begin{bmatrix} 1 & \cdots & 1 \\ 0 & \ddots & \vdots \\ & \cdots & 1 \end{bmatrix}$$

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

4.2 Fonts and text in mathematics

In mathematical mode TEX follows different spacing rules. Spaces are ignored, and the spacing between letters is slightly altered:

```
\emph{different spacing}\\
$different spacing$
```

different spacing
$differentspacing$

To make your documents have that subtle final touch, it is therefore important to understand where the boundary of text and math goes. We will first discuss putting *explanatory text* within math environments, and then talk about *mathematical text* styles like $\exp(f_{\text{stat}})$ below.

Good practices
Also the spacing of punctuation is slightly different in math mode. You should use math mode only for mathematics; in particular, the second example below is preferable:

```
Summing over $i, j, k$, we find\dots\\
Summing over $i$, $j$, $k$, we find\dots\\
```

Summing over i,j,k, we find…
Summing over i, j, k, we find…

4.2.1 Explanatory text

To put explanatory text inside a math display, use the `\text` command of amsmath. Since spaces are ignored in math mode, you need to put the spacing inside the argument. (Quite often, it is useful to add a bit more space with the manual spacing commands described below.)

```
\[
a_k < a_{k+1}
\text{ for all } k \in \mathbb N.
\]
```

$$a_k < a_{k+1} \text{ for all } k \in \mathbb{N}.$$

The argument inside `\text` is interpreted in text mode, but it is in fact possible to enter math mode again with `$`:

```
\[
a_k < a_{k+1}
\text{ for all $k \in \mathbb N$, and }
a_1 > 1.
\]
```

$$a_k < a_{k+1} \text{ for all } k \in \mathbb{N}, \text{ and } a_1 > 1.$$

Inside an `align` environment, you can do short textual interjections with `\intertext`. The benefit is that the alignment is kept in sync across mathematics lines:

```
\begin{align}
a^3 - b^3
&= \sum_{k=0}^3 \binom 3 k a^k b^{3-k},\\
\intertext{and by some computation this equals}
&= (a-b)(a^2 + ab + b^2).
\end{align}
```

$$\begin{aligned} a^3 - b^3 &= \sum_{k=0}^{3} \binom{3}{k} a^k b^{3-k}, \\ \intertext{and by some computation this equals} &= (a-b)(a^2 + ab + b^2). \end{aligned} \tag{4.19}$$

Good practices
As useful as `\intertext` is, remember that the display math environments do not participate in page breaking by default. If you have a long chain of equalities, you should think about the most reader-friendly way to present it.

4.2.2 Mathematics fonts

Then what about mathematics set in a diffenent font? First, we need to distinguish *operators* and *ordinary symbols* from each other, since the two also differ in spacing. Just compare the two examples:

```
$\sin x$\\
$\mathrm{sin} x$
```

$\sin x$
$\mathrm{sin} x$

The spacing rules around operators are discussed in Section 4.3.2. The `\operatorname` command described there also takes care of the upright font.

This leaves us finally to talk things like f_{stat}, $\mathbf{R}$ or $\mathbb{R}$, and $\mathcal{F}$. The commands are similar to those used in text mode.

The upright roman style is produced with `\mathrm`, bold with `\mathbf`, and the calligraphic style with `\mathcal`. There is also the sans serif style `\mathsf`.

```
$\mathrm{F(x)}$\\
$\mathbf{F(x)}$\\
$\mathsf{F(x)}$\\
$\mathcal{F(x)}$
```

$\mathrm{F(x)}$
$\mathbf{F(x)}$
$\mathsf{F(x)}$
$\mathcal{F(x)}$

As you see from above, not all characters are supported by the default fonts, sometimes producing confusing results.

Good practices
A common convention is to use upright style when something has a multi-letter name. That is, the identity matrix should be Id and not Id, and a stationary function should be f_{stat} and not f_{stat}.

Note that symbols are not typically set in bold font even when `\mathbf` is used:

```
\[
\mathbf{\sum_{k=1}^\infty \frac{k}{1+k^2}}
\]
```

$$\mathbf{\sum_{k=1}^\infty \frac{k}{1+k^2}}$$

There is the `\boldsymbol` command in amsmath, but it only supports a subset of symbols. If you need bold mathematics, the easiest way is to use the bm package and the identically named command:

```
% \usepackage{bm}
\[
\bm{\sum_{k=1}^\infty \frac{k}{1+k^2}}
\]
```

$$\sum_{k=1}^{\infty} \frac{k}{1+k^2}$$

Whenever the font has a special bold symbol, it is used; otherwise, a "poor man's bold" is applied by overprinting the character with small offsets.

By loading the amsfonts package, you gain a couple more styles: "blackboard bold" by `\mathbb` and fraktur by `\mathfrak`. Again, these support only a subset of the characters.

```
% \usepackage{amsfonts}
$\mathbb{ABC}$\\
$\mathfrak{ABCdef}$
```

$\mathbb{ABC}$
$\mathfrak{ABCdef}$

In some fields, a blackboard bold 1 is used (to signify an indicator function, for example). This symbol is not provided by amsfonts. There are two methods to fix this issue.

First, $\mathbb{1}$ can be found e.g. in the dsfont package that provides a double stroke font accessible via `\mathds`.

```
% \usepackage{amsfonts, dsfont}
$\mathbb{ABC123}$\\
$\mathds{ABC123}$
```

$\mathbb{ABC123}$
$\mathds{ABC123}$

Note that the other digits are now missing altogether!

Since `\mathbb` is in muscle memory for many of us, let us show one dirty trick. If you want to replace the blackboard bold font altogether with the double-stroke font, you can use the following code:

```
% \usepackage{dsfont}
\DeclareCommandCopy{\mathbb}{\mathds}

$\mathbb{ABC123}$
```

$\mathbb{ABC123}$

The second method is more correct, but it requires a modern LaTeX processing pipeline. It should thus be used still with caution.

New in LaTeX3
If you copy mathematical text from a LaTeX-produced PDF, the symbols come in a normal font or might be completely random. This is another example of TeX and Unicode having evolved separately. However, Unicode nowadays has code points for symbols like "blackboard bold R".

If you use a Unicode-native compiler (LuaLaTeX or XeLaTeX), you can load the unicode-math package. This package replaces all math symbols with their Unicode equivalents. Now they appear correctly for copy-paste and screen readers.

With this package, you no longer need to load amsfonts, *all digits* have blackboard bold variants, and there are no longer weird symbol substitutions. Moreover, it is possible to easily load a different Unicode font for maths.

Where's the catch? The package is still marked as experimental, it does not work with pdfLaTeX, and there are some subtle differences compared to original math commands. This package should only be used for new projects where backwards compatibility is not required.

4.2.3 Completely different typefaces

It is possible to use non-default fonts for mathematics, but there are many caveats. Most fonts do not contain a usable range of mathematical symbols, and those that do might not pair well with the text font.

The unicode-math package and Unicode fonts with good mathematics support are the best bet. Palatino and Noto families also have good mathematics support, which we will discuss in Section 6.3.

4.2.4 Units of measure

The siunitx package simplifies working with numerical quantities and units. There are a few basic commands that cover most of the cases.

Numbers are formatted with `\num`. This command accepts exponential notation and error ranges, and even both `.` and `,` as the decimal separator. Complex numbers are given with the `\complexnum` command.

```
\num{12345}\\
\num{3,9 +- 0,4}\\
\num{1.23(0.45:0.56)}\\
\num{1.28e-4}\\
\complexnum{1+2ie3}
```

$12\,345$
$3.9(4)$
$1.23^{+0.45}_{-0.56}$
1.28×10^{-4}
$(1 + 2\mathrm{i}) \times 10^{3}$

The next basic command is `\unit`, which offers two modes. In the "literal" mode, units are given as strings. Multiplication is done with `.`,

division by /, and sub- and superscripts in the usual way. In the "macro" mode, each part is given as a command (whose name corresponds to the usual spoken version). The benefit of this mode is that the appearance can be customized.

```
\unit{kg.m.s^{-2}}\\
\unit{\kilogram\metre\per\second\squared}\\
\unit{T_{eff}}
```

$\mathrm{kg\,m\,s^{-2}}$
$\mathrm{kg\,m\,s^{-2}}$
$\mathrm{T_{eff}}$

Finally, the two commands can be put together with \qty (or its sibling \complexqty):

```
\qty{1.7e3}{kg.m/s^2}\\
\qty{17}{\degreeCelsius}\\
\complexqty{1+2ie-3}{\ohm}
```

$1.7 \times 10^{3}\,\mathrm{kg\,m/s^{2}}$
$17\,^{\circ}\mathrm{C}$
$(1 + 2\mathrm{i}) \times 10^{-3}\,\Omega$

For angles, there is a special syntax with \ang:

```
\ang{60}\\
\ang{12;34;56}
```

60°
$12^{\circ}34'56''$

The package offers a lot of customization in how the quantities are displayed. (This aligns well with the general LaTeX philosophy.) See the package documentation for a thorough explanation.

Remark
See also page 147 for using siunitx in tables.

4.3 Mathematical symbols and whitespace

LaTeX and mathtools already provide quite a lot of symbols, but if those are not enough, there are many extension packages. The first two to consider are amssymb (AMS symbol font) and stmaryrd. There are also some specialized packages like braket for Dirac bra-ket notation.

Gotcha!
The stmaryrd package should be loaded *after* amssymb, since it extends and modifies some symbols there.

If you're not interested in browsing through package documentation to find symbols, the Detexify tool[3] is your friend. You can draw the symbol in this web app, and it searches for it in LaTeX symbol list and a large collection of extension packages.[4]

4.3.1 Spacing

TeX is quite smart about figuring out the spacing between different mathematical symbols. Just look at the difference between $2-1$ and $2(-1)$. Sometimes, it is important to choose the command properly to get expected spacing. Compare the two examples:

```
$f : A \to B$\\
$f \colon A \to B$
```

$f : A \to B$
$f \colon A \to B$

Let us first see the manual commands for spacing, since their sizes correspond to units used by TeX. Their sizes are defined based on the *math unit*, which depends on the font size. Note that the difference of the units is very small, just enough to be perceptible:

```
$xx$ (no space)\\
$x\,x$ (thin space, 3~mu)\\
$x\:x$ (medium space, 4~mu)\\
$x\;x$ (thick space, 5~mu)\\
$x\!x$ (negative thin space)\\
$x\mspace{8mu}x$ (custom space)
```

xx (no space)
$x\,x$ (thin space, 3 mu)
$x\:x$ (medium space, 4 mu)
$x\;x$ (thick space, 5 mu)
$x\!x$ (negative thin space)
$x\mspace{8mu}x$ (custom space)

These commands can be used to manually tweak spacings. However, even better is to let TeX automate things.

Internally, every mathematical symbol belongs to a symbol class. These are listed in Table 4.1. TeX puts a space between two symbols based on their respective classes. For example, there is no space between two ordinary characters, whereas there is a medium `\:` space between binary and ordinary symbols.

We talk more about operators below in Section 4.3.2, so let us consider binary operations as our example. Usually $\heartsuit$ is considered an ordinary symbol, but let us imagine that we have defined it as an operation between

[3] `detexify.kirelabs.org`

[4] `www.ctan.org/tex-archive/info/symbols/comprehensive` is the source if you really want to search manually; this PDF is over 30 MB and almost 500 pages.

Symbol class	Override command	Examples
Ordinary	`\mathord`	$2x$
Operator	`\mathop`	$\sin$
Binary	`\mathbin`	$2+x$
Relation	`\mathrel`	$2<x$
Opening	`\mathopen`	$($ $\lfloor$
Closing	`\mathclose`	$)$ $\rceil$
Punctuation	`\mathpunct`	,

Table 4.1: The mathematical symbol classes.

two expressions. Then it would be necessary to wrap its use in `\mathbin` in order to get correct spacing.

```
\newcommand{\friends}{\mathbin\heartsuit}

Bad: $x \heartsuit y$\\
Good: $x \friends y$.
```

Bad: $x\heartsuit y$
Good: $x \mathbin\heartsuit y$.

Some combinations of symbol classes are considered impossible, in which case TEX modifies one of the classes suitably. To go back to the example of the minus sign, `-` is usually defined as a Relation symbol. However, if it is not preceded by an Ordinary symbol (like a number), it is transformed into Ordinary itself – this means that `-1` produces no spacing at all.

Gotcha!
This somewhat explains the difference between `:` and `\colon`, but only partly. The symbol `:` is classified as Relation and `\colon` as Punctuation. However, amsmath further modifies the spacing of `\colon` to fit the $f\colon A \to B$ pattern – and only that pattern.

If you ever need the colon as a punctuation symbol, you can wrap it in `\mathpunct{:}`.

Gotcha!
Another source of confusion is `|` and `\mid`. The first is Ordinary and the second Relation. They should be used correspondingly: `|a|` for the absolute value $|a|$, `f|_A` for the restriction of a function $f|_A$, and `a \mid b` for the divisibility relation $a \mid b$.

Gotcha!
The symbols `. ! ?` are of class Ordinary, not Punctuation. This is to do with their mathematical meanings.

Gotcha!
If you write in a language where comma is used as the decimal separator, you should wrap the comma in parentheses:

```
$123,456$\\
$123{,}456$
```

$123,456$
$123{,}456$

Gotcha!
Adding an accent turns the symbol into Ordinary.

```
$a \hat= b$\\
$a \mathrel{\hat=} b$
```

$a \hat= b$
$a \mathrel{\hat=} b$

Gotcha!
The occasionally used notation for open intervals conflicts with the algorithm, since `]a, b[` is understood as Closing–Opening pair instead of the opposite. It can be fixed manually as in the next example:

```
$f \colon ]a, b[ \to ]0, 1]$\\
$f \colon \mathopen]a, b\mathclose[
    \to \mathopen]0, 1]$
```

$f \colon]a, b[\to]0, 1]$
$f \colon \mathopen]a, b\mathclose[\to \mathopen]0, 1]$

A quick fix is to wrap the intervals inside `{}`. If you use a lot of intervals, you should either define some custom commands or check out the interval package.

One common pain point is the humble dx that appears in integrals. It is nice to get the spacing consistently right, and note also the upright d (this is advocated by an ISO standard, but some prefer dx).

A common way to do this (based on Stack Exchange discussions) is:

```
\newcommand{\diff}{\mathop{}\!\mathrm{d}}

In an integral:
\[
\int_0^1 x \diff x,
\]
and inside parens: $\mu(\diff x)$.
```

In an integral:

$$\int_0^1 x \,\mathrm{d}x,$$

and inside parens: $\mu(\mathrm{d}x)$.

Note the automatic space between x and $\mathrm{d}x$.

What happens here is that the empty group `{}` is set in operator style. When preceded by an ordinary symbol, the operator style puts a thin `\,` space; when preceded by punctuation like `(`, the space is not present. The thin `\,` space between the (empty) operator and 'd' is cancelled with `\!`.

If you find yourself writing lots of complicated differentials like

$$\frac{\mathrm{d}^2 \log(x)}{\mathrm{d}x^2},$$

you should look into the diffcoeff package that provides a slightly shorter syntax.

4.3.2 Operators and limits

One common class of text in mathematics is operators. Compare the three examples here:

```
Awful: $2 sin \pi$,\\
Still bad: $2 \textrm{sin} \pi$,\\
Good: $2 \sin \pi$.
```

Awful: $2sin\pi$,
Still bad: $2\textrm{sin}\pi$,
Good: $2 \sin \pi$.

Here the `\sin` command not only sets the operator name properly, but also adjusts the spacing around the operator.

If the operator you need is not predefined as a LaTeX command, you can do the styling once with `\operatorname`…

```
\[
2 \operatorname{dim} X
\]
```

$$2 \operatorname{dim} X$$

…or define a new command with `\DeclareMathOperator` in the preamble:

```
% In the preamble:
% \DeclareMathOperator{\arccosh}{arccosh}

\[
2 \arccosh \pi
\]
```

$$2 \operatorname{arccosh} \pi$$

For certain operators, limits can be put either to the side or above/below the operator. Compare these two:

$$\text{text style } \sum_{i=1}^{\infty} \text{ and display style } \sum_{i=1}^{\infty}.$$

There are package options to amsmath/mathtools to control how the limits are placed in display equations. By default, they are placed on the side for integrals and above/below for everything else.

To put limits above/below your custom operator, use the starred form of the declaration:

```
\[
\operatorname{dim}_H^\ast X
\text{ vs }
\operatorname*{dim}_H^\ast X
\]
```

$$\dim_H^* X \text{ vs } \operatorname*{dim}_H^* X$$

If you need to put multiple lines of text in a sum argument, you can use the `\substack` command. Inside its argument, you can use the usual `\\` line breaks:

```
\[
\sum_{\substack{1 \leq j \leq 10\\ 1 \leq k \leq j}}
\]
```

$$\sum_{\substack{1 \leq j \leq 10\\ 1 \leq k \leq j}}$$

Some special operators that deserve a mention are the modulus operators, since they have some special spacing rules:

```
\begin{gather*}
a \equiv b \mod p\\
a \equiv b \bmod p\\
a \equiv b \pmod p
\end{gather*}
```

$$a \equiv b \mod p$$
$$a \equiv b \bmod p$$
$$a \equiv b \pmod p$$

4.3.3 Fractions

There are two basic fraction-like commands: `\frac` for fractions, and `\binom` for binomial coefficients. Both take the numerator and denominator as their arguments.

To produce generalized fraction-like operators like

$$\left(\frac{a}{b}\right) \quad \text{and} \quad \left\lfloor\frac{a}{b}\right\rceil,$$

the amsmath package provides the `\genfrac` command. Read the package documentation to understand its six parameters.

Continued fractions are produced with the `\cfrac` command:

```
\[
\cfrac{1}{\sqrt 3 + \cfrac{2}{\sqrt 3 + \dotsb}}
\]
```

$$\cfrac{1}{\sqrt{3}+\cfrac{2}{\sqrt{3}+\cdots}}$$

Warning
Plain TEX does fractions and binomials with the `\over` and `\choose` commands that have a different syntax: `n \choose m` instead of `\binom n m`. As natural as the syntax might seem, you should only use the LATEX constructs.

4.4 Size in mathematics

Delimiting symbols come in five sizes:

```
\[
( \big( \Big( \bigg( \Bigg(
\]
```

$$(\big(\Big(\bigg(\Bigg($$

Instead of setting the size manually, the `\left` and `\right` commands can be used. They choose the size based on the content between delimiters. Accordingly, the two commands must be paired correctly – however, the symbols need not be the same. There is also `\middle` for an optional middle delimiter in the matching size.

```
\[
\left\{ a \in \mathbb Q
  \middle| a^2 < \frac 1 2 \right\}
\]
```

$$\left\{ a \in \mathbb{Q} \middle| a^2 < \frac{1}{2} \right\}$$

If you only want to display one of the opening/closing symbols, you can write e.g. `\left( \right.`. The full stop suppresses the output of closing delimiter. See the example on page 82.

Gotcha!
It is not possible to have a line break between `\left` and `\right`, so in long formulas you might need to do the sizing manually.

Good practices
I like to define the following two commands in my documents:

```
\newcommand{\abs}[1]{\left|#1\right|}
\newcommand{\norm}[1]{\left\|#1\right\|}

Absolute value $\abs x$,\\
function norm $\norm f$.
```

Absolute value $|x|$,
function norm $\|f\|$.

They are both semantically meaningful and automatically sized.

There are two basic styles of sizing other mathematics: text style and display style (there are also two levels of sub-/superscript styles). The first is used with inline mathematics and the latter with display mathematics. It is possible to override the style with `\textstyle` and `\displaystyle`:

```
Text with oversize
$\displaystyle \binom n m \frac 1 2$
mathematics, and a display with small maths:
\[
\sqrt{\textstyle \binom n m \frac 1 2}
\]
```

Text with oversize $\displaystyle \binom{n}{m}\frac{1}{2}$ mathematics, and a display with small maths:

$$\sqrt{\textstyle \binom{n}{m}\frac{1}{2}}$$

For fractions, there are the abbreviated `\tfrac` and `\dfrac` commands to achieve this effect.

4.5 Decorations

4.5.1 Accents

You should not use accented Unicode characters like 'â' in math mode. Instead, write `\hat a`. The supported accents are listed in various places online.[5]

Gotcha!
The accents apply to the immediately succeeding character, and do not take subscripts into account. Compare the three:

```
$\hat a_0$ vs $\hat {a_0}$
vs $\widehat {a_0}$
```

$\hat a_0$ vs $\hat{a_0}$ vs $\widehat{a_0}$

To put material over and under symbols, there are the aptly named `\overset`, `\underset`, and `\overunderset` commands:

```
\[
\overset{\eqref{eq:pythagoras}}{=}
\quad \underset{\ast}{X}
\quad \overunderset{a}{b}{C}
\]
```

$$\overset{(5.1)}{=} \quad \underset{*}{X} \quad \overunderset{a}{b}{C}$$

To produce some very cursed versions of big operators, there is also the `\sideset` command:

```
\[
\sideset{^a_b}{^c_d}\sum_{i=1}^\infty
\]
```

$$\sideset{^a_b}{^c_d}\sum_{i=1}^\infty$$

4.5.2 Dots

The amsmath package provides a context-sensitive `\dots` command. Compare the vertical positions of the dots in these two examples:

[5] en.wikibooks.org/wiki/LaTeX/Mathematics#Accents

```
\begin{gather*}
1 + \dots + n\\
1, \dots, n
\end{gather*}
```

$$1 + \cdots + n$$
$$1, \ldots, n$$

Thanks to this, is is usually enough to use `\dots` everywhere in your mathematics. The exception is at the end of expression, since amsmath decides the placement by looking at the following symbol. You can give a hint with commands like `\dotsc` (dots with commas) and `\dotsb` (dots with binary operations) and even `\dotsi` (dots with integrals) for these cases.

