A LaTeX Primer

Required Reading for Seekers of TeX Support

Benedikt Wilde*

Revision from 2023-06-20

In the past, I have frequently given TeXnical support to others. While I enjoy spreading the good news of LaTeX, repeating the same general advice over and over started getting rather time-consuming and a bit tiresome. For everyone who knows me, this document is meant to replace that initial session of conveying basic knowledge and good practices as well as serve as a reference for these things. It is *not* meant to discourage you from approaching me with questions or problems regarding LaTeX, though I may refuse to explain things contained in this document. For everyone else, I hope that this may serve you as a valuable read and as a useful reference. I do however have to ask you not to contact me with questions or issues, as I am unable to provide large-scale LaTeX support.

Contents

1 Introduction **4**
- 1.1 What This Document Is . . . 4
- 1.2 What This Document Is Not . . . 4
- 1.3 Do I Have to Read All of This? . . . 5

2 What It's All About **6**
- 2.1 TeX and LaTeX . . . 6
- 2.2 The Output Format . . . 7
- 2.3 Extensions . . . 7

3 Installation and Setup **7**
- 3.1 Installing LaTeX . . . 8
- 3.2 Installing TeXstudio and JabRef . . . 8
- 3.3 Allowing TeX to Call External Programs . . . 9
- 3.4 Per-Document Configuration Using Magic Comments . . . 10
- 3.5 Keeping Things up to Date . . . 10

*Please post error reports and suggestions for improvement to `github.com/schtandard/latex_primer`.

3.6 A Word on Overleaf 11

4 Very, Very Basic Stuff 11
4.1 Special Characters 11
4.2 What You Absolutely Need To Know About Macros 12
4.3 What You Absolutely Need To Know About Groups 12
4.4 The Structure of a Document 12
4.5 Spaces and Newlines 13
4.6 Switching To and From Math Mode 14
4.7 Dimensions 14
4.8 Compiling Enough Times 15

5 First Principles 16
5.1 Never Use TEX primitives 16
5.2 Use Macros and Use Them Semantically 17

6 Taking Care of Spaces 18
6.1 Normal and End-of-Sentence Spaces 18
6.2 Using Non-Breaking Spaces 18
6.3 Spaces in Abbreviations 19
6.4 Avoiding Spurious Spaces 19

7 Using the Correct Dash 20

8 Writing Clean Source Files 20
8.1 Write One Sentence per Line 20
8.2 Choose Clear Label Names 21
8.3 Liberally Use Spaces in Math Mode 22

9 Avoiding Typographical Sins 22
9.1 Trust the Defaults 23
9.2 Never Use Manual Line Breaks 23
9.3 Never Use Manual Page Breaks 23
9.4 Avoid Widows and Orphans 23
9.5 Don't Make Your Tables Ugly 23
9.6 Let Floats Float 24
9.7 Cleanly Separate Text and Math Mode 25
9.8 Use the Correct Math Font 26
9.9 Manually Add Spacing in Math Mode When Helpful 27

10 Maintaining a Bibliographic Database 28
10.1 Ensuring Integrity 28
10.2 Using the Correct Entry Fields 28
10.3 Correctly Inputting Data into the Fields 29
10.4 Which Entry Fields to Provide 29

11 Document Classes **30**
11.1 The Base Classes . . . 30
11.2 KOMA-Script . . . 30
11.3 Other Document Classes . . . 31

12 Packages **31**
12.1 Always Include These . . . 32
12.2 Programming . . . 33
12.3 Font Selection . . . 33
12.4 Text . . . 34
12.5 Graphics Inclusion and Creation . . . 35
12.6 Mathematics . . . 35
12.7 Code Listings . . . 36
12.8 Other . . . 36