Of course, if memorizing these seems too much, the usual `\cdots` and `\ldots` can be used; they just tie the semantics and presentation together.

4.5.3 Braces and highlighting

There are two commands for producing horizontal braces: `\overbrace` and `\underbrace`. They both accept explanatory text as super-/subscript respectively.

```
\[
\overbrace{a^2 + b^2}^{\text{catheti}}
= \underbrace{c^2}_{> 0}.
\]
```

$$\overbrace{a^2 + b^2}^{\text{catheti}} = \underbrace{c^2}_{>0}.$$

The mathtools package also provides similar commands for brackets. They can be optionally customized in thickness and height, but they do not support text:

```
\[
\overbracket[1pt][0.5cm]{a^2 + b^2}
= \underbracket[3pt]{c^2}.
\]
```

$$\overbracket[1pt][0.5cm]{a^2 + b^2} = \underbracket[3pt]{c^2}.$$

Relatedly, let us just note that amsmath provides a `\boxed` command for drawing a box around some mathematics. If the argument continues across a `&` alignment point, you need the mathtools variant `\Aboxed`. A more extensible version is provided by the empheq package.

```
\[
\boxed{a^2 + b^2} = c^2.
\]
```

$$\boxed{a^2 + b^2} = c^2.$$

If you want to emphasize something with colour, the xcolor package offers a `\mathcolor` command that works just like its text counterpart:[6]

```
\[
\mathcolor{blue}{a^2 + b^2} = c^2.
\]
```

$$\mathcolor{blue}{a^2 + b^2} = c^2.$$

4.5.4 Arrows

Mathematicians seem to like arrows. Here is a sample of some; note that some have synonyms for semantically more meaningful code. You probably can deduce the rules for even further variants:

```
\begin{gather*}
\gets \to \mapsto\\
\leftarrow \uparrow \nwarrow \searrow \rightarrow\\
\Leftarrow \Leftrightarrow \Rightarrow\\
\Longleftarrow \Longleftrightarrow \implies\\
\hookleftarrow \rightharpoonup\\
\leftrightharpoons \rightleftharpoons
\end{gather*}
```

$$\begin{gathered}
\gets \to \mapsto\\
\leftarrow \uparrow \nwarrow \searrow \rightarrow\\
\Leftarrow \Leftrightarrow \Rightarrow\\
\Longleftarrow \Longleftrightarrow \implies\\
\hookleftarrow \rightharpoonup\\
\leftrightharpoons \rightleftharpoons
\end{gathered}$$

Sometimes it is useful to put longer sub- or superscripts on arrows. The amsmath package provides extensible versions of arrows. The command names are prefixed with `x`, and the optional argument goes below and the main argument above the arrow:

```
\begin{gather*}
f(x) \xrightarrow[n \to \infty]{\text{weakly}} 0\\
b^2 = 0
\xRightarrow{\eqref{eq:pythagoras}} a^2 = c^2
\end{gather*}
```

$$\begin{gathered}
f(x) \xrightarrow[n \to \infty]{\text{weakly}} 0\\
b^2 = 0
\xRightarrow{\eqref{eq:pythagoras}} a^2 = c^2
\end{gathered}$$

[6] To be precise, it works *very much unlike* its text counterpart in that if you try to use `\textcolor` in math mode, it breaks the spacing.

Gotcha!
These extended arrows should only be used in display size.

While we are on the topic of arrows, let us mention the commutative diagrams that are popular among some mathematicians. See Section 8.6.3 for a solution based on TikZ.

4.6 Theorem environments

4.6.1 The common way with amsthm

Basic theorem/definition/etc. environments are usually done with the amsthm package. The operative command is `\newtheorem`, which takes two required arguments: the environment name and the heading to display. It also takes one optional argument, whose location is significant:

- If it comes after the required arguments, it gives the level at which the environment is numbered. For example, `[section]` means that the counter is prefixed with section number.
- If it comes in between them, it gives the counter to use. If you have already defined a theorem environment named `definition`, then `[definition]` means that the new environment is part of the same numbering.

Good practices
This is now more of a personal opinion, but I strongly believe that all theorem-like environments should share the same numbering. It is infuriating to search for Theorem 5 in a document like

> …, Definition 3, Lemma 3, Theorem 3, Lemma 4, Lemma 5, Lemma 6, Theorem 4, Definition 4, …

where each environment has individual numbering.

The appearance of the environment is chosen with the `\theoremstyle` command. It applies to all following `\newtheorem` invocations. There are three predefined styles:

plain Boldface header, italic text.

definition Boldface header, upright text.

remark Italic header, upright text.

There is also a `\swapnumbers` command if you prefer "1.1 Theorem" instead of "Theorem 1.1".

A complete example of defining theorem environments could be as follows. All environments share the same numbering, which is section-based.

```
\theoremstyle{plain}
\newtheorem{theorem}{Theorem}[section]
\newtheorem{lemma}[theorem]{Lemma}
\newtheorem{corollary}[theorem]{Corollary}

\theoremstyle{remark}
\newtheorem{remark}[theorem]{Remark}
\newtheorem{example}[theorem]{Example}

\theoremstyle{definition}
\newtheorem{definition}[theorem]{Definition}
```

There is also the starred `\newtheorem*` command that does not produce a number. It might be useful in some special cases.

```
\theoremstyle{plain}
\newtheorem*{riemannhyp}{Riemann hypothesis}

\begin{riemannhyp}
The real part of every nontrivial zero of the Riemann zeta function is $1/2$.
\end{riemannhyp}
```

Riemann hypothesis. *The real part of every nontrivial zero of the Riemann zeta function is* $1/2$.

The theorem environment also accepts an optional argument, which is placed in parentheses after the theorem number.

```
\begin{theorem}[Pythagoras]
If $a$ and $b$ are the lengths of catheti of a right-angled triangle,
then the hypothenuse has length $\sqrt{a^2 + b^2}$.
\end{theorem}
```

Theorem 1 (Pythagoras)**.** *If a and b are the lengths of catheti of a right-angled triangle, then the hypothenuse has length* $\sqrt{a^2 + b^2}$.

The amsthm package automatically defines a `proof` environment. The "Proof" heading can be customized with the optional argument, and the environment automatically put a QED symbol at the end. The symbol can be changed by redefining `\qedsymbol` (either in the preamble or locally within the environment):

```
\begin{theorem}
\LaTeX{} course participants are fantastic.
\end{theorem}
\begin{proof}[Sketch of proof]
Just look at them.
\renewcommand{\qedsymbol}{$\heartsuit$}
\end{proof}
```

Theorem 2. *LaTeX course participants are fantastic.*

Sketch of proof. Just look at them. ♡

If the environment ends with a list or display equation, the QED symbol is placed on a new line. Since this is not very pretty, we can give LaTeX a hint with the `\qedhere` command.

```
\begin{proof}[Ugly proof]
It remains to notice that
\[
a^2 - b^2 = (a+b)(a-b).
\]
\end{proof}

\begin{proof}[Nice proof]
It remains to notice that
\[
a^2 - b^2 = (a+b)(a-b).
\qedhere
\]
\end{proof}
```

Ugly proof. It remains to notice that

$$a^2 - b^2 = (a + b)(a - b).$$

□

Nice proof. It remains to notice that

$$a^2 - b^2 = (a + b)(a - b).$$ □

4.6.2 Fancier alternatives

There are also some fancier theorem environments, such as those produced by ntheorem. However, ntheorem does have some subtly surprising behaviors at times, and it might have compatibility issues with some other packages.[7]

A much more compatible way to make your theorems slightly fancier is the thmtools package. It provides an easier syntax in the preamble. It is maybe best illustrated by example:

```
\usepackage{thmtools}

\declaretheorem[numberwithin=section]{theorem}
\declaretheorem[sibling=theorem]{lemma}
\declaretheorem[sibling=theorem, style=definition]{definition}
```

There are also some basic styling options. The "Note to Overleaf users" boxes in these notes are set with the following code:

```
\declaretheoremstyle[
    headpunct={\\},
    spaceabove=8pt,
    spacebelow=8pt,
    shaded={textwidth=352pt,margin=4pt,rulewidth=2pt,
        rulecolor=Green,bgcolor=Green!10}
]{overleaf}
\declaretheorem[numbered=no, style=overleaf,
    name={Note to Overleaf users}]{overleaf}
```

The package documentation is really nice and illustrative, so there is no point in repeating more of it here.

[7] This observation sponsored by: using ntheorem in quite a few projects, including Version 1.0 of these notes. Just one example: with amsthm you are supposed to renew \qedsymbol, whereas with ntheorem you are supposed to call it.

Gotcha!
As of June 2025, there seems to be a bug where hyperlink targets can sometimes be wrong: a link to Lemma 3.4 might lead to Lemma 1.4, because only the ".4" part is recorded. This can be worked around by passing `hypertexnames=false` to hyperref, either as a package option or via `\hypersetup`.

Chapter 5

Tools for serious publications

5.1 Cross-references and hyperlinks

One of the main benefits of a programmable typesetting system is that it takes care of numbering for you. I can say "We talk about floats in Section 7.2, which starts on page 138", and be sure that the numbers are correct. Moreover, the numbers are clickable hyperlinks in the PDF version.

The basic infrastructure consists of two parts. First, the `\label` command is used to annotate points that we want to refer to. This command points to the latest numbered thing: it could be a sectioning command, a numbered list item, a theorem environment, or a numbered equation.

Good practices
Put the `\label` command immediately after the numbered thing. This is both for readability and for maintainability.

Technical side note
LaTeX recognizes the "latest numbered thing" by the `\refstepcounter` command mentioned on page 33.

The `\ref` command reproduces the number that the label was associated to. There is also a `\pageref` command that outputs the page number instead. Finally, `\eqref` puts the number inside parentheses just like in the equation tag.

```
\begin{equation}\label{eq:pythagoras}
  a^2 + b^2 = c^2.
\end{equation}

As we see from Equation~\ref{eq:pythagoras},
that is \eqref{eq:pythagoras},
on page~\pageref{eq:pythagoras}\dots
```

$$a^2 + b^2 = c^2. \tag{5.1}$$

As we see from Equation 5.1, that is (5.1), on page 106…

(Note the `~` that keeps the prefix and number on the same line. This is a good practice.)

Good practices

The label can be quite free-form, and I do not dare give any advice on how things should be named. I personally use a quite verbose style with spaces, like `eq:2d variational formula`, but you could also argue against that.

However, one nice and common practice is to prefix the label with a type. Above, I have used `eq:` to denote equations. Other common ones are `def`(inition), `fig`(ure), `rem`(ark), `sec`(tion), `tbl` (table), and `thm` (theorem).

In the writing phase, it is useful to see the keys in PDF output. The showkeys package does just that: wherever a `\label` is defined, the key is printed to the page margin. Check out the package documentation for options; in particular, the `notref` and `notcite` options suppress the display of keys at `\ref` and `\cite` commands respectively.

5.1.1 Keeping them correct

Warning

Some good things do not last.

The first version of these notes advocated for the cleveref package, which provided a simple and powerful interface. However, the package needed to patch things deep inside the cross-referencing mechanism. The package has not seen an update since 2018, whereas LaTeX core developers have been revamping the internals heavily.

The November 2024 LaTeX kernel update broke theorem references, which could still be worked around with thmtools. Then the June 2025 kernel update broke even further references. In this case a patch was added to the kernel to fix the issue, but this does not fix the underlying incompatibility.

My recommendation is to replace cleveref with zref-clever in new projects. One can define workaround commands (see below) to make the syntax similar.

So you go about writing, adding references to Theorem 4.1 or something similar. Then near the end of your writing process, you decide to call it a Lemma instead. LaTeX takes care of the numbering, but what about all those references coded as `Theorem~\ref{...}`? Chances are, you forget to change some of them to `Lemma~\ref{...}`, and get referee feedback regarding non-existent results…[1]

Shouldn't LaTeX also take care of the reference types? Isn't that exactly the kind of problem computers are supposed to solve? Luckily, there are some packages that do just it.

Good practices
If the preceding paragraph sounded like a true story that has happened to the present author, you are absolutely right.

One of them is zref-clever. It is marked as experimental but seems to be quite stable and regularly updated. It also contains some advanced features missing from cleveref.

This package adds a `\zcref` command that does exactly what you would expect: you write `\zcref{thm:pythagoras}`, and it outputs "theorem 4.1". The command automatically supports all standard LaTeX labels. Theorem types are recognized if you define theorems with amsthm or thmtools.

In the beginning of sentence, you can write `\zcref[S]{...}`. The optional argument tells the package to capitalize and not abbreviate the first reference. In cleveref the operative command is `\cref` or `\Cref`, depending on capitalization.

Reference style can be customized by quite a lot with the `\zcsetup` command. For example, `\zcsetup{cap}` would capitalize all references, which is often desired. These can also be passed as an optional argument to `\zcref`. See the package documentation for all options.

A nice feature of these packages is that you can also pass several references, and they sort out how to present them. That is,

```
\zcref{thm:first,thm:second,thm:third}
```

might produce something like "Theorems 4.1 to 4.3" or "Theorems 4.1, 4.7, and 5.1". Again, see the package documentation for how to customize this.

There is also the `\zcpageref` command to produce page references (with the same support for ranges).

The package does have predefined names for many common theorem environments, but in some cases you need to define the name by yourself. See the how-to section in the package documentation.

[1] On a positive note: such a referee has really read the paper.

Remark
The zref-clever package has some support for non-English languages, even those that have declension. See the package documentation for more.

Good practices
If you have used cleveref, you may be able to convert to zref-clever by something like

```
% Old
\usepackage[capitalize]{cleveref}

% New
\usepackage{zref-clever}
\zcsetup{cap} % etc.
\newcommand{\cref}{\zcref}
\newcommand{\Cref}{\zcref[S]}
```

Make sure that your document still compiles with no warnings and that the references look correct. You might need to do some adjustments to package setup.

5.2 Bibliography management

It seems that every journal has a slightly different preferred style for bibliographic references. Some require full author names, some only initials – some put *et al.* already after one author, some only after three. Not to speak about numbered, author-year, and abbreviated citations. Again, something for computers to do.

There are three standard technologies to create bibliographies, so let us go over them in the order of increasing preferability. Finally, we also mention an unfortunate hack that some journals apparently require.

5.2.1 Citation commands and the manual way

Regardless of the method of creating a bibliography, citations are inserted into the text with the `\cite` command. It takes one argument, the citation key. Several citations can be combined by separating the keys with commas. An optional argument can be used to specify a location within the citation.

```
Documentation can be found in
\cite{TLC, tikz}.

See \cite[Chapter~8]{TLC}.
```

Documentation can be found in [4, 5].
See [4, Chapter 8].

(Note again the use of `~` to keep the location on one line.)

The bibliography is then created within a `bibliography` environment. This environment takes a single parameter, which is the largest expected citation number. That is, `9` means that 1 character is reserved for indentation, and `99` means that 2 characters are reserved.

The bibliography is effectively a list environment where entries are created by the `\bibitem` command. It takes the citation key as its sole parameter. Any text following this command is the bibliography entry, and you are free to format it as you like.

```
\begin{thebibliography}{9}

\bibitem{Oksendal}
    Bernt \O{}ksendal,
    \textit{Stochastic Differential Equations:
    An Introduction with Applications},
    Springer,
    6\textsuperscript{th} edition, 5\textsuperscript{th} corrected printing,
    2010.

\end{thebibliography}
```

Good practices
This method is only advisable for short throwaway notes when collecting a `.bib` file would be too much of an effort. If you use a proper reference management application, this is almost never the case.

Technical side note
It might be useful to wrap the bibliography in a `flushleft` environment. Bibliography entries are often hard to justify nicely, so this might give a visually more pleasing appearance.

5.2.2 BibTeX: the automatic way

Already in Lamport's LaTeX2ε book [3] a better way is described. The **bibtex** compiler whose working we discussed on page 10 automates the process of bibliography creation. In a nutshell:

- You collect bibliographic data into a special `.bib` file.
- You write citations as usual.
- BibTeX takes care of formatting the bibliography in your preferred style, only including the references you have cited.

For this to work, you need to run **bibtex** as part of the compilation. This program creates a `.bbl` file that contains the actual bibliography. (This only needs to be done when the bibliography is changed.)

Remark
When you submit your work to arXiv, you need to send the compiled `.bbl` file. The arXiv processing system does not run the **bibtex** compiler. This means that you should run it manually, verify that the resulting bibliography is up to date, and then send both the `.tex` and `.bbl` files. You should then check that the bibliography and citations appear correctly in the version compiled by arXiv.

Good practices
You should also automate the process of collecting the bibliography. Reference management applications can help you organize your references and are able to download information from bibliographic databases.

Most reference managers support exporting in the `.bib` format. One good, free application is Zotero[2].

A small note as a Zotero user (as of April 2024): unfortunately, the `.bib` export escapes LaTeX commands like `$` into `\$`, and `^` into even more horrendous `\textasciicircum`. You can install the *Better BibTeX* plugin to Zotero to get customizable, better-quality export.[3]

Gotcha!
The **bibtex** compiler is not Unicode-aware, so you need to write non-English characters with TeX special commands. If an author name begins with a non-English character, it may be sorted into an odd position. (You can use the `key` entry to provide an alternative sorting key.)

The `.bib` file format consists of entries that begin with a `@type` declaration, followed by `{}` that contains the citation key and further data in key-value format. This is probably easiest to explain with examples, so let us see some common entry types.

[1] `www.zotero.org`
[1] `retorque.re/zotero-better-bibtex/`

```
@article {OnPyth,
    author = {Firstname Lastname and Secondauthor, Also},
    title = {On {Pythagoras'} Theorem},
    journal = {Journal of Fake Mathematics},
    year = {2024},
    month = {jan},
    volume = {13},
    number = {1},
    pages = {653--675},
    note = {Redacted in Feb~2024.},
    doi = {10.1234/5678}
}
```

Let us dissect this example a bit:

- BibTeX keywords are not case-sensitive, so you could just as well write `@ARTICLE` or `aUtHoR` (please don't).
- This example is of type `@article`, which means a published journal article.
- It can be cited with `\cite{OnPyth}`.
- Values for entries can be wrapped in `{...}` like here or in `"..."`.
- The two authors (F. Lastname and A. Secondauthor) are separated with `and`. Author names can be written in three formats:
  - "First von Last",
  - "von Last, First",
  - "von Last, Jr, First".

  The bibliography style defines how the name will be printed.
- The bibliography style also defines if *Each Word in The Title Will Be Capitalized*, or if *The title should follow sentence case*. To protect words from automatic (de-)capitalization, wrap them inside `{}`; here *Pythagoras* is protected from becoming *pythagoras*.
- Publication month should be given as a three-letter abbreviation.
- The page range should have the `--` en dash as per good style!
- The `note` entry can be used for anything else that should be printed in the bibliography.

- The `doi` entry is not understood by original BibTeX styles, but newer styles (such as those provided by natbib) display them as clickable links. Similarly, there is an `eid` entry type for electronic article IDs used by purely electronic journals.

Let us then see what a bibliographic entry for a book could look like.

```
@book {Pyth2,
    author = {von Lastname, Firstname},
    title = {Pythagoras Reloaded},
    year = {2024},
    publisher = {Fake Press},
    address = {Kumpula, Finland},
    edition = {Third},
    series = {Misunderstanding Historical Mathematics},
    number = {17}
}
```

Now it is also necessary to specify the publisher. The optional `address` field can be used to give the city of the publisher. The edition of the book should be specified in words for BibTeX (but as a number or a complete string like "Third edition" for BibLaTeX…). There are also keys for book series.

If you want to cite an article that appears as part of a book, you can use the `@incollection` type. This is similar to `@book`, but now the name of the article is specified with `title` and the name of book with `booktitle`. You should include both the authors of the article and editors of the book in the data. There is also the `@inproceedings` variant for conference proceedings.

There are also the self-explanatory `@mastersthesis` and `@phdthesis` types; the precise type can be specified with a `type` key. There are also `@techreport` for volumes in a report series, and `@manual` for technical documentation. See Table 5.1 for the typical entry keys in these types.

How to cite a preprint? Unfortunately, BibTeX is quite a bit older than arXiv and friends, so there is no built-in support for referencing online repositories. The best bet is to use the `@unpublished` entry type and write something like "arXiv preprint yyyy.xxxxx" in the `note` field.

Entry type	Required	Optional
`@article`	`author, title, journal, year`	`volume, number, pages, month,` and preferably also `doi, eid`
`@book`	`author` or `editor, title, publisher, year`	`volume` or `number, series, address, edition, month`
`@incollection`	`author, title, booktitle, publisher, year`	`pages, chapter,` and those in `@book`
`@inproceedings`	`author, title, booktitle, year`	`pages, organization,` and those in `@book`
`@manual`	`title`	`author, organization, address, edition, year, month`
`@mastersthesis` `@phdthesis`	`author, title, school, year`	`type, address, month`
`@techreport`	`author, title, institution, year`	`number, address, month`
`@unpublished`	`author, title, note`	`year, month`
`@misc`	At least one of `author, title, howpublished, year, month, note`	

Table 5.1: Some common BibTeX entry types and recommended fields. All entry types also support the `note` field.

Good practices
As mentioned above, the built-in BibTeX styles have no support for DOI codes either. Even if your current style does not show them, you should still include `doi` entries in article data, since

1. The style used by a journal might understand DOIs;
2. BibLaTeX (presented below) understands them very well;
3. They are very useful when you need to cross-check the data.