13 Important Macros and Environments **37**
13.1 Font Switching . . . 37
13.2 Counters and Numbering . . . 37
13.3 Sectioning . . . 38
13.4 Numbers and Units . . . 39
13.5 Chemical Symbols . . . 40
13.6 Display Math Environments . . . 41
13.7 Spaces in Math Mode . . . 44
13.8 Quotations . . . 44
13.9 Verbatim Text . . . 45
13.10 Footnotes . . . 46
13.11 Lists . . . 46
13.12 Tables . . . 47
13.13 Including Images . . . 50
13.14 Floating Content . . . 51
13.15 Cross Referencing . . . 53
13.16 Citations and Bibliographies . . . 53
13.17 Hyperlinks . . . 54
13.18 PDF Properties . . . 55

14 How Macros Really Work **55**
14.1 Category Codes . . . 56
14.2 Macro Expansion . . . 57
14.3 Writing Macros: The TEX Way . . . 57
14.4 Expandability and Robustness . . . 61
14.5 Writing Macros: The LATEX 2ε Way . . . 63
14.6 Writing Macros: The LATEX3 Way . . . 64

15 Getting Help **67**
15.1 How Can I Produce This Symbol? 67
15.2 How Does This Package Work? 67
15.3 I'm Getting Lots of Errors! 68
15.4 I Still Need Help! 68

16 Further Reading **68**
16.1 Weblinks 68
16.2 References 69

17 Index **71**

1 Introduction

1.1 What This Document Is

This document is a list of things that everybody working with LaTeX should know. Many sections explain concepts and guidelines that are usually the first things I tell people when giving TeX support. Some of them are just good practice, others are serious mistakes that way too many people make.

This document is a rather minimal reference. If you are unsure what packages to load or which macros and environments to use for some standard task, you should find them here. Of course, this list isn't anywhere close to exhaustive.

Mainly, though, this document is a jumping-off point for people wanting to improve their LaTeX writing or ask me for help with that. Hopefully, I can reach more people with it than by individual coaching. When I do give personal advice, it shall serve me as a solid foundation I can refer to and expect to be read. If you are thinking of asking me to review your TEX file or about some specific TeXnical issue, please do, but first at least skim through this document.

As already mentioned in the abstract, I have to restrict this offer of personal help to people who actually know me. I just don't have the resources to provide personal support to the whole internet. If you are looking for help with a specific problem that you cannot solve yourself, please refer to section 15 and the resources mentioned there.

1.2 What This Document Is Not

This document is not a tutorial. If you have never even seen a LaTeX document, you will need to find help elsewhere. Preferably, ask someone who knows the stuff to help you set it up for the first time and create your first document. [learnlatex][1] and [overleaf-doc][2] may also be good starting points.

This document is not the whole truth. In fact, it isn't even nothing but the truth. In the spirit of Donald Knuth [Knu84], I choose to sometimes bend the truth or omit exceptions in

[1] https://www.learnlatex.org/
[2] https://www.overleaf.com/learn

order to make things easier to explain at first. Unlike Knuth, though, I will not clear up all of these inaccuracies later in the text. You will have to turn to other works for the full truth.

This document is not a completed work or infallible. If you spot mistakes or typos, find some passage confusing or want to request the inclusion of some aspects that I omitted, please send me an email at `benedikt.wilde@uni-tuebingen.de`.

1.3 Do I Have to Read All of This?

No. I would like everyone to read sections 4 through 9 at some point. Other than that, it really depends on your needs. Have a look at the table of contents to get an idea of what's interesting to you. Here's a summary of what is contained in which section.