To print the bibliography, you put a `\bibliography` command at the expected position. This command takes the name of the `.bib` file as its argument. The command must be accompanied with a `\bibliographystyle` command; see below for some common values.

If you want to make a reference appear in the bibliography even though it is not cited in the text, you can use the `\nocite` command. Putting `\nocite{key}` anywhere creates an invisible reference to the key. You can also specify `\nocite{*}` to include all entries from the `.bib` file.

The magic of BibTeX is with the bibliography styles. These customize the look of the final output. The simplest style is `plain`, produced with `\bibliographystyle{plain}` somewhere in the document. This style has numeric labels and references sorted by author and year. A basic bibliography would thus be produced by:

```
\bibliography{my-refs}
\bibliographystyle{plain}
```

Another built-in style is `alpha`, which produces short labels from author names and year; for example [Mit23]. Most journals have their own preferred style.

If you prefer author-year type citations (Mittelbach 2023), the cleanest way is to load the package `natbib` with the `authoryear` option. This requires replacing the bibliography style with `plainnat` or other compatible style.

This package also provides some new citation commands: `\citep` puts the full citation in parentheses, whereas `\citet` wraps only the year; the additional `\citeauthor` and `\citeyear` should be quite self-descriptive. See the package documentation for more details.

Good practices
Even if you don't use author-year citations, the `plainnat` style is useful in its support for DOIs and URLs.

There is a wide variety of pre-built styles in different packages. If you still cannot find a good enough style, you can build your own. The style language is quite complicated, but there is a nice **custom-bib** program that does the effort for you. It is a command-line application (often included in the LaTeX distribution) that asks a series of questions and then outputs a `.bst` file that you can reference with `\bibliographystyle`.

5.2.3 BibLaTeX: the next generation

As good as BibTeX is, it has its drawbacks: no support for Unicode, Internet support only depending on the style, and limited understanding of names outside the "First (von) Last (Jr.)" pattern.

Enter BibLaTeX, an extremely versatile replacement. Its documentation [1] is quite extensive, so let us just compare the essential differences to BibTeX. The `.bib` database format is unchanged, but BibLaTeX adds a lot of new entry and key types.

Gotcha!
BibLaTeX is quite backwards-compatible and BibTeX ignores unrecognized keys, so it is quite easy to write a database compatible with both.

However, there are minor differences in e.g. how the `edition` key is handled. Check the bibliography carefully before submitting!

Most importantly, the `doi`, `eid` (electronic article ID), and `url` keys understood by natbib are natively supported by BibLaTeX. If you don't need BibTeX compatibility, you can specify a preprint with the new syntax:

```
@unpublished{Pyth,
    title = {On {Pythagoras'} relativity theory},
    author = {Firstname Lastname},
    year = {2022},
    eprint = {2211.16111v4},
    eprinttype = {arxiv}
}
```

Supported `eprinttype` values include `arxiv`, `jstor`, `pubmed`, and `hdl`. These all produce clickable hyperlinks.

The syntax to using BibLaTeX is quite different. The package is loaded with `\usepackage{biblatex}`. Here it is possible to pass customization options like the style (see below). Then the `.bib` files are specified with `\addbibresource` commands. Finally, `\printbibliography` outputs the bibliography in the specified place.

That is, a complete minimal example would be:

```
% In the preamble
\usepackage{biblatex}
\addbibresource{my-refs}

% ... in the document
\printbibliography
```

Moreover, to compile the bibliography you need to use the **biber** program. If you cannot use **biber**, it is possible to pass `backend=bibtex` as a package option to use legacy **bibtex** instead. Do note that Unicode support is then unavailable, and some more advanced features might not work.

Note to Overleaf users
Overleaf takes care of calling the right bibliography compiler automatically.

Bibliography styles are customized with package options. (When converting a project to BibLaTeX, you should also remove any references to natbib and friends.) To use the author-year style, you only need to pass `style=authoryear`. The natbib commands like `\citep` are also implemented by BibLaTeX when `natbib=true` is passed to the package.

There are quite a few built-in styles [1, Section 3.3], and even more can be found in extension packages.

Let us finally comment on one cool feature provided by BibLaTeX, if you are not yet convinced to convert to it: back-references. Just put `backref=true` as a package option, and the bibliography now contains references to pages where the corresponding entries have been cited. This is surprisingly useful! (See the bibliography of these notes for an example.)

Good practices
Joking aside, unfortunately many journals are still to adopt BibLaTeX. This means that one still also needs to work with legacy BibTeX as well.

5.2.4 Embedding the bibliography

Some journals require you to submit only one `.tex` file, with the bibliography embedded in the text. This is no reason to go to the manual entry; instead, you can copy and paste the BibTeX output:

0. Do all your editing, and only do the following as the last step.
1. Compile the bibliography with **bibtex**.
2. Open the produced `.bbl` file. Observe that it contains a `bibliography` environment!
3. Select all and copy.
4. Replace the `\bibliography` command in your document with the pasted environment.

If you use BibLaTeX, there are some good news and some bad news. Good news: you can use BibLaTeX with all its benefits, since the journal system is not in your way! Bad news: the `.bbl` file is not readily copy-pasteable.

There is a hack for doing this in the form of biblatex2bibitem package. Check out the package page on CTAN for more information.

5.3 Tables of contents

A table of contents is printed by the `\tableofcontents` command. If you have used the standard LaTeX sectioning commands, everything works as it should. See the notes in Section 2.7 for how to abbreviate the section titles in the table of contents.

Starred sectioning commands like `\section*` do not produce a number, and they are not added to the table of contents by default. If you need to manually add something to the table of contents, you can use `\addcontentsline`. It takes three arguments: the name of the contents list (here `toc`), the sectioning level, and the entry to add. For example:

```
\addcontentsline{toc}{section}{A phantom section}
```

By default, the bibliography does not appear in the table of contents. If you have made the bibliography manually or with **bibtex**, you can load the tocbibind package or use `\addcontentsline`. If you use biblatex, you can add the `[heading=bibintoc]` optional argument to `\printbibliography`.

It is also possible to create lists of tables and figures. These are produced by the `\listoftables` and `\listoffigures` commands. To add entries manually, you need to pass `lot` and `lof` respectively to the `\addcontentsline` command.

5.3.1 *Indexing**

In the uncommon case that you need to prepare an index, let us very briefly go over the basic syntax. You need to put `\makeindex` in the preamble, and the index is printed with `\printindex`.

As with bibliographies, an external program is used to compile the index. This used to be done with **makeindex**, but nowadays a better option is **upmendex**; the latter is fully compatible with **makeindex** syntax but additionally supports Unicode.

Index entries are added with the `\index` command. At the simplest, it takes the text to be inserted to the index. The index will point to the page where command is executed. However, the syntax also supports subindex entries and customizing the look of the entry.

A subindex entry like “counters, list” is created by putting `!` between the index key and the subindex key. If you need to control the placement of the key, or want to apply styling, you can put `@` between the sorting key and the visual appearance.

See the examples below for how this is done.

```
\index{counters} % Basic index entry for 'counters'
\index{counters!list} % Entry for 'counters, list'
\index{tex file@\texttt{.tex} file} % Sorted with the letter 'T', not '.'
```

As a final note, there is also the `\see` command that yield "*see* ..." entries. These index entries do not feature a page number, so the command can be placed anywhere. As a real example from these notes:

```
\index{dots per inch!\see{DPI}}
```

Good practices
A good index is hard to make, but extremely useful for a reader. There are even books on the art of indexing.

These notes are not to be looked as a model for good indexing; most of the index is created automatically by the commands that format command, environment, and package names. The manually created part of the index has not received the love it would deserve.

5.4 Tools for the editing process

5.4.1 Line numbers

Some journals require line numbers for the referee copy of a submitted work. This is usually achieved with the lineno package.

The syntax is very simple. Load the package and specify `\linenumbers` in the preamble or beginning of the document. You can pause the numbering with `\nolinenumbers` and resume it with `\linenumbers` again.

You can specify the `\modulolinenumbers` command to print a number only every fifth line; the gap can be specified as an optional argument like `\modulolinenumbers[2]` (for a number on every other line).

By default, equation environments are not numbered. Load the package with `[mathlines]` optional argument to enable this.

Gotcha!
Built-in support for amsmath was only included in July 2022, so you may need to update packages in your LaTeX distribution.

There is also the `[pagewise]` package option to reset the line numbers for each page, but be advised: it might require two LaTeX runs to be correct.

Technical side note
This package needs to do some deep magic, since line numbers get fixed very late in the typesetting process.

5.4.2 Line spacing

Some journals also require some extra space between lines, typically so-called 1.5 spacing. The LaTeX internal parameters are a bit complicated, so it is easiest to load the setspace package and use its commands `\singlespacing`, `\onehalfspacing`, and `\doublespacing` (possibly in the preamble).

5.4.3 latexdiff

The **latexdiff** program is an external script that is immensely useful in editing. It compares two `.tex` files and outputs a `.tex` file where any differences between the two are highlighted.

Good practices
Many referees and journal editors really appreciate getting a latexdiff along with a revised version. They are also useful to pass between coauthors, and sometimes even for your own reference.

To make this work, you should preserve a copy of each submitted version for later diffing. (This combines really well with version control software, but that is a topic for a different course.)

The **latexdiff** program is a Perl script, so you will need a Perl interpreter installed. It is usually installed by default on Linux and macOS systems, but for Windows you will need to install it.[4]

On Linux and macOS the usage is as simple as

```
perl path/to/latexdiff.pl old.tex new.tex > diff.tex
```

It could be that **latexdiff** is even on your search path, in which case it suffices to execute `latexdiff`.

On Windows, things are slightly more complicated. Windows writes the output of the script (through the `>` operator) in the UTF-16 encoding, which is not compatible with UTF-8. If your document contains Unicode characters, they will be garbled and probably fail to compile. To fix this, we need to tell Windows to produce output in UTF-8.[5]

In the legacy Command Prompt, you should execute the following commands:

[4] There is also a "portable" distribution called Strawberry Perl, if you are uncomfortable about installing things.

[5] The reason for this mismatch is historical. The Windows command-line interface was introduced with Windows NT in 1993, and UTF-8 saw the light at around the same time. It took some more years for UTF-8 to become the standard choice. The developers of Windows have been as serious as the LaTeX developers about backwards compatibility, so the default is stuck to be UTF-16.

```
chcp 65001
perl path/to/latexdiff.pl old.tex new.tex > diff.tex
```

Here the `chcp` command stands for "choose code page", and UTF-8 is numbered to be 65001. (Code pages are synonymous for LaTeX encodings like `latin1`.)

In Windows PowerShell, you can enter the legacy Command Prompt by executing `cmd`. (The present author is not aware of a robust PowerShell-native method that would work across all the versions in active use.)

The new `.tex` file produced by **latexdiff** is a merged copy of the two input files, with additions wrapped inside `\DIFadd` and deletions wrapped inside `\DIFdel` commands. By default, these produce coloured and underlined text.

Gotcha!
Unfortunately, **latexdiff** sometimes produces LaTeX code that fails to compile. For example, a subscript and the text preceding it might be separated by a `\DIFadd` command.

You need to manually fix up these issues, which makes for an exercise in LaTeX debugging. If you keep your lines reasonably short, at least you get some assistance from the line numbers.

Gotcha!
Sometimes the differencing algorithm also produces non-obvious results: If you delete some text and add new text in its place, you might not get two neat "deletion" and "insertion" blocks. Instead, the diff is a salad of deleted, inserted, and kept-intact pieces.

It is impossible to develop perfect heuristics for detecting the intended changes. In principle, you could manually edit the diff, but most likely you should just keep it as is. One can always look at the two versions side by side; what is important is that the diff shows that *something big* has changed at that position.

If your document is split into multiple `.tex` files, you only need to give the paths to the main files. The algorithm is clever enough to parse `\input` and `\include` commands.

5.5 Embedding PDF files

Including figures in PDF format is discussed in Section 7.1, but what about embedding complete documents? The most common use case is, of course, the article-based PhD thesis.

The pdfpages package solves this problem. It provides an `\includepdf` command that extracts a page range from a PDF document. The extracted pages are laid out on new pages, centered and scaled to fit.

The package documentation clearly explains the customization options, but really the only necessary argument is `pages`. It takes a range of pages like `{1,3,5}` or `7-11`. To include all pages, just write `pages=-`.

```
% Preamble
\usepackage[final]{pdfpages}

\includepdf[pages=-]{article1.pdf}
```

The `final` package option ensures that pages are embedded even if the rest of the document is in `draft` mode.

New in LaTeX3
Accessibility information in embedded PDF files is lost, and the package might have some other compatibility issues with `\DocumentMetadata` features (as of May 2025).

Chapter 6

Further customization

6.1 Alternative LaTeX compilers

During the years, the preferred output format of TeX has changed:

1. The original implementation of TeX produced DVI files – this was a device-independent format that could be processed into commands to be sent to the printer or phototypesetting machine.

2. Simultaneously in the 1980s, Adobe developed the PostScript format. PostScript is a language for expressing complex drawing commands to printers.[1] It became the universal standard, and in the 1990s a lot of TeX documents were put online as `.ps` files. These files were produced from DVI files by a converter.

3. PDF (Portable Document Format), also by Adobe, was created in the 1990s and has become the dominant standard. PDF encapsulates a subset of PostScript, image data, and metadata about the document.

Originally, PDF files were produced by first converting DVI to PostScript and then to PDF. In early 2000s, a new implementation of TeX appeared: **pdftex**. It produces PDF files directly and supports some special features of the format: For example, the pdflscape package can tell the PDF reader to display a landscape page in rotated mode.

Technical side note
The only difference between **pdftex** and **pdflatex** is whether the LaTeX macros are automatically loaded. These notes make no effort at being consistent with the names.

[1] Processing drawing commands requires processing power and memory. The original Apple laser printers were *more powerful* than the Macintosh computers used to lay out the documents!

Another development was with the character sets. Original TEX used 7 bits per character, because 128 characters is enough for anybody (that is, anybody writing in English). In the 1980s, this was recognized as unsustainable, so the character set was *doubled* into 8 bits. Any further extensions to that are based on swapping fonts on the fly.

This limitation applies also to **pdftex**, but in the current[2] age of Unicode there are now alternatives. The two major alternative compilers are

- **luatex**, which is based on **pdftex** source code. It uses Unicode as its native character set. As the name suggests, extension packages can be also written in the Lua programming language. Some packages like TikZ use the Lua support for a speed boost.
- **xetex** is an independent Unicode-native development that uses a completely different PDF backend. It produces a bit smaller PDF files than **luatex**, but it is less actively maintained.

Both of these "modern" engines support loading arbitrary OpenType fonts with the fontspec package, unlike **pdftex**. (This is discussed in Section 6.3.) The native use of Unicode also means that letters like 'ä' really are output as the glyph 'ä', not ¨a superimposed on top of each other – something you only see when copying text from a PDF file.

Note to Overleaf users
You can choose the compiler in the settings menu that can be opened at the top-left corner. Most editor programs have a similar menu.

Good practices
Based on the discussion above, it would be natural to suggest always using **luatex**. For personal writing, either of the Unicode engines should be the first choice.

Alas, arXiv and many publishers still use **pdftex** in their pipeline. This means that we are still stuck with making our articles **pdftex**-compatible. Hopefully the situation will change soon enough.

6.2 Accessibility of documents

The PDF file format is principally a set of instructions to a printer: draw a curve here, another there, and so on. This poses a problem for accessibility. Some examples of accessibility issues are:

- Can a "screen reader" program read the document aloud for a blind person?

[2]Read: past 20 years.

- Can a visually impaired person increase the color contrast on graphics?
- Can the text from the document be copy-pasted into another program?

Note that the last point might not sound like an accessibility issue, but it is caused and solvable by the same reasons.

There are three layers of accessibility work going on:

1. The PDF format can include semantic metadata about the document contents. LaTeX is increasingly better at writing this metadata, but it might require you to opt in to the new behavior.
2. LaTeX can be compiled into another format, such as HTML web page. This approach is increasingly used by major publishers and arXiv. It requires you to use a sensible subset of LaTeX.
3. Source code is very accessible to a screen reader program; a well-written `.tex` file is a good workaround. This still requires you to write clean code.

We will briefly explore each of these layers below.

Good practices
Accessibility is for everybody. Most people benefit from some accommodations at some point in their life. Even without an "obvious" disability, think about reading your document

- on an ebook reader with grayscale screen (visual impairment),
- on a mobile phone in bright sunlight (ditto), or
- by a sleep-deprived parent between meetings (cognitive impairment).

Not all accessibility issues have technological solutions – they also require you to think about how you present your message. A complicated figure is hard to read, no matter the technology used to produce it.

6.2.1 PDF accessibility

This is the core accessibility work done by the LaTeX team during the last few years. The goal is to embed semantic metadata about the document into the PDF format. A lot of the semantics is in the `.tex` source in form of sectioning commands and such.

The translation is hampered by the need to rewrite LaTeX internals without breaking compatibility, and all the abuse of commands to get semantically incorrect but visually good results.

To enable PDF metadata, you should do the following steps:

- Use a modern LaTeX compiler like **luatex** or **xetex**. Since **luatex** is more actively maintained, it is preferable.
- Add `\DocumentMetadata` as the first line (before `\documentclass`) of the code.
- Load the hyperref package.
- Consider loading the unicode-math package (and remove other mathematics symbol and font packages; see page 88).

Warning
Since some behavior changes subtly, you should take these into use only in new projects and only when everybody involved (including the publisher) uses a Unicode-native LaTeX pipeline.

Be extra careful with checking the output when converting old projects, or if you need to compile the project with both old and new toolsets.

Warning
As of April 2024, arXiv *does not* use Unicode-native LaTeX tools; their process involves **pdftex**. For arXiv submissions, follow the next section instead.

The `\DocumentMetadata` command tells LaTeX that you opt in to new and rewritten behaviors. For example, it enables some new features in the hyperref package.

New in LaTeX3
This work is still heavily in progress. There are some arguments to `\DocumentMetadata` that enable further in-development features. As of April 2024 these include tagging of mathematics and tables.

Since these arguments are unstable, they are not documented here and should not be used in "real" documents quite yet. You can follow the progress through LaTeX release newsletters:

`www.latex-project.org/news/`

In order for the tagging to work, you need to produce semantically correct code. That is, use `\subsection*` when you need an unnumbered subsection, and only then – do not abuse it for bold, heavy text. Similarly, use `\emph` to emphasize, instead of `\textit` or `\textbf` that carry only visual meaning.

6.2.2 HTML conversion

Another option is to skip PDF altogether. The most common alternative format is HTML, the technology behind web pages. It has a few benefits:

- It is processed on the user device, so e.g. colours can be customized by the reader.
- The text can be reflowed based on the screen size and magnification factor.
- Existing screen reader programs can already read out web pages.

However, there are a few drawbacks too:

- TEX is built on the model of printing on paper; it assumes a fixed layout in many places.
- The usual LATEX compilers only produce PDF output or equivalent.

The HTML approach is now pursued by several publishers and arXiv, together with e.g. American Mathematical Society. ArXiv uses a tool called LaTeXML[3] that is essentially a new TEX compiler that selectively reimplements some packages.

Since LATEX is already semantically focused, this works surprisingly well. A lot of packages are already supported. Importantly, TikZ can output its pictures also in the SVG (Scalable Vector Graphics) format used together with HTML.

Good practices
Load and use only the packages that you really need. Prefer mainstream packages supported by the arXiv converter. ArXiv has a help document outlining some more best practices for HTML conversion.[4]

6.2.3 Accessible source code

Both of the above methods require you to use semantically meaningful commands: otherwise, the automated tools are unable to do tagging/conversion. Yet writing good semantic LATEX code is also an accessibility feature: you can always read the source code.

Remark
Source code posted to arXiv is public and downloadable.

[3] `github.com/brucemiller/LaTeXML`
[4] `info.arxiv.org/help/submit_latex_best_practices.html`

If you want to set vectors in bold font, then do not write `\mathbf` but redefine `\vec` to set vectors in bold style:

```
% Preamble
\renewcommand{\vec}{\mathbf}

$\vec a$
```

a

Now it is immediate from the source code that `a` refers to a vector. It can not be confused with other things marked with boldface. Besides, if you change your mind about the style, you only need to change one command in the preamble.

6.3 Typefaces

We will discuss two methods for changing the typeface in your document. In neither case, we do not comment too much on the aesthetic qualities – use at your own risk!

Warning
When changing fonts, especially the one used for mathematics, make sure that all the special characters still appear correctly. The character support of different fonts varies wildly.

6.3.1 Fonts via packages

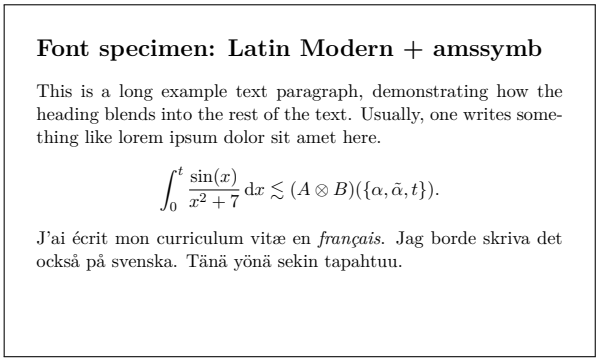
Several common typefaces are packaged on CTAN as ready-to-use packages. This is a preferable way since the package author might have done some necessary typographical tweaks already for you.

To find packaged typefaces, you can use the *LaTeX Font Catalogue*[5] maintained by TeX Users' Group. Each catalogue entry displays the code needed to load the font with typical installations.

In the *Font implementation* section, you should check what types are available. Type 1 fonts can be used with pdfTeX, whereas LuaTeX and XeTeX additionally support (and prefer!) OpenType or TrueType fonts.

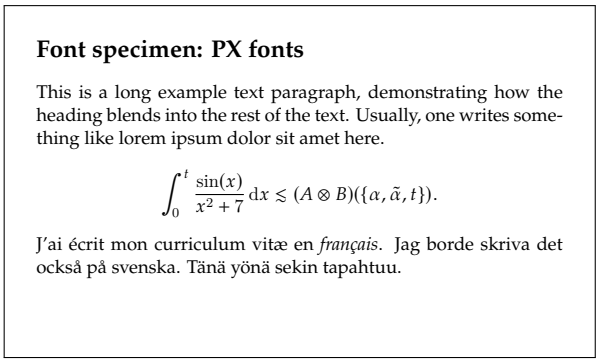
Let us see a few examples of common fonts. First, for comparison, here is a small standard text written with the default Latin Modern typeface (the $\lesssim$ symbol comes from amssymb):

[5] `tug.org/FontCatalogue/`



The Palatino typeface designed by legendary Hermann Zapf is a classic and commonly used also in the TEX world. It is packaged as the PX text and math fonts. The font automatically comes with `\lesssim`, so we do not need to load the symbol package. (See the next subsection for a different version with OpenType fonts.)

```
\usepackage{newpxtext,newpxmath}
% No need to load amssymb
```



Technical side note
Typeface designs cannot be copyrighted, but the names are trademarks. This is why near-identical implementations of the same typeface can appear with very different names.

As a slightly different example, we use the Noto font. This typeface design by Google aims to contain *all Unicode characters*.[6] It too can also be loaded with the mechanism described in the next section.

[6] Its name comes from **no to**fu, since the rectangular 'missing character' symbol in web browsers is sometimes called 'tofu'.

```
\usepackage{notomath}
% No need to load amssymb
```

Font specimen: Noto

This is a long example text paragraph, demonstrating how the heading blends into the rest of the text. Usually, one writes something like lorem ipsum dolor sit amet here.

$$\int_0^t \frac{\sin(x)}{x^2+7}\,\mathrm{d}x \lesssim (A \otimes B)(\{\alpha, \tilde{\alpha}, t\}).$$

J'ai écrit mon curriculum vitæ en *français*. Jag borde skriva det också på svenska. Tänä yönä sekin tapahtuu.

For one more example, let us see a sans serif typeface. The Source Sans typeface is an open source font by Adobe. When the package is loaded with the `default` option, sans serif becomes the default font family. However, it does not come with mathematics support. Without further tweaks, the display mathematics comes in a very different typeface!

```
\usepackage[default]{sourcesanspro}
\usepackage{amssymb}
```

Font specimen: Source Sans + Latin Modern Math

This is a long example text paragraph, demonstrating how the heading blends into the rest of the text. Usually, one writes something like lorem ipsum dolor sit amet here.

$$\int_0^t \frac{\sin(x)}{x^2+7}\,\mathrm{d}x \lesssim (A \otimes B)(\{\alpha, \tilde{\alpha}, t\}).$$

J'ai écrit mon curriculum vitæ en *français*. Jag borde skriva det också på svenska. Tänä yönä sekin tapahtuu.

Many sans serif packages have options to make sans serif the default style. The same can be achieved manually with

```
\renewcommand{\familydefault}{\sfdefault}
```

6.3.2 OpenType fonts via fontspec

Remark
This section only applies to LuaTeX and XeTeX.

Like character sets, also font formats have undergone a parallel evolution between TEX and the rest of the world. Nowadays, fonts are distributed in the OpenType format (and its predecessor TrueType).

The fontspec package allows loading OpenType/TrueType fonts on the Unicode-native compilers. In the preamble, the default roman, sans serif, and typewriter fonts are set with the respective commands:

```
% Note: only works on Windows, where these are installed!
\setmainfont{Times New Roman}
\setsansfont{Comic Sans MS}
\setmonofont{Consolas}
```

In this example, we used the system font names.

Technical side note
A XeTeX-specific issue: If fontspec cannot find the font, XeTeX tries to fall back to the old MetaFont technology. This manifests itself as log messages about **mktexmf** failing to build (non-existent) files.

It is also possible to give file names to fonts, if you have the font files in the same folder tree as your document.[7] In this case, you might need to specify file names for each weight separately:[8]

```
\setmainfont{FiraSans-Book.otf}[
    BoldFont={FiraSans-SemiBold.otf},
    ItalicFont={FiraSans-BookItalic.otf}]
```

Instead of overriding the default fonts, it is also possible to define new font families with `\newfontfamily`:

```
% Preamble
\newfontfamily{\comicfont}{Comic Sans MS}

% Document
Some people hate {\comicfont Comic Sans},
but its history is often {\comicfont misunderstood}.
```

[7] Or anywhere on TEX's search path.

[8] This example takes advantage of it, and uses the Book and Semibold weights instead of Regular and Bold.

All these font loading command support *a lot of* optional arguments – see the package documentation. These can be used to control the appearance of the typeface and to enable stylistical variants of some letters.

For example, when using typefaces not originally designed to play together, it might be useful to slightly adjust the sizes of fonts:

```
% Fits a bit better with Latin Modern
\newfontfamily{\comicfont}{Comic Sans MS}[Scale=0.9]
```

The unicode-math package introduces the further `\setmathfont` command that works just like the commands above, only for the math mode. The font needs to include the mathematical symbols, which most fonts do not have! This package also turns all mathematical symbols into proper Unicode characters that can be copied from and searched within the PDF.

Let us illustrate one nice combination of OpenType text and mathematics fonts. TEX Gyre Pagella is another version of Palatino, usually included with LATEX distributions. The Asana Math font is also part of TEX Live, but here we load it from an included file:

```
\usepackage{fontspec}
\setmainfont{TeX Gyre Pagella}
\usepackage{unicode-math}
\setmathfont{Asana-Math.otf}
% No amssymb loaded
```

Font specimen: TEX Gyre Pagella + Asana Math

This is a long example text paragraph, demonstrating how the heading blends into the rest of the text. Usually, one writes something like lorem ipsum dolor sit amet here.

$$\int_0^t \frac{\sin(x)}{x^2+7}\,\mathrm{d}x \lesssim (A \otimes B)(\{\alpha, \tilde{\alpha}, t\}).$$

J'ai écrit mon curriculum vitæ en *français*. Jag borde skriva det också på svenska. Tänä yönä sekin tapahtuu.

Technical side note
Another odd corner of copyright law: typeface designs cannot be copyrighted, but *font files* are! The reason is that the OpenType format is a small programming language, which makes fonts be classified as computer programs.

Only use fonts that you have the right to use: those that come with your computer system, those you have bought, or ones available under a permissive license (like the SIL Open Font License). As long as you have the license to use a font, you can freely distribute any documents typeset with it.

6.3.3 Point sizes and microtypography

Let us just briefly comment on two more nuances of setting text. LaTeX provides the semantically meaningful font size commands (page 40), but what if you need to specify the exact point size?

The low-level `\fontsize` command can be used in this case. It takes two arguments: the desired size of the font, and the new baseline skip value. If the font does not support arbitrary size, the closest match is chosen. This declaration must be followed by `\selectfont`, which does the hard work of loading the font.

```
\fontsize{12pt}{14pt}\selectfont
This is set in 12~pt font,
with 14~pt baseline skip.
Some extra text to illustrate
the line heights.
```

This is set in 12 pt font, with 14 pt baseline skip. Some extra text to illustrate the line heights.

Technical side note
LaTeX might need to do a fair bit of work to prepare a font for the given size, so you might hit some internal limitations if there are too many different sizes in use.

Some more typographical tweaks are provided by the microtype package. Once loaded, it adds features like hanging punctuation (punctuation characters extending into the margins) and some tiny spacing adjustments. We are now talking of extremely small typographical details.

Warning
Support for microtype features depends a lot on the compiler. If you use the package, be aware that the page layout can change significantly between different compilers.

Warning
Some fonts might not pair well with the built-in adjustment parameters.

6.3.4 Emoji

Emoji are Unicode characters, so in principle they can be included just as is in the source code. The problem is choosing a font that can display the characters.

The emoji package automatically chooses an available emoji font (that is, the output will depend on your operating system!). It only works with recent LuaTeX (2020 or newer). The command provides readable shorthand names for the characters (listed in documentation):

```
% Preamble
\usepackage{emoji}

% In text
\emoji{thinking-face}
```

There is also the manual way of loading the emoji font with fontspec:

```
% Preamble
\usepackage{fontspec}
\newfontfamily{\emojifont}{Noto Color Emoji}[Renderer=Harfbuzz]

% In text
{\emojifont ...}
```

Here, `...` should be replaced with the emoji (not displayed here, since these notes do not have an emoji font set up!). You need to install the emoji font or replace it by the system emoji font. (On Windows, it is called `Segoe UI Emoji`; on macOS it is `Apple Color Emoji`.)

Note to Overleaf users
The Overleaf editor cannot handle the Unicode emoji characters as of May 2024. If you use Overleaf, you need to use the emoji package.

Warning
With the manual way, black-and-white emoji can be displayed by both XeTeX and LuaTeX, but color emoji requires LuaTeX with the HarfBuzz renderer. If you use fontspec to load an emoji font, you need to pass `Renderer=Harfbuzz` as an optional font feature. This option is effective only with LuaTeX.

Chapter 7

Figures and tables

7.1 Embedding pictures

Before we go into the details of figure environments, let us first recall the syntax for embedding pictures and some good practices.

TEX does not natively understand picture files; instead, they are placed with cooperation between the graphicx package and the particular LATEX compiler (pdfLaTeX or friends).

Warning
Again, careful with the spelling! The graphicx package replaces the old graphics package, which has similar basic syntax but less flexibility.

The basic command is `\includegraphics`. It takes the path (relative to source file) of the picture file, and some optional arguments that we describe below. On Linux file paths are case-sensitive, so be careful with the spelling (otherwise your file will compile only on case-insensitive systems like Windows and macOS). Only `/` can be used as the directory delimiter. The file extension (here `.jpg`) is optional.

```
\includegraphics[width=\textwidth]
    {pictures/TheDogs.jpg}
```

There are quite a few optional parameters using the key=value syntax:

width Sets the width of the picture in document units. Can be any expression like `width=0.5\textwidth` or `width=8cm`. Height of the picture is chosen to match the aspect ratio.

height Alternatively, the height can be fixed and width chosen automatically. If both `height` and `width` are set, then the picture is *stretched* to the given height and width.

angle Rotation (in degrees counterclockwise). The `totalheight` option can be set to restrict the height of the final rotated image.

origin Origin for the rotation. Most useful value is `c` for center; others are `l` for left, `r` for right, `t` for top, `b` for bottom, and combinations of the five.

bb, clip Named for "bounding box", the `bb` parameter defines the region of the image to use in size computations. The `clip` option then crops the image to this size. If this option is not set, then the image extends outside the region reserved for it!

The bounding box is specified as `bb=1 2 3 4`; in this example the lower-left corner is at $(1, 2)$ and the upper-right at $(3, 4)$, and the origin $(0, 0)$ is at the lower-left corner of the image. By default the unit is "big points" ($1/72^{\text{th}}$ of an inch), but other TEX length units can be used. Due to the choice of units, this is most useful with PDF images.

draft This is usually passed as a package or document class option. It suppresses the inclusion of pictures into the final output; only boxes of the correct size are included. (The size computation requires the picture file to exist!)

... and some more can be found in the package documentation. I would, however, do any complicated graphical things in a proper graphics editor.

```
\includegraphics[bb=2.3cm 1.4cm 7cm 6cm,clip]
    {pictures/TheDogs.jpg}
```

```
\includegraphics[totalheight=3cm, angle=45]
    {pictures/TheDogs.jpg}
```

```
\includegraphics[height=3cm, draft]
    {pictures/TheDogs.jpg}
```

pictures/TheDogs.jpg

What type of file should your graphics be? Nowadays the choice is between essentially three formats:

PDF For vector graphics like graphs and line art. Vectorized graphics can be zoomed without blurring and embedded text is still selectable.

However, for very complicated graphics (say, a scatter plot with 1000 points and transparency), it might be better to use a rasterized format.

PDF graphics are rendered on the reader's device, so on a slow device the picture can take a while to load!

PNG For complex line art. PNG is a lossless format, meaning that the picture is reproduced pixel-perfectly. The compression algorithm is designed for computer graphics with solid colors and simple gradients. It is not suitable for photographs (due to massive file sizes).

JPEG For photographs. The algorithm is lossy, meaning that compression artifacts can be seen when zoomed in. The artifacts are hard to see in photographs, but are very distinguishable in line art and text.

The algorithm can be set to various compression levels. You should experiment with your particular picture, but values between 80–90 usually yield a good balance between quality and file size.

What size should the picture be? To answer this, we need to talk about DPI – dots per inch. Computer screens have something between 100 to 300 pixels per inch. The standard for printed documents is usually 300 dots per inch, and up to 600 dpi for high-quality line art.

This means that if your image will be printed 5 cm wide, its pixel width must be at least

$$\frac{5 \text{ cm}}{2.54 \text{ cm/in}} \times 300 \text{ pixels/in} = 590 \text{ pixels}.$$

Conversely, the pixel width should not be much larger than this: the extra resolution would only be seen at very large zoom levels. LaTeX does not rescale images to a reasonable DPI value, so extra-large images cause the final document to have larger file size.[1]

The PDF format uses inches instead of pixels. Still, by setting the image size to the correct value you ensure that text and lines are drawn at a scale matching the rest of the document.

Good practices
To summarize:

- If it is a photograph, use JPEG with suitable size and compression level.
- If you're exporting graphics from an application or script, set the figure size and DPI to the intended size and export as PDF. If PDF is not available or the image file gets very large, use PNG.

[1] You might ask whether file sizes matter in the age of terabyte hard drives and fiber optic internet. I have downloaded enough many articles onboard long-distance trains to be allergic to any unnecessary kilobytes. Fast connections are not universal.

Good practices
Avoid screenshots; always export your graphics directly from the application if possible. Most operating systems render text with smoothing adapted to your screen – the (no longer adapted) smoothing is visible in a screenshot.

In particular, Windows uses a sub-pixel rendering algorithm that is very visible as coloured artifacts around black text.

Good practices
If you are a Python user, you probably use matplotlib to create graphics. It can draw text using Computer Modern fonts and proper mathematical typesetting.

- `matplotlib.org/stable/users/explain/text/mathtext.html`
- Some quick hints: `duetosymmetry.com/code/latex-mpl-fig-tips/`

New in LaTeX3
Since late 2021, `\includegraphics` has also supported setting *alt text*: a textual description of the image. It is passed as the optional `alt` parameter; see the example on page 140. This text is used when the document is read aloud by a screen reader program.

The alt text should give enough information to understand the message in the image. Writing good alt text is not easy, but there are some resources online. For some examples, see the decision tree and links therein by W3C[2].

7.2 Floats

Numbered figures and tables in LaTeX are collectively known as *floats*.[3] The algorithm to place floats is again very complicated, with some 20 tunable parameters; these are explained in detail in [4, Chapter 7.1].

A rough summary of the rules is:

- Floats can appear inline, at the top or bottom of the page, or on a separate page. The possible locations are preset by the document class and can be restricted in the code. In two-column mode, there is also the top of page spread across the two columns.

[2] `www.w3.org/WAI/tutorials/images/decision-tree/`. The WWW Consortium maintains web standards, so parts of the tutorial are web-specific.

[3] Named after their tendency to float around into inconvenient places.

- Floats are always placed in order within a float class. That is, Figure 1 always appears before Figure 2 and Table 1 before Table 2. However, tables and figures are placed independently, so their order with respect to each other might be anything.
- LaTeX first tries to put the float on the page where the surrounding code is set. If that fails, the float goes into a queue to be reconsidered for the next page. Consequently, a float might appear above the surrounding text on the same page, but never on an earlier page.
- LaTeX might produce one or more pages containing only floats if the queue is full enough.
- At the end of document, all remaining unplaced floats are printed.

To override the float placement, you can put one or more letters in an optional argument after the `\begin{figure}` (or similar) command. These specifiers *exclude* the unmentioned specifiers, so defining this always restricts LaTeX's opportunities.

`t` Allow putting the float at top of page.

`b` Allow putting the float at bottom of page.

`p` Allow putting the float on a separate page.

`h` Tries to put the float inline at the given position. If there is not enough space, the specifier is converted to `t`.

`!` Ignores certain restrictions about the number of floats on a single page.

Quite often the default is `tbp`, which corresponds to what is usually considered good style. You can use `h` to suggest placing the float inline, but it might still appear at the top of next page. If you really need to force an inline float (and most probably you should not), you can load the float package and use its `H` specifier.

Sometimes the algorithm leads to very many float pages that consist of mainly whitespace. If you are in control of the document layout, then the fewerfloatpages package might be useful. Its documentation also contains some more explanation of the placement algorithm.

Good practices
As with overfull lines, you should not tweak the placement of floats until at the very end. Small changes to the text might change the page layout significantly. Unless you get a visually unpleasing arrangement, avoid adding placement restrictions.

Gotcha!
Due to the algorithm, there are a few surprising things:

- The order of placement specifiers is irrelevant; LaTeX always evaluates its options in the order "here"–"top"–"bottom"–"queue it".
- In two-column mode, `b` has no effect unless bottom-of-column floats are allowed with a special package. If `b` is the only placement specifier, this ensures that your float will not be set!
- In some rare cases, the placement of floats might push footnotes into an incorrect page.

Figure captions are generated with the `\caption` command. This command also updates the figure number, so a possible cross-reference `\label` must come after it.

As a complete example, Figure 7.1 is produced with the following code. The specifier `[ht]` forces LaTeX to put the figure either immediately after this paragraph or on top of a following page; it cannot appear at the bottom of a page. It is usually customary to center the figure, which is done with the `\centering` command that applies until the environment ends.

```
\begin{figure}[ht]
\centering
\includegraphics[width=8cm,
  alt={An old labrador retriever close up,
    with intense gaze and mouth slightly open.
    In the background, a younger labrador also stares at the camera.}]
  {pictures/TheDogs.jpg}
\caption{Cira (2009--2022) having a say.}
\label{fig:cira}
\end{figure}
```

It is possible to print a list of figures with the `\listoffigures` command. This command requires one extra run of the compiler in order to be in sync with the document. If your figure caption is very long, you can specify a shorter version for the list in an optional argument:[4]

```
\caption[An old labrador]{Cira (2009--2022) having a say.}
```

[4] Akin to the sectioning commands and table of contents.

Figure 7.1: Cira (2009–2022) having a say.

7.2.1 Subfigures

Floats are just containers for arbitrary LaTeX code, so in principle you are free to put anything in there, even text. This can be abused to create subfigures as in Figure 7.2:

```
\begin{figure}
\centering

\includegraphics[width=0.4\textwidth, alt={The two dogs.}]
  {pictures/TheDogs.jpg}
\hspace{1em}
\includegraphics[angle=180, origin=c,
    width=0.4\textwidth, alt={Same dogs upside down.}]
  {pictures/TheDogs.jpg}

\caption{Left: some dogs. Right: some upside-down dogs.}
\label{fig:hacky subfigs}
\end{figure}
```

However, there is also the subcaption package that does the same automatically and with more style.

Warning
Do not use the subfigure or subfig packages; they are no longer maintained and do not play as well together with hyperref.

This package offers the `\subcaptionbox` that takes 2–5 arguments: optional short caption, the caption, optional width for the subfigure, optional alignment for the subfigure (`l`, `c`, or `r`), and finally the figure contents.

Figure 7.2: Left: some dogs. Right: some upside-down dogs.

Figure 7.3 shows how this looks in practice. Note how the lines of figures need to be broken by a paragraph break (empty line); one could put a vertical spacing command here. For the third subfigure (Figure 7.3c), the width is set manually so that the caption is not broken into two lines. Note that `\label` can be used, but it needs to be inside the caption argument.[5]

```
\begin{figure}
\centering

\subcaptionbox{As usual.}{
  \includegraphics[width=0.4\textwidth]{pictures/TheDogs.jpg}}
\subcaptionbox{Upside down.}{
  \includegraphics[angle=180, origin=c, width=0.4\textwidth]
    {pictures/TheDogs.jpg}}

\subcaptionbox{The smaller one.\label{fig:subcaption small dog}}[3cm]{
  \includegraphics[bb=1cm 2cm 3cm 5cm, clip]
    {pictures/TheDogs.jpg}}
\subcaptionbox{The larger one.}{
  \includegraphics[bb=2.8cm 1.4cm 7cm 6cm, clip, scale=0.8]
    {pictures/TheDogs.jpg}}
\subcaptionbox{Right-aligned and fixed width.}[3.5cm][r]{
  \scshape Lorem ipsum dolor sit amet.}

\caption{A collection of dogs and a typographical example sentence.}
\label{fig:subcaption}
\end{figure}
```

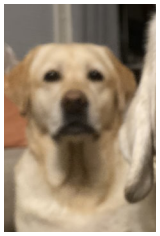
7.2.2 Custom float environments

By default, LaTeX provides the `figure` and `table` environments and the corresponding commands for lists of figures and tables. It is possible to also

[5] To reference the subfigure letter without the main figure number, the package provides a `\subref` command.

(a) As usual.

(b) Upside down.



(c) The smaller one.

(d) The larger one.

LOREM IPSUM DOLOR SIT AMET.

(e) Right-aligned and fixed width.

Figure 7.3: A collection of dogs and a typographical example sentence.

introduce new float environments with the float package. This is useful for e.g. source code listings.

The `\newfloat` command takes 3–4 arguments: the environment name to use, the default placement specifier (remember that the default is usually `tbp`), file name extension for the auxiliary file (must not be already in use!), and optionally the number-within level.

To customize the look of the new float, a `\floatstyle` command can be put before the definition. By default there are four options: `plain`, `plaintop` (caption on top), `boxed`, and `ruled`. The displayed name of the new environment can be set with the `\floatname` command.

For example, a float class for source code listings could be defined as follows. (Note: the listings package also has an option to define a float class, so it is not necessary to do this manually.)

```
\floatstyle{ruled}
\newfloat{sourcecode}{tbp}{lst}[chapter]
\floatname{sourcecode}{Code Listing}
```

It would then produce a float like Code Listing 7.1.

```
\begin{sourcecode}[h]
\begin{lstlisting}[language=Pascal]
program Hello;

begin
writeln("Hello world!");
end.
\end{lstlisting}
\caption{A Hello World program in Pascal.}
\label{src:hello pascal}
\end{sourcecode}
```

Code Listing 7.1 A Hello World program in Pascal.

```
program Hello;

begin
writeln("Hello world!");
end.
```

7.2.3 Landscape floats

If your material is very wide (for instance, a table with many wide columns), it might be beneficial to turn the page sideways.

This is achieved most easily with the rotating package. It defines the `sidewaysfigure` and `sidewaystable` environments that work just like their counterparts, except that they are set on landscape pages.

If you need the same behaviour for custom floats, the rotfloat package bridges rotating and float together.

As of this writing in May 2024, the package does not know how to tell your PDF reader to show the page in rotated mode. This could be circumvented with the method in Section 3.3.1, but the manual addition of landscape pages always introduces a page break at the position.

7.3 Creating tables

The `tabular` environment offered by LaTeX is sufficient for simple tables. It consists of table contents and certain special elements:

- In the beginning, there is a *column specification*. The number of columns is fixed here, and for each column the alignment of text is indicated: `l` for left-aligned, `c` for centered, and `r` for right-aligned. Fixed width can be given with `p{...}`.

Additionally, vertical lines can be specified with `|` and other inter-column material with `@{...}`.

- On each line, columns are separated by `&`.
- Each line is ended with `\\`. If you forget to end a line, you will get an error at the next `&`.
- Horizontal lines can be created with `\hline`. No `\\` is then necessary.

Let us illustrate these with a simple example:

```
\begin{tabular}{l| c p{3cm} c @{ $\Rightarrow$ } r}
Breed & Drops fur & Notes & Rating & Conclusion\\
\hline
Corgi & Somewhat & Has fairly short legs but a lot of attitude
    & 13/10 & Good dog\\
Labrador & Very much & Excitable about other dogs, humans, playing in water
    & 13/10 & Good dog
\end{tabular}
```

Breed	Drops fur	Notes	Rating ⇒ Conclusion
Corgi	Somewhat	Has fairly short legs but a lot of attitude	13/10 ⇒ Good dog
Labrador	Very much	Excitable about other dogs, humans, playing in water	13/10 ⇒ Good dog

The `table` environment acts exactly like its figure counterpart, and should be used for tables where the exact placement is not so important. The following code produces Table 7.1:

```
\begin{table}[ht]
\centering
\begin{tabular}{l|ccc}
  Breed & Drops fur & Tall & Long\\
  \hline
  Corgi & Somewhat & Very not & Quite\\
  Labrador & Very much & Yes & Yes
\end{tabular}
\caption{An extensive comparison of dog breeds.}
\label{tbl:dogs}
\end{table}
```

Breed	Drops fur	Tall	Long
Corgi	Somewhat	Very not	Quite
Labrador	Very much	Yes	Yes

Table 7.1: An extensive comparison of dog breeds.

If you need more customization for your tables, then the array package is the starting point. First off, it adds the following format specifiers:

`m` Fixed-width column like `p`, but centered vertically.

`w` This takes two parameters: alignment and width. There is *no* automatic line breaking, so the contents may overprint adjacent cells!

`W` Like above, but tries to squeeze the text a bit and if still fails, at least raises an overfull hbox warning.

`>` Takes one argument, which is inserted in front of each entry.

`<` Like above, but after the entry.

`!` Like the `@` specifier, but preserves the horizontal space that would surround a vertical line.

The previous example can be customized as follows. We use the `>` and `<` specifiers to set the rating in a bold font and to remove the duplicated `/10` texts. Since `!` preserves the whitespace, we no longer need spaces around the arrow symbol.

```
\begin{tabular}{l| c m{3cm} >{\bfseries}c<{/10} !{$\Rightarrow$} r}
Breed & Drops fur & Notes & Stars & Conclusion\\
\hline
Corgi & Somewhat & Has fairly short legs but a lot of attitude
    & 13 & Good dog\\
\hline
Labrador & Very much & Excitable about other dogs, humans, playing in water
    & 13 & Good dog
\end{tabular}
```

Breed	Drops fur	Notes	**Stars/10**	⇒	Conclusion
Corgi	Somewhat	Has fairly short legs but a lot of attitude	**13/10**	⇒	Good dog
Labrador	Very much	Excitable about other dogs, humans, playing in water	**13/10**	⇒	Good dog

Note that the change from `p` to `m` does not improve legibility here; that's why I added a horizontal line to separate the two rows. There would also be several methods to add vertical whitespace to the table:

- Optional argument to `\\`: however, the value is interpreted as the desired *total height* of the row, and not as a skip. If the value is smaller than the current height, this has no effect.
- The `\extrarowheight` length specified by array is added to the beginning of each row. By default it is `0pt`; see the example on page 148 for a different value.
- The cellspace package can be used to ensure top and bottom vertical clearances when the cell contents are taller than usual characters. This package defines a modifier for cell styles.

The siunitx package also introduces a new format specifier `S`. It aligns numbers at the decimal point. This specifier has a lot of customization options, for which we refer to the package documentation. As an important note, the column is interpreted numerically: textual material might need protection with `{}` (so that 'e' is not interpreted as an exponent specifier).

```
\begin{tabular}{l| S}
Important constant & {Value}\\
\hline
Pi & 3.14159\ldots\\
Gravity & 9.81\\
Inches to feet & 12\\
Finnish population & {some } 5.61e6
\end{tabular}
```

Important constant		Value
Pi		3.141 59…
Gravity		9.81
Inches to feet		12
Finnish population	some	5.61×10^6

7.3.1 Special cells

If you need a cell to span several columns, you can use the `\multicolumn` command. It takes three arguments: the number of columns to extend to, a new column specification for the internal contents, and finally the contents.

```
\setlength\extrarowheight{2pt}
\centering
\begin{tabular}{l| c c @{ $\Rightarrow$ } r}
Breed & Drops fur & Rating & Conclusion\\
\hline
Corgi & Somewhat & 13/10 & Good dog\\
Labrador & Very much & 13/10 & Good dog\\
Roomba & Negatively?! &
    \multicolumn{2}{c}{\emph{We're not quite sure}}
\end{tabular}
```

Breed	Drops fur	Rating ⇒ Conclusion
Corgi	Somewhat	13/10 ⇒ Good dog
Labrador	Very much	13/10 ⇒ Good dog
Roomba	Negatively?!	*We're not quite sure*

In this example, the last two columns (specified `c r`) were replaced by a single `c` cell.

To use colours, one can use the colortbl package. Remember to use colour only moderately and to maintain a good colour contrast. (The following example fails at any artistic merits for illustrative purposes.)

```
\centering
\begin{tabular}{>{\columncolor{green!15}}l | c c @{ $\Rightarrow$ } r}
\rowcolor{blue!15} Breed & Drops fur & Rating & Conclusion\\
\hline
Corgi & Somewhat & 13/10 & Good dog\\
Labrador & Very much & 13/10 & Good dog\\
Roomba & \cellcolor{orange!20} Negatively?! &
    \multicolumn{2}{c}{\emph{We're not quite sure}}
\end{tabular}
```

Breed	Drops fur	Rating	⇒	Conclusion
Corgi	Somewhat	13/10	⇒	Good dog
Labrador	Very much	13/10	⇒	Good dog
Roomba	Negatively?!	*We're not quite sure*		

Here the `\columncolor` command goes into the column specifier, and the `\rowcolor` command goes into the beginning of each row. As you can see, the latter is not fully compatible with custom separators. As each cell forms a scope, you can also use `\color` to set the text color in a cell.

The diagbox makes it possible to split cells diagonally. It has quite a lot of configuration options as optional arguments, but we refer the reader to the package documentation. It also has a somewhat esoretic syntax that accepts *two or three* required arguments, meaning that the two-argument version cannot be followed by a `{`. (This is quite unlikely in a table cell.)

```
\centering
\begin{tabular}{l|ccc}
  \diagbox{Breed}{Property} & Drops fur & Tall & Long\\
  \hline
  Corgi & Somewhat & Not & Quite\\
  Labrador & Very much & Yes & Yes\\
  Roomba & Negatively & Very not & It's round
\end{tabular}
```

Breed \ Property	Drops fur	Tall	Long
Corgi	Somewhat	Not	Quite
Labrador	Very much	Yes	Yes
Roomba	Negatively	Very not	It's round

7.3.2 Footnotes in tables

Finally, let us talk about footnotes in tables, as sometimes mandated by scientific style. As discussed in Section 2.5, `\footnote` commands inside a `tabular` environment do not print the footnote text anywhere.

A simple fix is to place the table inside a `minipage` environment. Often, an easier alternative is to use the threeparttable package that provides its own footnote command. You are required to take care of the numbering, but this is actually useful since commonly the same footnote is used for several locations.

```
\centering
\begin{threeparttable}
\begin{tabular}{l|ccc}
  Breed & Drops fur & Tall & Long\\
  \hline
  Corgi & Somewhat\tnote{a} & Not\tnote{b} & Quite\tnote{b}\\
  Labrador & Very much\tnote{c} & Yes\tnote{b,c} & Yes\tnote{b,c}\\
  Roomba & Negatively\tnote{d} & Very not\tnote{b} & It's round\tnote{b}
\end{tabular}
\begin{tablenotes}[para]
  \item[] Sources:
  \item[a] Googling.
  \item[b] General knowledge.
  \item[c] Having lived with some.
  \item[d] Rumours.
\end{tablenotes}
\caption{Some extensive research into things.}
\end{threeparttable}
```

Breed	Drops fur	Tall	Long
Corgi	Somewhat[a]	Not[b]	Quite[b]
Labrador	Very much[c]	Yes[b,c]	Yes[b,c]
Roomba	Negatively[d]	Very not[b]	It's round[b]

Sources: [a] Googling. [b] General knowledge.
[c] Having lived with some. [d] Rumours.

Table 7.2: Some extensive research into things.

If you ever need to set a table that spans multiple pages, the supertabular and longtable packages are your friends. The first one is simpler but the second is more extensible and more maintained.

7.3.3 Final words

There are several more useful packages for controlling the layout of tabular material. Most mathematicians will probably not need them, but [4] has over 70 pages on this topic if the need arises.

New in LaTeX3
As of this writing in April 2024, PDF tagging of tables has recently entered testing. This means that tables should no longer be completely meaningless jumbles of text and numbers to a screen reader.

The syntax for annotating tables is still being worked out. In a future LaTeX version, there should be syntax for annotating column headers. Follow the news on LaTeX homepage to know what's happening!

Chapter 8

Graphics with TikZ

There are several methods for producing line graphics in LaTeX. We cover here TikZ, which is quite portable across compilers and extremely versatile. It is contained in the tikz package.

There are several extension libraries that are bundled with TikZ and loaded with the `\usetikzlibrary` command. Moreover, many other packages are built with TikZ. One such package is pgfplots, which provides much more tools for plotting data.

TikZ has bit of a learning curve and its documentation spreads over no less than 1321 pages (as of April 2024). We are able to only cover the essentials here. An unofficial web version of the manual can be accessed at `tikz.dev`.

Technical side note
TikZ is an acronym for *TikZ ist kein Zeichenprogramm* – "TikZ is not a drawing program". From this we can learn two things:

- Many LaTeX core developers come from German-speaking Europe.
- That as useful as TikZ is, for complex graphics you probably should use a proper, visual graphics editor.

Of course, the the acronym is not only recursive, but also oxymoronic, since TikZ is a program (written in the TeX language) that outputs graphics…

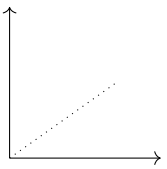
Technical side note
Internally, TikZ consists of three layers: compiler-specific support code for primitive drawing operations, a core layer called PGF, and the (relatively) user-friendly frontend called TikZ.

Due to this, you can see references to PGF scattered in various places; for example the CTAN page for tikz redirects to that of pgf. In practice, the two are synonymous.

8.1 Coordinates, nodes, and paths

Let us begin with a very simple example:

```
\begin{tikzpicture}
  \draw[->] (0,0) -- (2,0);
  \draw[->] (0,0) -- (0,2);
  \draw[dotted] (0,0) -- ({sqrt(2)}, 1);
\end{tikzpicture}
```



TikZ pictures are defined inside a `tikzpicture` environment. (For small pictures, there is also a `\tikz` command that takes the contents of the environment as its single argument.) The output appears inline in text, just as with `\includegraphics` and friends.

Inside this environment, there are drawing commands. Each of these commands *ends with a semicolon*. Forget the `;` and you will get an error message.

The `\draw` command is the basic workhorse of TikZ. The `--` operation draws a line between the two coordinates. Let us first look at the syntax of coordinates before going back to the optional arguments and different operations.

There are several coordinate systems, of which I believe the next four are most common.

xyz As in the example above, coordinates in this system are tuples or triples of numbers separated by commas. These are with respect to unit vectors defined in the environment options. By default, the x and y unit vectors are 1 cm long and along the usual axes. The z axis is skewed towards the reader.

It is possible to use mathematical syntax like `+`, `/`, and `sqrt(...)` to give more complicated coordinate expressions. You might need to wrap the calculation inside `{}`.

canvas The numbers can also have units like `(1cm, 20pt)`. In this case the coordinates are the absolute length away from canvas origin. It is possible to use expressions like `1cm + 2pt`.

These units are still not completely absolute: the canvas can still be scaled with an optional argument.

Warning: While it is technically possible to mix canvas and xyz coordinates inside the tuple, it is ill-advised. Similarly, an expression like `1+2cm` is interpreted as `1pt+2cm`, not as "unit vector plus 1 cm to the right".

xyz polar Polar coordinates are like `(30:2)`, which means: 30 degrees counterclockwise, radius 2 units.

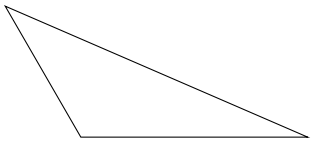
If the x and y unit vectors are set to non-default values, the angle corresponds to a point on the ellipse specified by x and y vectors, and the resulting vector is multiplied by the radius factor.

canvas polar Here the radius has a specified unit: `(30:2cm)`.

There are also the barycentric coordinate system (weighted average of basis vectors) and a possibility to compute tangents of shapes. These are covered in [5, Section 13.2].

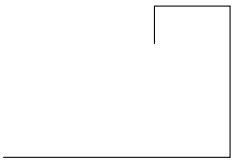
Coordinates can be given names with the `\coordinate` command. In the example below, we also use the `cycle` specifier to close the loop.

```
\begin{tikzpicture}
  \coordinate (A) at (0:3cm);
  \coordinate (B) at (120:2cm);

  \draw (0,0) -- (A) -- (B) -- cycle;
\end{tikzpicture}
```



It is also possible to specify relative coordinates: by prefixing the coordinate with `++`, it is relative to the previous position.[1]

```
\begin{tikzpicture}
  \draw (0,0) -- ++(0:3) -- ++(90:2)
    -- ++(180:1) -- ++(270:1/2);
\end{tikzpicture}
```



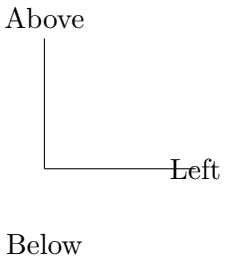
8.1.1 Nodes

Nodes can be thought of as text placed in the picture. The text can contain mathematics and formatting commands, and even pictures.

A node can be placed either as part of a path, or separately with a `\node` command. In the latter case, it is possible to refer to the node later.

[1] It is also possible to prefix with just `+`, in which case the current position is not updated.

```
\begin{tikzpicture}
  \draw (0,0) -- (2,0) node {Left};
  \node (A) at (0,2) {Above};
  \draw (0,0) -- (A);
  \node at (0,-1) {Below};
\end{tikzpicture}
```

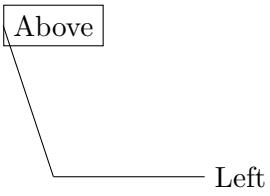


Note the difference between the lines connecting the origin to the Left and Above nodes. When a node is created as part of a path operation, it is *centered* at the specified coordinate. That is, the center of the "Left" text is at $(2, 0)$. The position of the node can be customized by passing `above`, `below`, `left`, or `right` as an optional argument.

On the other hand, when a path is drawn to a previously defined node, TikZ tries to stop at the boundary of the node. It is possible to specify the position on the border by cardinal directions (like `north`) or an angle; or to specify `center` if you really mean the center. These are given by the `name.anchor` syntax as in the example below.

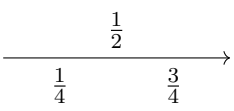
To make the boundary of the "Above" node visible, we also pass the `draw` option to the `\node` command.

```
\begin{tikzpicture}
  \draw (0,0) -- (2,0) node[right] {Left};
  \node[draw] (A) at (0,2) {Above};
  \draw (0,0) -- (A.west);
\end{tikzpicture}
```



One more interesting positioning specifier is the optional `pos` argument. It is understood as a position (in the range 0 to 1) on the previous path segment. It can be combined with other specifiers as in here:

```
\begin{tikzpicture}
  \draw[->] (0,0) -- (3,0)
    node[pos=0.5,above] {$\frac 1 2$}
    node[pos=0.25,below] {$\frac 1 4$}
    node[pos=0.75,below] {$\frac 3 4$};
\end{tikzpicture}
```



To embed pictures, you can use `\includegraphics` as usual inside a node. To illustrate this once more with our fur-shedding assistants:

```
\newcommand{\dogfile}{pictures/TheDogs.jpg}
\begin{tikzpicture}
  \node[draw] (Both) at (0,0)
    {\includegraphics[height=3cm]{\dogfile}};
  \node[draw] (Netta) at (-5,0)
    {\includegraphics[bb=1cm 2cm 3cm 5cm, clip, height=2cm]{\dogfile}};
  \node[draw] (Cira) at (5,0)
    {\includegraphics[bb=2.8cm 1.4cm 7cm 6cm, clip, height=2cm]{\dogfile}};
  \draw[->] (Both.west) -- (Netta.east) node[pos=0.5, above] {Netta};
  \draw[->] (Both.east) -- (Cira.west) node[pos=0.5, above] {Cira};
\end{tikzpicture}
```

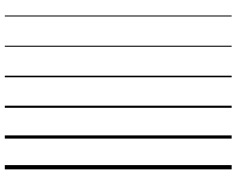
8.1.2 Path options

There are lots of options that can be passed to the drawing commands. As the very first one, let us consider `color`. It takes a colour name as its argument; xcolor extensions and custom colours are supported. It is possible to blend the color with white with the `!` specifier (see page 45).

```
\definecolor{Sciency}{cmyk}{0,0.46,1,0}
\begin{tikzpicture}
  \draw[color=blue] (0,0)--++(3,0);
  \draw[color=LimeGreen] (0,-0.5)--++(3,0);
  \draw[color=Sciency] (0,-1)--++(3,0);
  \draw[color=Sciency!60] (0,-1.5)--++(3,0);
\end{tikzpicture}
```

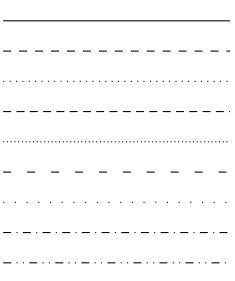
There is also a range of line weights. Do note that their names may include spaces. Of these, `thin` is the default.

```
\begin{tikzpicture}[yscale=0.8]
  \draw[very thin] (0,-0.5)--++(3,0);
  \draw[thin] (0,-1)--++(3,0);
  \draw[semithick] (0,-1.5)--++(3,0);
  \draw[thick] (0,-2)--++(3,0);
  \draw[very thick] (0,-2.5)--++(3,0);
  \draw[ultra thick] (0,-3)--++(3,0);
\end{tikzpicture}
```



Then, there are the line styles. The basic ones are (it is possible to specify custom ones, but that you have to read from the docs):

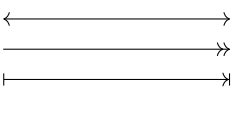
```
\begin{tikzpicture}[yscale=0.8]
  \draw[solid] (0,0.5)--++(3,0);
  \draw[dashed] (0,-0)--++(3,0);
  \draw[dotted] (0,-0.5)--++(3,0);
  \draw[densely dashed] (0,-1)--++(3,0);
  \draw[densely dotted] (0,-1.5)--++(3,0);
  \draw[loosely dashed] (0,-2)--++(3,0);
  \draw[loosely dotted] (0,-2.5)--++(3,0);
  \draw[dash dot] (0,-3)--++(3,0);
  \draw[dash dot dot] (0,-3.5)--++(3,0);
\end{tikzpicture}
```



Good practices
As remarked in the section on colour, you should never use colour as the sole means of giving information. In simple line graphics, the combination of colour and line style is often useful. However, complex line styles are also hard to read!

Finally, there are the arrows. The syntax for arrow tips is ⟨*start*⟩`-`⟨*end*⟩, where the start/end specification can contain one or more `<`, `>`, or `|`.

```
\begin{tikzpicture}[yscale=0.8]
  \draw[<->] (0,-0)--++(3,0);
  \draw[->>] (0,-0.5)--++(3,0);
  \draw[|->|] (0,-1.0)--++(3,0);
\end{tikzpicture}
```

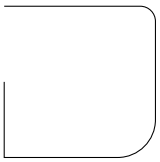


The `arrows.meta` extension library contains many more tip styles and options to customize their size; see [5, Section 16].

```
% \usetikzlibrary{arrows.meta}
\begin{tikzpicture}[yscale=0.8]
  \draw[Circle-{Circle[open]}] (0,-0)--++(3,0);
  \draw[-{Stealth[length=5mm,width=2mm,red]}] (4,0)--++(3,0);
  \draw[{->[harpoon]}] (8,0)--++(3,0);
\end{tikzpicture}
```

One more option: the `rounded corners` option does what is says. It takes as its argument a radius of the corner. It is one of the few options that can be changed in the middle of a path (none of the above can, unfortunately).

```
\begin{tikzpicture}
  \draw[rounded corners=2mm] (0,0) -- (2,0)
    [rounded corners=5mm] -- (2,-2)
    [sharp corners] -- (0,-2) -- (0,-1);
\end{tikzpicture}
```



8.1.3 *Absolute positioning**

In some very special cases it might be useful to position graphics at absolute coordinates on the page. This can be done by specifying coordinates relative to the `current page` anchor, and specifying two optional arguments to the environment:

`remember picture` tells TikZ to remember the position on the page between LaTeX runs. You might need to compile twice.

`overlay` tells TikZ to ignore bounding box computations. This ensures that the picture does not interact with the rest of the page contents.

For example, here we start from the right edge of the page, move a bit upwards and to the left, and then fill a small circle:

```
\begin{tikzpicture}[remember picture, overlay]
  \fill[blue] (current page.east) ++(-1cm, 2cm) circle [radius=0.2cm];
\end{tikzpicture}
```

See [5, Section 17.3.2] for the full details.

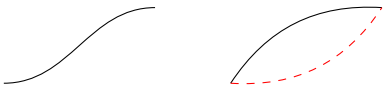
8.2 Special paths

In addition to the `--` style path joining two points, there are the `|-` and `-|` styles. These split the line into vertical and horizontal segments, the order of which you can probably guess.

```
\begin{tikzpicture}
  \draw (0,0) -| (2,1);
\end{tikzpicture}
```

To draw a curved path between two points, you can use the `to` operation with optional arguments. The `in` and `out` arguments take an angle in degrees (clockwise). There are also the `bend left` and `bend right` options for automatic control.

```
\begin{tikzpicture}
  \draw (0,0) to[out=0,in=180] (2,1);
  \draw (3,0) to[bend left] (5,1);
  \draw[red,dashed] (3,0) to[bend right] (5,1);
\end{tikzpicture}
```

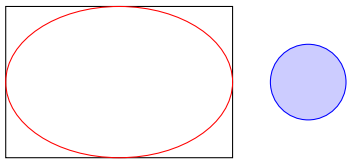


To draw rectangles, there is the `rectangle` shorthand that is put between coordinates of the two opposite corners. Similarly, there is the `circle` command that is put after the centre coordinate and takes the radius as an optional parameter. It is possible to specify `x radius` and `y radius` separately to get an ellipse (which can be further rotated with the `rotate` parameter).

Moreover, the command `\fill` does exactly what you would expect it to; it fills the enclosed region. The combination `\filldraw` both fills and draws. If you have defined the path through line segments, the path is automatically closed.[2]

[2] If the lines cross, the definition of interior can be customized with optional arguments `nonzero rule` and `even odd rule`. These are defined in [5, Section 15.5.2].

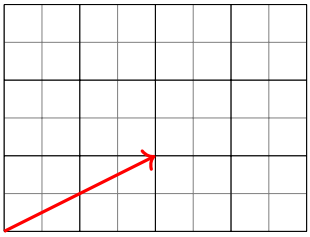
```
\begin{tikzpicture}
  \draw (0,0) rectangle (3,2);
  \draw[red] (3/2,1) circle
    [x radius=3/2, y radius=1];
  \filldraw[blue, fill=blue!20] (4,1) circle [radius=0.5];
\end{tikzpicture}
```



There are also commands for drawing arcs, parabolas, and even Bézier curves; see [5, Section 14].

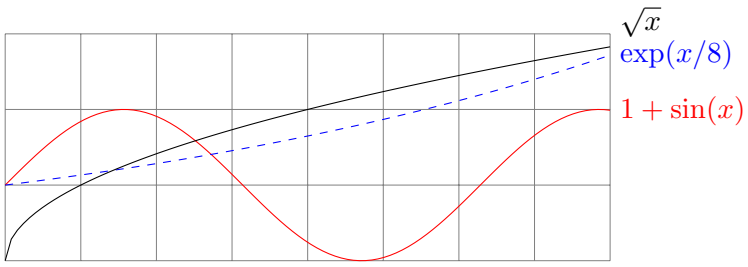
There is a helper command for drawing coordinate grids. TikZ provides a shorthand style `help lines` that can be customized as in Section 8.5. Here we overlay two grids with different step sizes (default is one unit).

```
\begin{tikzpicture}
  \draw[help lines, xstep=0.5, ystep=0.5] (0,0) grid (4,3);
  \draw (0,0) grid (4,3);
  \draw[red,very thick,->] (0,0) -- (2,1);
\end{tikzpicture}
```



Now that we know how to draw a coordinate grid, it is very natural to draw some functions on it! TikZ includes a reasonably complex calculation engine, which makes it possible to draw some complicated functions.

```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (8,3);
  \draw[red, domain=0:8, samples=100] plot (\x, {1+sin(\x r)})
    node[right] {$1+\sin(x)$};
  \draw[blue, dashed, domain=0:8, samples=100] plot (\x, {exp(\x/8)})
    node[right] {$\exp(x/8)$};
  \draw[black, domain=0:8, samples=100] plot (\x, {pow(\x, 1/2)})
    node[above right] {$\sqrt x$};
\end{tikzpicture}
```

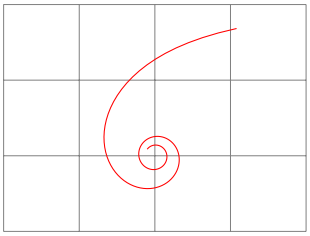


Some notes on the usage:

- The domain is specified using the syntax `lower:upper`.
- The number of samples is optional to specify, but the default value is quite small.
- The variable x is written with a backslash `\x`, but the mathematical commands are spelled without a backslash.
- Mathematical expressions should be wrapped inside `{}`.
- Powers like x^2 can be specified with `pow(\x, 2)`.
- Trigonometric functions assume degrees by default; the suffix `r` after `\x` converts the variable into radians.

Note that the plot is still defined as a pair of (x, y) coordinates. This makes it possible to draw parametric plots. The name of the variable can be customized.

```
\begin{tikzpicture}
  \draw[help lines] (-2,-1) grid (2,2);
  \draw[red, domain=1:15, samples=150, variable=\t]
    plot ({2*cos(\t r) / \t}, {2*sin(\t r) / \t});
\end{tikzpicture}
```



It is also possible to plot individual points or bar graphs, or to even load a dataset from an external file. See [5, Section 22] for more, but also note the warnings below.

Warning
"Division by zero" and "number too big" errors can sometimes lead to confusing (and repeated!) error messages.

Good practices
TikZ is excellent for producing small and simple function plots in a style consistent with the rest of the document. However, compilation times can get quite long for even slightly complicated functions.

If you need to plot lots of complex functions, you should precompute the function values in a file and use the data plotting facility of TikZ; see [5, Section 22].

8.3 Transformations

The `xshift`, `yshift`, `rotate`, and `scale` optional arguments can be used to locally change the coordinate system.[3] You can use a `scope` environment within the picture to pass the transformation to multiple drawing operations:

```
\begin{tikzpicture}
  \draw (0,0) rectangle (2,1);
  \begin{scope}[xshift=3cm]
    \draw[dotted] (0,0) rectangle (2,1);
  \end{scope}
  \begin{scope}[xshift=6.2cm, scale=0.5, rotate=30]
    \draw[blue] (0,0) rectangle (2,1);
  \end{scope}
\end{tikzpicture}
```

[3] There are some more; see [5, Section 25.3] for details. A particular example is `rotate around`, which rotates around a specified point.

Note that the `xshift` parameter should be specified with a unit (otherwise the unit is assumed to be `pt`). The `scope` environment can be used to pass any optional arguments to enclosed commands; for example, color or line style declarations can be passed as well.

Sometimes it is necessary to extend or to shrink the bounding box of the picture. For this, the `\useasboundingbox` command is useful.[4] (This is especially useful in a Beamer slide where you gradually reveal parts of the picture. You can use this command to reserve enough space for the full picture.)

Here we instead shrink the bounding box so that the picture contents extend beyond the reserved space. The dramatic effect is somewhat ruined by displaying the bounding box.

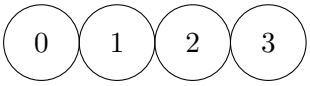
```
Then a huge seagull
\begin{tikzpicture}
  \useasboundingbox[draw,dotted,gray] (-1em,-0.7em) rectangle (1em,0.8em);

  \draw (0,0) circle [radius=0.7em];
  \fill[orange] (-0.3em,-0.1em)--(0.3em,-0.1em)--(0em,-1em)--cycle;
  \draw (0.7em,0em) parabola bend (5em,1em) (6em,0.8em);
  \draw (-0.7em,0em) parabola bend (-5em,1em) (-6em,0.8em);
\end{tikzpicture}
came swooping down.
```

8.4 Loops

TikZ automatically loads the pgffor package that provides a useful `\foreach` command. It can be used in two ways. The first is to loop over a given set:

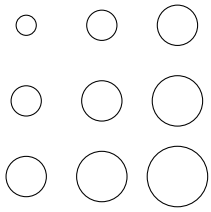
```
\begin{tikzpicture}
  \foreach \x in {0,1,2,3}
    \draw (\x, 0) circle [radius=0.5] node {$\x$};
\end{tikzpicture}
```

[4] Let us note that this is a shorthand for `\path[use as bounding box]`; TikZ certainly has many ways to do common operations.



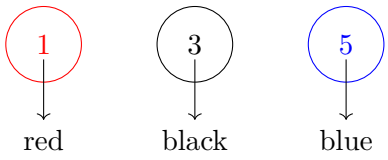
Loops can be nested:

```
\begin{tikzpicture}
  \foreach \x in {1,2,3}
    \foreach \y in {1,2,3}
      \draw (\x, -\y) circle [radius={(\x+\y)/15}];
\end{tikzpicture}
```



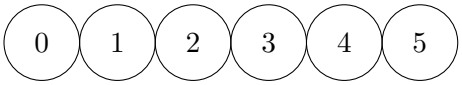
With `{}`, it is possible to define several commands to be executed. It is also possible to define several variables in one loop:

```
\begin{tikzpicture}
  \foreach \x/\col in {1/red, 3/black, 5/blue} {
    \draw[\col] (\x, 0) circle [radius=0.5] node {$\x$};
    \draw[->] (\x,-0.2) -- (\x,-1) node[below] {\col};
  }
\end{tikzpicture}
```



Another way to use `\foreach` is to specify a range. It takes the first, second, and last element, and `...` before the final element.

```
\begin{tikzpicture}
  \foreach \x in {0,1,...,5}
    \draw (\x, 0) circle [radius=0.5] node {$\x$};
\end{tikzpicture}
```



Finally, let us note that the `\foreach` command can freely be used outside TikZ. (If you do not need full TikZ, you can just load the pgffor package.)

```
\foreach \dog/\yr in {Cira/2009, Netta/2016}
  {\dog{} was born in \yr{}.\\}
```

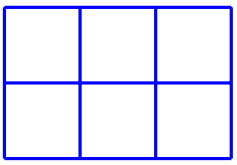
Cira was born in 2009.
Netta was born in 2016.

8.5 *Setting styles**

In the grid example, we saw the `help lines` style that produced thin, gray lines. TikZ provides a general facility for setting up such shorthand styles. If you find yourself repeating graphical options, you should consider using a style instead.[5]

To customize the style, you use the `/.style={}` syntax in the optional argument to the picture environment:

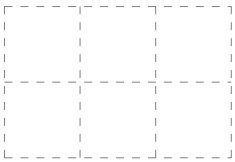
```
\begin{tikzpicture}[help lines/.style={very thick,blue}]
  \draw[help lines] (0,0) grid (3,2);
\end{tikzpicture}
```



If you only want to customize part of the style, you can keep the previous elements in place by using `/.append style={}` instead. Here we keep the lines thin and gray, but also make them dashed:

```
\begin{tikzpicture}[help lines/.append style={dashed}]
  \draw[help lines] (0,0) grid (3,2);
\end{tikzpicture}
```

[5] Those familiar with web development notice that this system is similar to Cascading Style Sheets (CSS); indeed, the TikZ style system is inspired by it.



You can define your own styles with the `/.style` syntax. To reuse them across multiple TikZ environments, you can put the definition in a `\tikzset` command. Such a command could go into the preamble. It is still possible to override styles locally. Here we define a style called `highlight`:

```
\tikzset{highlight/.style={draw,very thick,orange}}
\newcommand{\dogfile}{pictures/TheDogs.jpg}

\begin{tikzpicture}
  \node[highlight] at (-2,0)
    {\includegraphics[bb=1cm 2cm 3cm 5cm, clip, height=2cm]{\dogfile}};
  \node[highlight, blue] at (2,0)
    {\includegraphics[bb=2.8cm 1.4cm 7cm 6cm, clip, height=2cm]{\dogfile}};
\end{tikzpicture}
```

Good practices
Do note that this idea goes very well with the separation of content and presentation. If your document contains a lot of TikZ graphics with repeated graphical styles, you should use styles with descriptive names. Not only is the code semantically meaningful, but also easier to change.

8.6 Some extensions

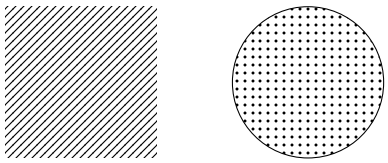
TikZ comes with several extension libraries that are not loaded by default. They can be enabled with `\usetikzlibrary`. Their full documentation fills [5, Part V], but let us briefly show a couple illustrative examples.

Good practices
TikZ has a lot of features, *aber es ist kein Zeichenprogramm*. Some of the features are total overkill to do as part of TEX compilation. Also remember that simplicity is often better in scientific illustration!

8.6.1 Pattern fills

The `patterns` extension library allows filling shapes with a pattern. In some cases it might be a visually appealing alternative to colour.

```
% In the preamble:
% \usetikzlibrary{patterns}

\begin{tikzpicture}
\fill[pattern=north east lines] (-1,-1) rectangle (1,1);
\fill[pattern=dots, draw] (3,0) circle[radius=1];
\end{tikzpicture}
```

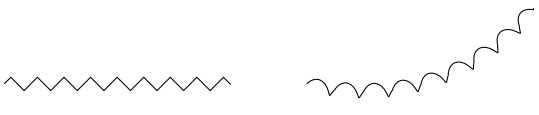


8.6.2 Path decorations

If straight and curved lines are not enough for you, it is possible to add some visual extravaganza. There are a couple basic types of decorations (and quite a few less basic ones – see the documentation if you dare), separated into their own libraries. A useful overview is at [5, Section 24].

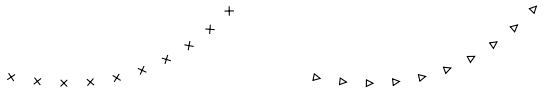
First, the decoration can change the shape of the path. Some of the decoration types can be further customized with parameters (see the documentation):

```
% \usetikzlibrary{decorations.pathmorphing}

\begin{tikzpicture}
\draw[decorate,decoration=zigzag] (0,0)--++(3,0);
\draw[decorate,decoration={coil,segment length=0.38cm}]
  (4,0) to[bend right] ++(3,1);
\end{tikzpicture}
```



Second, the decoration can replace the path with symbols. The easiest way is the following. Note how the markings rotate with the path:

```
% \usetikzlibrary{decorations.shapes}

\begin{tikzpicture}
\draw[decorate,decoration=crosses] (0,0) to[bend right] ++(3,1);
\draw[decorate,decoration=triangles] (4,0) to[bend right] ++(3,1);
\end{tikzpicture}
```



This replaces the whole path. To add markings on top of a path, you can either draw the path and markings separately, or use the more powerful marking extension.

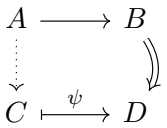
The super-generic marking library already has quite a lot of syntax, which can be referenced from [5, Section 50.6]. The important part is that the `postaction={decorate}` argument preserves the path instead of replacing it.

```
% \usetikzlibrary{decorations.markings}

\begin{tikzpicture}
\draw[postaction={decorate}, decoration={markings,
        mark=between positions 0.1 and 1.0 step 4mm with {\arrow{stealth}}
        }] (4,0) to[bend right] ++(3,1);
\end{tikzpicture}
```

8.6.3 Commutative diagrams

In certain fields of mathematics, everything is explainable with sufficiently many arrows. If you are one of those people, then you will love the tikz-cd package. It can also be loaded through `\usetikzlibrary{cd}`.

```
% \usetikzlibrary{babel} % Might be needed, see below
% \usetikzlibrary{cd}

\begin{tikzcd}
A \arrow[r] \arrow[d, dotted] & B \arrow[d, Rightarrow, bend left]\\
C \arrow[r, maps to, "\psi"]  & D
\end{tikzcd}
```



Gotcha!
If you get an error message when you try to add a label (like `"\psi"` above), you should try loading the `babel` TikZ library. This relates to the handling of quotation marks by babel package; the TikZ support library fixes up the handling inside TikZ environments.

There is a nice online editor for commutative diagrams by Yichuan Shen.[6] It only supports a subset of tikz-cd features, but is a nice and visual way to edit the diagram before exporting its LaTeX code. (It also supports *importing* code for further editing!)

[6] `tikzcd.yichuanshen.de/`

Chapter 9

Presentations with Beamer

If you ask me, Beamer is an odd tool. Presentations are inherently visual, so a programming tool like LaTeX seems ill-suited for the job. On the other hand, most graphical programs cannot match the quality of LaTeX mathematics, and technical talks often have modest visual needs.

Beamer does strike a nice balance between the two, if only your talk is not too ambitious with the visuals. At the same time I find its complexity to be similar to TikZ – a lot of things are possible, but not all of them are advisable.

Good practices
Even though this course is not about presentation skills, I want to use this opportunity for an important reminder: Beamer makes it very easy to throw too much mathematics at your audience.

Leave whitespace on your slides, and focus on pictures to support your point. If you want to dive deep into a mathematical topic, there is an even superior tool: blackboard.

9.1 Basic structure

Beamer is implemented as the **beamer** document class. A minimal Beamer file is thus:

```
\documentclass{beamer}

\begin{document}

\begin{frame}
\end{frame}

\end{document}
```

Beamer automatically loads hyperref. It also loads amsthm and creates some theorem environments by default.

Since TEX always believes it is typesetting for printing, Beamer modifies the page size to be suitable. By default it is 128×96 mm, which corresponds to 4:3 aspect ratio.

To modify the aspect ratio, you can pass the `aspectratio` class option. It is a string of two to four digits, and interpreted like: `aspectratio=32` means 3:2, `169` means 16:9, and `1610` means 16:10.

Good practices
Many computer screens have 16:9 or 16:10 aspect ratio, so 4:3 might seem old-fashioned. However, 4:3 has the benefit of increased vertical space. Many projectors can output either aspect ratio comfortably. If possible, check the venue beforehand.

The default 11 pt font fits very nicely with the simulated paper size. You can pass the `10pt` or `12pt` class options to modify the font size a bit.[1]

Gotcha!
You should put `\usefonttheme{professionalfonts}` in the preamble if you use LuaTeX or XeTeX. It appears that Beamer does not always recognize that Latin Modern is in use with these compilers. If the font theme is not changed, Beamer might produce bad kerning for some specific letter combinations.

For basic title design, you can use the usual `\title`, `\author`, and `\date` commands. There is also an `\institute` command for specifying the author institution, and a `\titlegraphic` command that can be used for a logo etc. Thus a basic presentation with just a title page is set with:

```
\title{My fantastic presentation}
\author{Firstname Lastname}
\institute{University of Puuhamaa}
\titlegraphic{\includegraphics[width=3cm]{TheDogs.jpg}}
\date{20 May 2024}

\begin{document}

\maketitle

\end{document}
```

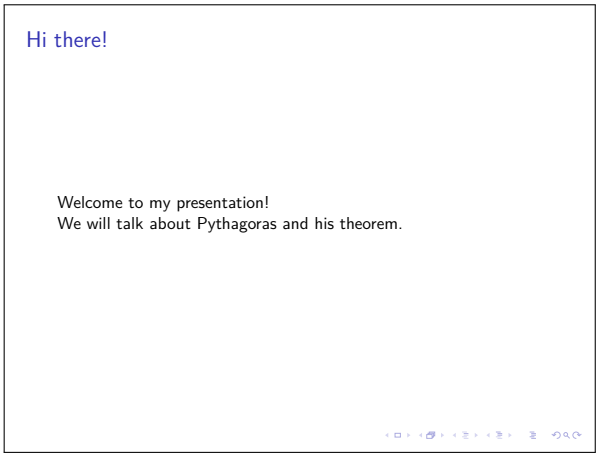
[1] There are also some more choices, but they make the text either very large of very unreadable.

(The `\maketitle` command can be inside or outside a `frame` environment. The command is synonymous with `\titlepage`.)

Technical side note
If you want to customize the look of the title page, there are basically two options: One is to modify the template used by `\maketitle` (see page 187). The easier one is to just set the title manually on an usual frame. You can use `\inserttitle` and friends to insert the contents specified in `\title` command and friends.

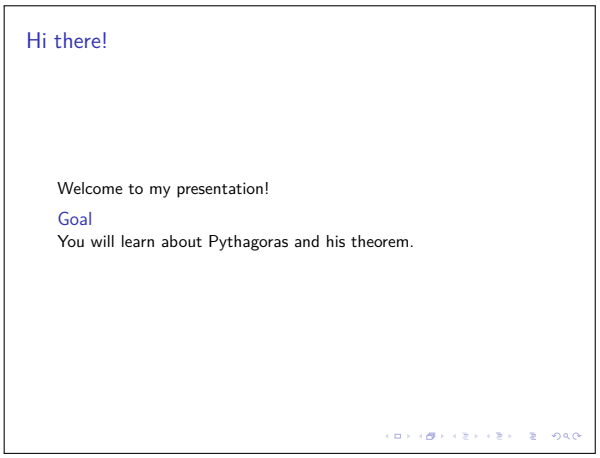
New slides are created with the `frame` environments. Each slide acts like a page, so the basic typesetting commands are available. The title for the frame can be set with `\frametitle`; it is displayed in the frame header.

```
\begin{frame}
\frametitle{Hi there!}

Welcome to my presentation!

We will talk about Pythagoras and his theorem.
\end{frame}
```



Note that the paragraphs have no visual separation whatsoever. You can of course set `\parskip` to a better value (I suggest a larger value than for printed documents, like 1 em), but Beamer presentations are often composed of *blocks*. The `block` environment has a title and content:

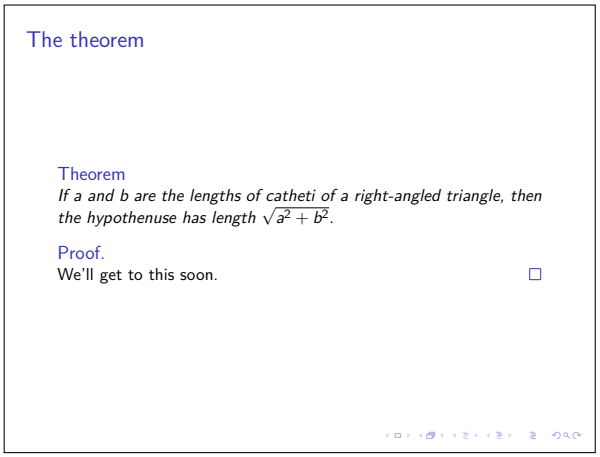
```
\begin{frame}
\frametitle{Hi there!}

Welcome to my presentation!

\begin{block}{Goal}
You will learn about Pythagoras and his theorem.
\end{block}
\end{frame}
```



In this example the block is not very highlighted, but that can be changed with a suitable colour theme. We'll get to that in Section 9.5.

By default, Beamer creates a few common theorem environments. More can be created with the usual amsthm commands (see Section 4.6.1).

```
\begin{frame}
\frametitle{The theorem}

\begin{theorem}
If $a$ and $b$ are the lengths of catheti of a right-angled triangle,
then the hypothenuse has length $\sqrt{a^2 + b^2}$.
\end{theorem}
\begin{proof}
We'll get to this soon.
\end{proof}

\end{frame}
```



To highlight text, it is advisable to use the `\alert` command. By default it makes the text red, but the style can be customized.

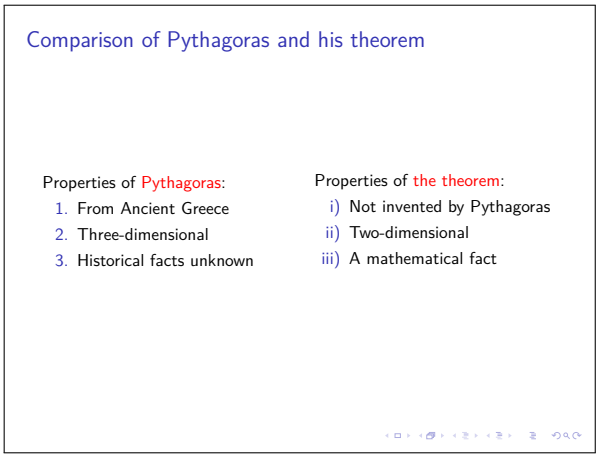
The usual list environments can be used. Beamer loads the enumerate package that enables a shorthand styling syntax for `enumerate` environments.

Let us illustrate these elements with a multi-column layout. Such a layout is created with a `columns` environment. It takes an optional vertical alignment specifier (top by default; vertical centres with `c` and bottoms by `b`).[2] Inside this environment, columns are created with the `column` environment, which takes column width as a required argument.[3]

[2] If things do not align as you would expect, you should also try the `T` option: it aligns baselines of the first lines instead of tops of lines.

[3] There is also an equivalently named command if you prefer to avoid deeply nested environments.

```
\begin{columns}
\begin{column}{0.5\textwidth}

Properties of \alert{Pythagoras}:
\begin{enumerate}[1.]
    \item From Ancient Greece ...
\end{enumerate}
\end{column}

\begin{column}{0.5\textwidth}
Properties of \alert{the theorem}:
\begin{enumerate}[i)]
    \item Not invented by Pythagoras ...
\end{enumerate}

\end{column}
\end{columns}
```

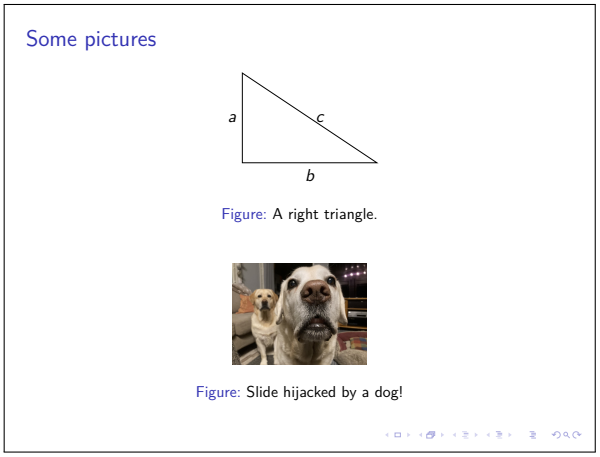


Gotcha!
Frames do not really support the `verbatim` environment; the listings package is probably visually nicer anyways.

9.2 Including graphics

The usual graphics commands of LaTeX and TikZ can be used as usual. Moreover, the usual `figure` and `table` environments can be used.

```
\begin{figure}
\begin{tikzpicture}
    \draw (0,0) -- (0,2) node[pos=0.5, left] {$a$}
        -- (3,0) node[pos=0.5, right] {$c$}
        -- cycle node[pos=0.5, below] {$b$};
\end{tikzpicture}
\caption{A right triangle.}
\end{figure}

\begin{figure}
    \includegraphics[width=3cm]{TheDogs.jpg}
\caption{Slide hijacked by a dog!}
\end{figure}
```



Gotcha!
As with article classes, figures are not placed side by side. The subcaption package is again useful for that; see Section 7.2.1. Moreover, there is no automatic page breaking: if you put too much stuff on a slide, they will overflow.

If you need to position graphics absolutely, you can use the absolute positioning system of TikZ (Section 8.1.3). For instance, the following code could be used as the basis of a slide header:

```
\begin{tikzpicture}[remember picture,overlay]
\node[yshift=0.15\paperheight] at (current page.center){%
\includegraphics[width=\paperwidth]{slideheader.jpg}};
\end{tikzpicture}
```

This assumes the picture to be wide but narrow. The code needs to be compiled twice before anything is visible.

Warning
You can find code examples and Beamer documentation about embedding video files inside a presentation. This is a great way to make a presentation work only on a specific Acrobat Reader version and only on your current computer. It is 99 % guaranteed not to work at a conference venue.

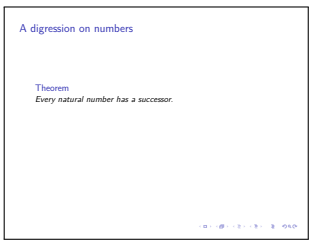
As annoying as it is to switch between programs to show videos, it is the only reliable option.

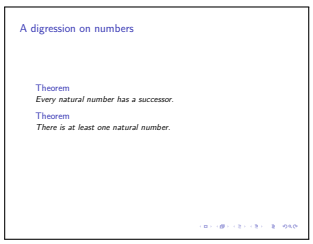
Technical side note
As with embedded videos, it is possible to specify slide transition effects, but they might not work with all PDF viewers. Because of this, we do not discuss them here.

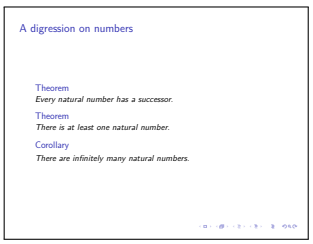
9.3 Uncovering things

The easiest way to uncover things piecewise is the `\pause` command. As the name suggests, it inserts a slide break at the source location.

```
\begin{frame}
\frametitle{A digression on numbers}

\begin{theorem}
Every natural number has a successor.
\end{theorem}

\pause

\begin{theorem}
There is at least one natural number.
\end{theorem}

\pause

\begin{corollary}
There are infinitely many natural numbers.
\end{corollary}
\end{frame}
```







The `\pause` command can be used in many places: between blocks and paragraphs and inside lists.

Gotcha!
The `\pause` command does not work inside amsmath environments.

To get more control, it is possible to use the `\uncover` command and *overlay specifications*. Here are some example overlay specifications:

`<2>` Only shown on slide 2.

`<2->` Shown on slide 2 and all following slides.

`<2-3,5>` Shown on slides 2–3 and 5.

The block environments support overlay specifications, so the previous example could be rewritten as:

```
\begin{frame}
\frametitle{A digression on numbers, version 2}

\begin{theorem}<1->
Every natural number has a successor.
\end{theorem}

\begin{theorem}<2->
There is at least one natural number.
\end{theorem}

\begin{corollary}<3->
There are infinitely many natural numbers.
\end{corollary}
\end{frame}
```

There is also a shorthand syntax `<+->` that is functionally similar to `\pause`: it increments the previous overlay specification by one.[4] In the previous example, all overlay specifications could be replaced by `<+->`. This is particularly useful for lists:

```
\begin{itemize}
    \item<+-> This is shown starting from the first slide.
    \item<+-> This is shown starting from the second slide.
    \item<+-> This is shown starting from the third slide.
\end{itemize}
```

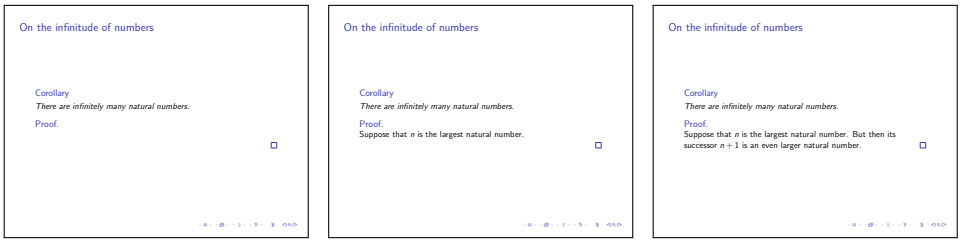
[4] To be precise, the magic command is `+`: it is replaced by the previous slide number, and the slide number is then incremented.

Beamer introduces an optional argument to produce such lists with even less repetition:

```
\begin{itemize}[<+->]
    \item This is shown starting from the first slide.
    \item This is shown starting from the second slide.
    \item This is shown starting from the third slide.
\end{itemize}
```

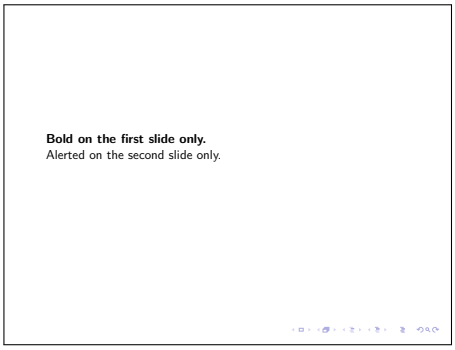
Outside an environment or list, the `\uncover` command works the same:

```
\begin{frame}
\frametitle{On the infinitude of numbers}

\begin{corollary}
There are infinitely many natural numbers.
\end{corollary}
\begin{proof}
\uncover<2->{Suppose that $n$ is the largest natural number.}
\uncover<3->{But then its successor $n+1$ is an even larger natural number.}
\end{proof}

\end{frame}
```



Overlay specifications can also be applied to text styling commands like `\alert`. In this case, the text styling is only applied on the specified frames.

```
\begin{frame}

\textbf<1>{Bold on the first slide only.}\\
\alert<2>{Alerted on the second slide only.}

\end{frame}
```



Lists have a special shorthand syntax for setting an element in alerted style. An overlay specification like `<alert@2>` will make the item alerted on the second slide only. This can be combined with the `+` syntax to simplify matters:

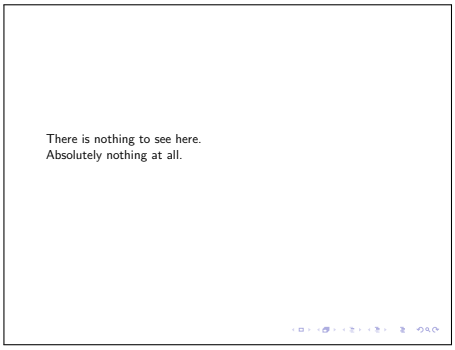
```
\begin{itemize} % Or put [<alert@+>] here
    \item<alert@+> This is highlighted on the first slide.
    \item<alert@+> This is highlighted on the second slide.
    \item<alert@+> This is highlighted on the third slide.
\end{itemize}
```

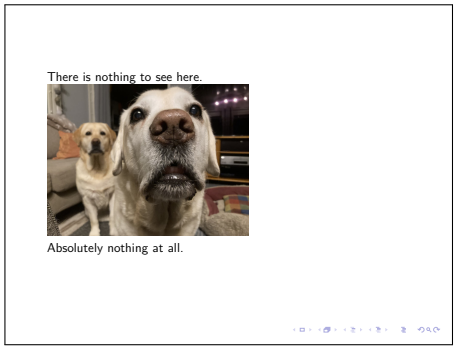
If necessary, you can combine uncover and alert specifications with syntax like `<1- | alert@2>`, or the monstrous `<+-|alert@+>` (which alerts on the same slide as where the item appears).

Good practices
The Beamer manual advises against gradually revealing lists, so making them all visible from the start, but alerting each item in turn might be a better option.

There is also an `\only` command. Its difference to the `\uncover` command is that no space is reserved. Since it may cause the positioning of the adjacent text change between slides, it should only be used when necessary. (This example also has some extra syntax described below.)

```
\begin{frame}<1-2>[label=surprise-dogs]

There is nothing to see here.\\
\only<2>{\includegraphics[width=6cm]{../pictures/TheDogs.jpg}\\}
Absolutely nothing at all.\\
\only<3>{\Large What was that?!}

\end{frame}
```





It is possible to repeat a frame without duplicating the code. In the previous example, we set a label for the frame. The `\againframe` command can be used to repeat a frame, even if there were other frames in between. This mechanism also works together with overlay specifications: the above example only shows slides 1–2, and the next code shows the final slide:

```
\againframe<3>{surprise-dogs}
```

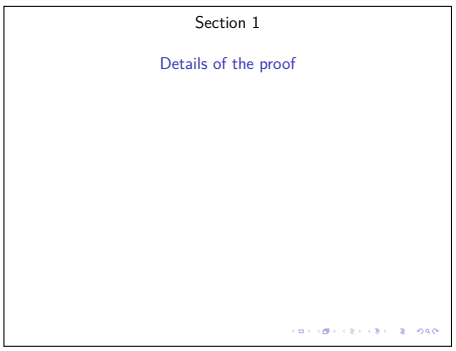
Remark
You can also uncover parts of a TiKZ picture gradually. By default TikZ reserves space only for the visible elements; to avoid elements moving around you can use the `\useasboundingbox` command we met on page 163.

9.4 Sections and parts

The `\section` command can be used to organize your presentation into sections. In some themes (see Section 9.5), the current section – and even progress within the section – is displayed in the header. As usual, a shorter section name can be passed in an optional argument.

The `\sectionpage` command can be used to set a "title page" for the section. (The default theme used in this example does not set it very beautifully.)

```
\section{Details of the proof}

\sectionpage
```



However, it might be more useful to create a frame that shows the new section highlighted in the table of contents. This can be achieved with

```
\begin{frame}
    \tableofcontents[currentsection]
\end{frame}
```

Without the optional argument, an ordinary table of contents is printed.

It is possible to put "backup" slides in an appendix. Any sections and frames following `\appendix` will not be shown in navigation bars in the header.

There are also similar subsection-level commands, if you really need that level of granularity. In the other end of the granularity spectrum, the `\part` command can also be used. Each part is essentially its own isolated presentation – the table of contents only shows the current part, and so on.

You can create a bibliography with the manual method described in Section 5.2.1. The reason *not* to use BibTeX is that the bibliography might need to be broken across several frames. (You can use the `.bbl` file generated by BibTeX as a starting point, of course.)

Good practices
Please think of your audience. Throwing seventeen references at them in the last 20 seconds of your talk is beyond useless. Consider using e.g. footnotes at the point of citation instead.

9.5 Styling it

Like the rest of LaTeX, Beamer attempts to separate content and presentation. It does so via an extensive styling system. The appearance of presentation is controlled with commands in the preamble.

First off, controlling the navigation symbols. There are several styles for them, but I believe you only need one – the one where they are not shown:[5]

```
\beamertemplatenavigationsymbolsempty
```

Remark
As mentioned above, if you use LuaLaTeX or XeLaTeX, you should usually also specify

```
\usefonttheme{professionalfonts}
```

to avoid small kerning issues with some characters.

Some people prefer to show "covered" text in slightly transparent style, so that the audience sees that it is there. This can be controlled with the `\setbeamercovered` command. It takes one argument, and the common values for it are:

`invisible` The default: covered text is invisible.

`transparent` Covered text is partially visible. This is in fact equivalent to `transparent=15`, where the percentage can be tweaked.

`dynamic` The transparency effect depends on the number of slides until the piece is uncovered. Use with extreme caution, might be distracting.

9.5.1 Themes

Beamer comes with a complex theming system for changing the appearance of things. Essentially, you should only need to modify the preamble to change how things look.

The overall appearance is determined by the *presentation theme*. It is chosen with `\usetheme` in the preamble. The presentation theme, in turn, consists of four parts, any of which can be changed independently:

Inner theme The appearance of frame contents (blocks, lists, etc.) Changed with `\useinnertheme`.

[5] Have you ever seen anyone use them?

Outer theme The appearance of frame border (footer, navigation bar etc.) Changed with `\useoutertheme`.

Color theme The colours used for elements. Individual colours can be further customized. Changed with `\usecolortheme`.

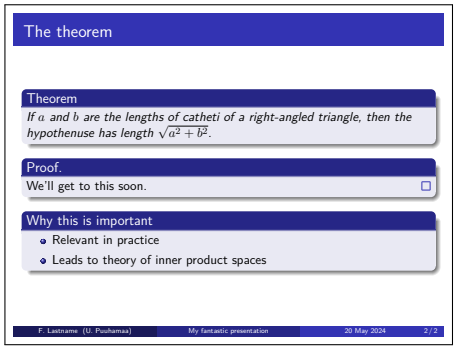
Font theme The fonts to be used for elements. Also these can be customized individually. Changed with `\usefonttheme`.

It is thus possible to start from a presentation theme, and then change parts of the style to suit your artistic tastes.

The built-in presentation themes are showcased in [6, Chapter 15], inner and outer themes in [6, Chapter 16], and color and font themes in the following chapters.

In this example, we use the `Madrid` presentation theme without any further customization:[6]

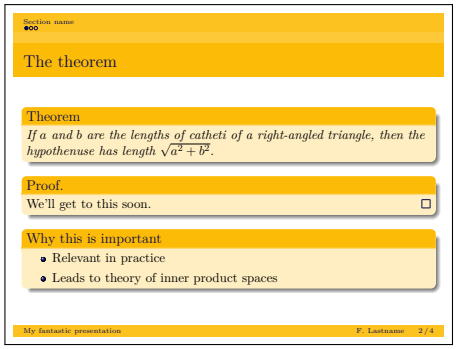
```
\title{My fantastic presentation}
\author[F.~Lastname]{Firstname Lastname}
\institute[U.~Puuhamaa]{University of Puuhamaa}
\date{20 May 2024}

\beamertemplatenavigationsymbolsempty
\usefonttheme{professionalfonts}
\usetheme{Madrid}
```



Then, let us modify the look a bit. We take the `miniframes` outer theme into use; it displays the sections and progress within section in the header. We also use its optional argument to simplify the footer. We then change to the `crane` colour theme in order to induce a headache in our audience, and use the default serif font:

[6] Except for using `professionalfonts`, since the file was compiled with XeLaTeX.

```
\beamertemplatenavigationsymbolsempty
\usefonttheme{professionalfonts}
\usetheme{Madrid}
\useoutertheme[footline=authortitle]{miniframes}
\usecolortheme{crane}
\usefonttheme{serif}
```



9.5.2 Colours

Good practices

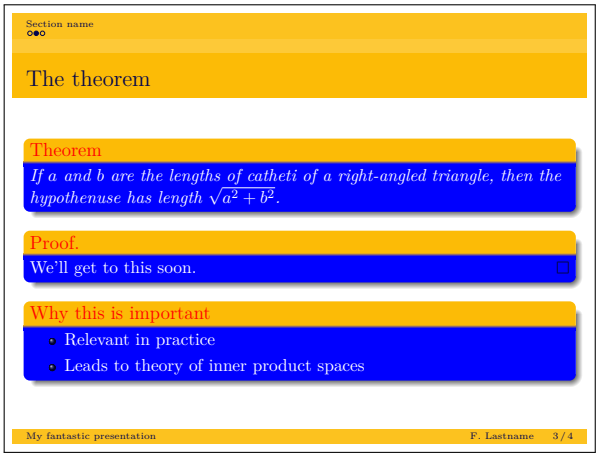
It is *extra important* to take care of colour contrast in presentations. Projectors have a far smaller colour range than computer displays, especially if the room is bright. What might appear like two distinct colours on your laptop might be indistinguishable when projected.

If the colour scheme provided by `\usecolortheme` is still not what you want, it is possible to tweak the individual colours. This is done with `\setbeamercolor`.

Beamer uses named colours for everything. For example, the `block body` colour is used inside all `block` environments. Each colour also has two components: foreground (`fg`) and background (`bg`). These can be set to any values supported by xcolor.

For example, here we change the foreground colour of block titles, and both the foreground and background colours of block bodies:

```
\setbeamercolor{block title}{fg=red}
\setbeamercolor{block body}{fg=white, bg=blue}
```



Some of these colours are also inherited from "parent colours". Maybe the most interesting of these is `structure`, which applies to all frame and block headings, unless overridden by a child style. The colour of the "background canvas" is determined by the `bg` colour of `normal text`.

You can find the full list of predefined colours in the Beamer documentation for each element type. The source code of each Beamer colour theme is also very readable, and can be used as a reference.[7]

9.5.3 Fonts

Good practices

Beamer uses a sans-serif font by default. This is a good choice since sans-serif fonts are usually more readable in low resolution. However, a serif font can be used to give a nice classic look.

Using *italic* for emphasis might be too subtle; prefer alerted and boldface text in presentations.

As with colours, Beamer comes with a few built-in font themes accessible with `\usefonttheme`. In addition to the `professionalfonts` mentioned above (which just disables some internal features, and can be used together with another font theme), there is the sans-serif `default`, its counterpart `serif`, and a few others that modify headings only [6, Section 18.1].

The font size is controlled with the package options. Moreover, since Beamer uses the default serif and sans-serif fonts, you can replace the fonts with the methods described in Section 6.3.

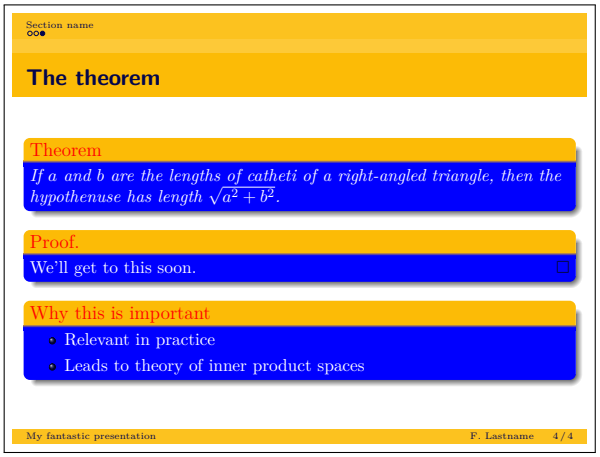
If you really need to customize the font used for a particular element, there is a system similar to the one used for colours. The `\setbeamerfont` command takes the element name and a list of attributes to override. The

[7] Where your particular LaTeX installation puts package sources might be complicated. For Beamer, the source is also available on Github: `github.com/josephwright/beamer`.

attributes are `size`, `shape`, `series`, and `family`, and their possible values are the declarations listed in Section 2.2.

In this (now awfully ugly) example, we modify the frame title to be in bold, sans-serif font:

```
\setbeamerfont{frametitle}{series=\bfseries, family=\sffamily}
```



9.5.4 Templates

Finally, the appearance of repeated elements like titles can be controlled by *templates*. Since this goes deeper into the internals of Beamer, we only refer to the documentation [6, Section 16.3].

9.6 Handouts

To produce handouts, pass the `handout` optional argument to the document class. In this mode, the overlays are "flattened" so that everything is visible at once.

If you still need to split a frame into multiple parts in the handout, you can modify the overlay specification to also apply to handouts. The following code shows its contents on the third slide in presentation mode, and on the second slide in handout mode:

```
\only<3|handout:2>{...}
```

To suppress the inclusion of a frame, you can put `<handout:0>` immediately after `\begin{frame}`. This overlay specification means that only the 0th slide of the frame will be printed in the handout – which does nothing, since the slide numbers start at 1.

Chapter 10

Posters with tikzposter

There are several methods to creating posters with LaTeX. To mention just three of the many packages: tikzposter, beamerposter, and tcolorbox. All of them suffer from the fact that posters are very visual, but TeX can really only do boxes.

Here we discuss tikzposter since it is reasonably simple. However, the package has not been actively maintained since 2014, and it appears to have some bugs. The other packages use fairly similar principles.

Remark
In particular, I would like to highlight tcolorbox for its good documentation. You should definitely check it out.

Note that its poster template does not handle large paper sizes. You can either configure the paper and font sizes by yourself, or scale up the A3-size output in the printing phase. The latter may be frowned upon by typography enthusiasts, since a 12-point font scaled 4× is *very much not the same* as a 48-point font.

10.1 Basic layout

The package is loaded with

```
\documentclass{tikzposter}
```

By default, the options `a0paper`, `25pt`, and `portrait` are used. Paper sizes A1 and A2 are also available, and the `landscape` option produces a landscape page. The range of font sizes is limited to 12, 14, 17, 20, and 25 points. Of these, only the last one is suitable for A0 size.

Good practices
As with presentations, you should not have too much text. Use whitespace and structure to guide the reader.

For one reason or other, the package adds a watermark to the lower-right corner. It can be turned off with the monstrously long command

```
\tikzposterlatexaffectionproofoff
```

There are also some package options to control margins and default block spacing; see the documentation for details.

The title is set with `\maketitle` as usual; you can set `\title`, `\author`, and `\institute` in the preamble. You can also put graphical (or rather, any) commands inside `\titlegraphic`. By default these elements are set on top of each other. The customization is briefly discussed in Section 10.2. The `\maketitle` command also accepts some optional arguments like

```
\maketitle[width=40cm]
```

As with Beamer, the basic layout consists of blocks. The `\block` command takes optional customization arguments, a title, and the contents:

```
\block[]{Block title goes here}
{
Contents of the block
}
```

The block will fill the available horizontal space, and the vertical size depends on the contents. Contents can be any material (except floats). If the title is empty, the "block header" disappears.

To set several blocks side by side, it is possible to use columns specified inside a `columns` environment. A new column is started with the `\column` command. As its argument it takes the fraction of text area width.[1] Any blocks specified after this command will be placed vertically in the column. For example, here we set two columns of equal size, and place two blocks in the left column:

[1] You need to ensure that they sum to at most 1. The documentation also talks about absolute widths, but they do not seem to actually work.

```
\begin{columns}

\column{0.5}
\block{Above left}{}
\block{Below left}{}

\column{0.5}
\block{Right}{}

\end{columns}
```

The `subcolumns` and `\subcolumn` can be used to further divide a column (the arguments are fractions of the *outer column* width).

After the `columns` environment ends, all new blocks again fill the whole horizontal space.

You cannot use the usual vertical spacing commands to adjust spacing between blocks. A workaround is to add an offset both to the block title and contents. Negative y values point downwards.[2]

```
\block[titleoffsety=-2cm, bodyoffsety=-2cm]{Title}{Contents}
```

Inside a block, you can use the `\coloredbox` command to create highlighted regions.

```
\block{Goal}
{
In this poster we discuss Pythagoras' theorem.

\coloredbox{It is maybe the most important theorem in human history!!}

\coloredbox[bgcolor=PeachPuff, fgcolor=red]{\centering I mean it!!!}
}
```

It is also possible to add small notes. They are defined outside blocks, but they attach to the previous block. By default, they connect to the center of the block, but the precise point can be adjusted with optional arguments, as with the distance and angle (here east-southeast) to the anchor point:

```
\note[angle=-20, radius=8cm, connection, targetoffsety=-2cm]
    {This poster will convince you about it!}
```

[2] It appears that this method might not work with blocks without title; this seems to be a bug.

Let us see how these pieces (and a few more content additions) play together. The output of the following code is shown in Figure 10.1.

```
\documentclass[17pt,a2paper]{tikzposter}
\usepackage{graphicx}
\usepackage[svgnames]{xcolor}
\usepackage{fontawesome6}

\tikzposterlatexaffectionproofoff

\author{Firstname Lastname}
\title{My first poster}
\institute{University of Puuhamaa}
\titlegraphic{\Huge\faDungeon}

\begin{document}
\maketitle[width=40cm]

\begin{columns}

% The left column
\column{0.5}
\block{Above left}
{
In this poster we discuss Pythagoras' theorem.
\coloredbox{It is maybe the most important theorem in human history!!}
\coloredbox[bgcolor=PeachPuff, fgcolor=red]{\centering I mean it!!!}
}

\note[angle=-20, radius=8cm, connection, targetoffsety=-2cm]
    {This poster will convince you about it!}

\block{Below left}
{
Some facts about Pythagoras:
\begin{itemize}
    \item Ancient Greek
    \item A bit weird
    \item Did not invent the theorem
\end{itemize}
}

% The right column
\column{0.5}
\begin{subcolumns}

\subcolumn{0.6}
\block{Right}
{
Here's the theorem:
\[
a^2 + b^2 = c^2
\]
```

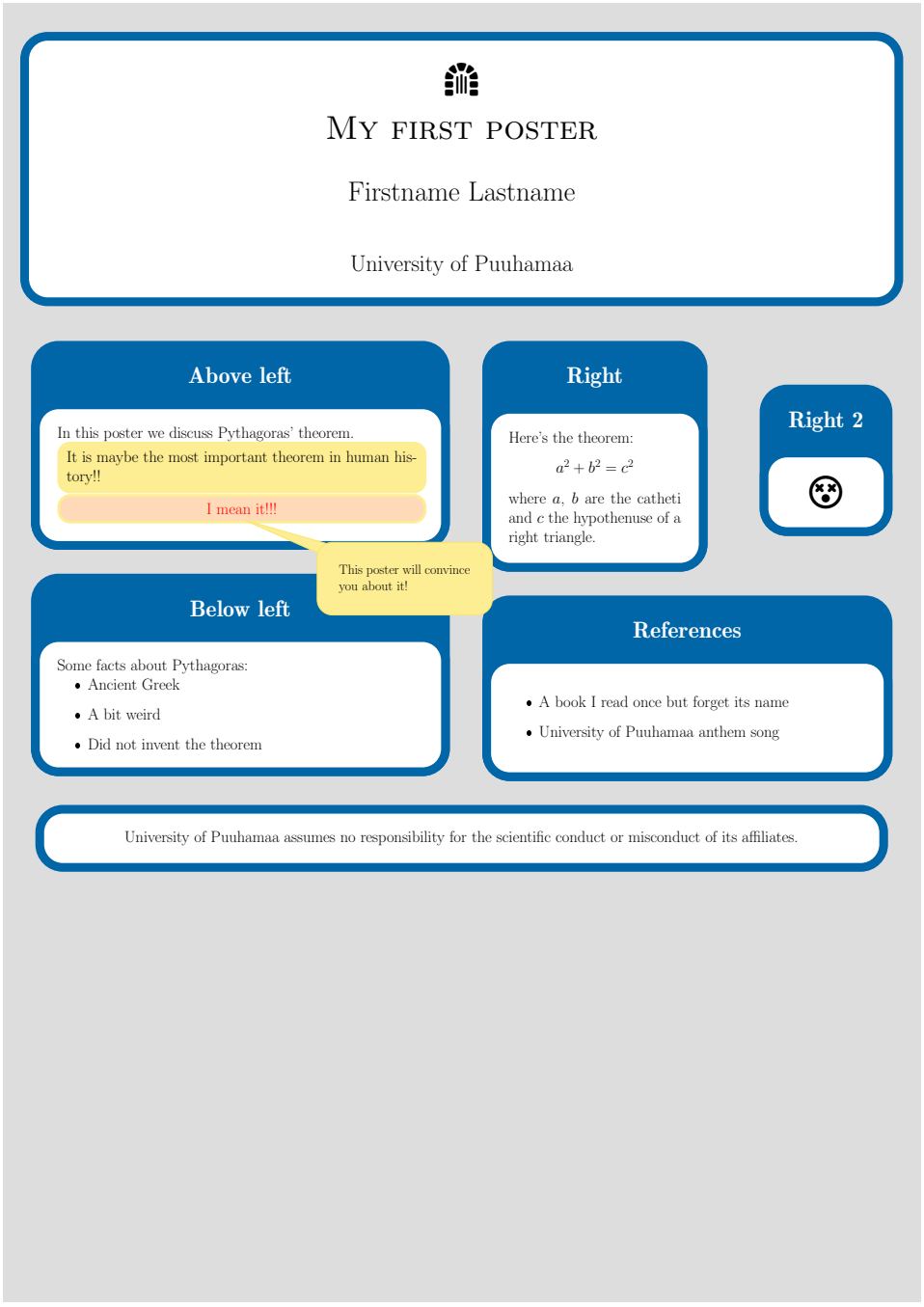


Figure 10.1: The example poster (set in A2 size, and scaled here).

```
where $a$, $b$ are the catheti
and $c$ the hypothenuse of a right triangle.
}

\subcolumn{0.4}
\block[titleoffsety=-2cm, bodyoffsety=-2cm]{Right 2}
    {\centering\Huge\faFaceDizzy[regular]}

\end{subcolumns}

\block{References}
{
\begin{itemize}
    \item A book I read once but forget its name
    \item University of Puuhamaa anthem song
\end{itemize}
}

\end{columns}


\block{}
{
\centering
University of Puuhamaa assumes no responsibility
for the scientific conduct or misconduct of its affiliates.
}

\end{document}
```

10.2 Styling

The styling system of tikzposter is very similar to that of Beamer. There are a few built-in styles.[3]

Technical side note
The source code for the predefined styles is very readable. If you want to customize styles, I recommend using their code as a starting point.

The installation location of packages depends on your specific LaTeX installation, but the code can also be browsed online at `bitbucket.org/surmann/tikzposter/`.

The overall look is controlled by `\usetheme`, and then e.g. the colour and background styles can be customized. Individual colours can be changed with `\colorlet` (unlike in Beamer, colours are not separated into foreground and background colours).

For example, this code selects the theme in Figure 10.2:

[3] `bitbucket.org/surmann/tikzposter/downloads/styleguide.pdf`

```
\usetheme{Envelope} % Block and title style set by this
\usecolorstyle{Denmark} % Red-white colour scheme

\usebackgroundstyle{VerticalGradation} % Vertical gradient...
\colorlet{backgroundcolor}{blue!40} % ...with a faint blue colour
```

To modify the layout of the title contents, it is unfortunately necessary to go to low-level LaTeX code. The `\settitle` command can be used to define the layout code for the title contents. The variables set by the `\maketitle` companion commands can be used as below. The following code is also used in Figure 10.2:

```
\settitle{
    \makebox[6cm][c]{\@titlegraphic}
    \parbox[b]{22cm}{
        \color{titlefgcolor} {\bfseries \Huge \@title \par}
        \vspace*{1em}
        {\huge \@author{} \Large (\@institute)}
    }
}
```

Technical side note
It appears that `\title` as defined by the package does not permit automatic line breaking of the title.

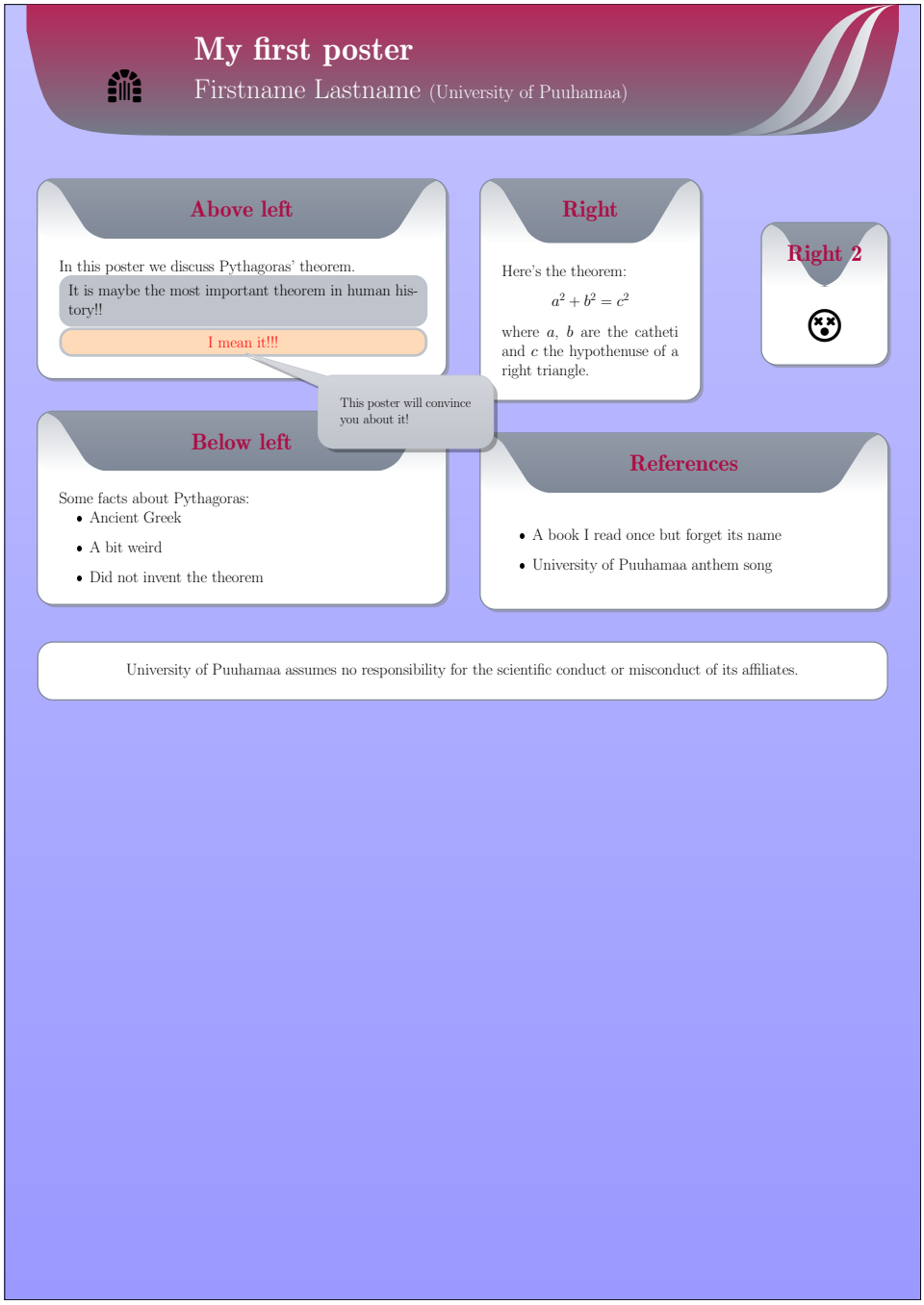


Figure 10.2: A styled version of the previous example.

Index

Bibliography

[1] Philip Kime, Moritz Wemheuer, and Philipp Lehman. *The `biblatex` package*. Version 3.20. 2024. URL: `https://ctan.org/pkg/biblatex` (cit. on pp. 114, 116).

[2] Donald Knuth. *The TEXbook*. Sixth printing. Addison-Wesley, 1986 (cit. on p. 55).

[3] Leslie Lamport. *LATEX, a document preparation system. User's guide and reference manual*. Second edition. Addison-Wesley, 1994 (cit. on p. 109).

[4] Frank Mittelbach and Ulrike Fischer. *The LATEX Companion*. Third edition. Tools and Techniques for Computer Typesetting. Addison-Wesley, 2023 (cit. on pp. 5, 18, 19, 61, 109, 138, 151).

[5] Till Tantau. *The TikZ and PGF packages*. Version 3.1.10. 2023. URL: `https://ctan.org/pkg/pgf` (cit. on pp. 109, 154, 157–160, 162, 166–168).

[6] Till Tantau, Joseph Wright, and Vedran Miletić. *The BEAMER class*. Version 3.71. 2024. URL: `https://ctan.org/pkg/beamer` (cit. on pp. 184, 186, 187).

Colophon

These notes are set in Latin Modern, an updated version of Donald Knuth's Computer Modern typeface. The typeface is a Didone design, meaning a high contrast between thin and thick lines. Knuth designed it after mathematics textbooks set in metal type. I am personally not that fond of Computer Modern, especially when reading on computer screen. I still kept it so that the notes are honest to LaTeX's default behaviour.

As you can see from the source code, there is nothing too special in these notes. The highlighted *Gotcha!* blocks are created with thmtools. There are also a few custom commands for printing command and package names: these commands also contribute to the index.

The largest programming trick was the examples. Code for the examples appears in the `.tex` code inside a `VerbatimOut` environment. The code is stored into a temporary file, and then read twice: once to show the code, and a second time to evaluate the output.

Due to this design, the examples are guaranteed to really work. However, a few examples include a few lines of secret extra code. This code mainly hides the fact that the examples live inside a minipage, and restores some customizations to LaTeX defaults.