- Section 2 clarifies the basic terminology of TEX and LATEX. If the differences between TEX, ε-TEX, LATEX, pdfLATEX and LuaLATEX confuse you, consider reading it.
- Section 3 explains how to set up LATEX and TEXstudio on your computer and suggests some configuration. If your configuration differs significantly from this recommendation and you ask me for help or to look over your document, I may complain.
- Section 4 briefly lists some foundational knowledge about LATEX that anyone who has worked with it for some time should already be familiar with. If that is you, maybe skim through it anyway, just in case that you have been using LATEX oblivious to some basic knowledge.
- Sections 5 through 9 discuss matters of good practice regarding both the source file and the finished product of your document. *Everyone* should read these! The concepts describe there will significantly improve your documents and make writing them easier.
- Section 10 discusses how to keep a bibliographic database for all your citation needs. If you need a bibliography in your document, you should read this.
- Section 11 offers an overview of the most important document classes. If you are unsure which one is right for your document, read it.
- Section 12 does the same for packages. You should definitely read section 12.1. Read the other subsections depending on what you need in your document.
- Section 13 lists many common macros and environments. This is the most reference-like part of this document. Read only the parts you need.
- Section 14 explains how macros work on a technical level and how to define your own. Don't read this if you are just starting out with LATEX. Once you are a bit familiar with it and start wanting to write your own macros however, it is a good idea to get a grip on the inner workings of macro expansion, which you can hopefully achieve by reading this section.

- Section 15 contains some information on how to deal with errors and where to find help regarding LaTeX and its packages. If you are an absolute beginner, it is better to just ask someone you know who is not. Otherwise, do read this section.
- Section 16 lists all the websites, package documentations and other references mentioned in this document.
- Finally, section 17 is an index containing macros, environments, packages, document classes and some keywords as a quick guide of where to find what in this document.

2 What It's All About

2.1 TeX and LaTeX

Let's get some nomenclature out of the way first. TeX is the name of both a program that translates text files into beautifully typeset documents (if used properly) and the language that is used to describe the document in the text file. This language consist of a couple hundred of primitive commands and the possibility to write macros, which allows the creation of TeX programs to solve complex typesetting tasks with a simple user interface. When its creator, Donald Knuth, published TeX in *The TeXbook* [Knu84] in 1984, he already included a small collection of useful macros in addition to the primitive commands, which together are referred to as plain TeX.

Due to its usefulness for publications especially in the sciences and mathematics, TeX was soon widely adopted in those fields and many authors created large macro systems to make working with TeX more convenient. The most prominent of those is LaTeX (originally created by and named after Leslie Lamport), which today is the de facto standard for work with TeX. (There are others, most notably ConTeXt, which I do not really know and will not mention further in this document.) LaTeX introduced a more standardized structure to macros and documents (document classes, packages, environments and optional arguments as you (will) know them are all LaTeX concepts) that allowed authors to easily share collections of macros with others as packages and document classes. Today there are thousands of LaTeX packages freely available online that cover virtually all needs most document authors could have.

The current version of LaTeX is called LaTeX 2_ε and was released in 1994 [Lam94]. Since then, it has only been modified very conservatively, fixing bugs and performing some changes to make it more suitable to new technological standards (e.g. UTF-8 support). When I write "LaTeX" in this document, I am referring to LaTeX 2_ε.

A lot of work has gone into developing a future version of LaTeX with major improvements since the late nineties, called LaTeX3. This development is still underway and nowhere close to finished. The programming layer of LaTeX3 however is rather advanced and functional already and offers tremendous advantages to programming in plain TeX. So, if you ever consider writing a complex macro or even your own package, consider reading up on that.

2.2 The Output Format

Originally, TEX output documents in the DVI format, specially developed for this purpose, and the program `tex` (and its derivative `latex`) still outputs this format. Since then, the PDF format has been developed and become the standard for digital documents. If you don't have very good reasons not to do so, you should thus use `pdflatex` or `lualatex`, which directly output PDF, instead of `latex`. Since using `pdflatex` is the default configuration of any modern LATEX editor, you need not worry about this for your first document unless you are compiling manually from the command line.

2.3 Extensions

There are several extensions to the original TEX, adding new primitives and functionality. I will briefly present the most important ones here.

Firstly, there is ε-TEX (*extended TEX*) which adds some new primitives to TEX that make writing and modifying macros easier. This extension is now the default version of TEX; when you run `pdflatex`, ε-TEX commands are available, so you don't need to worry about it.

Then there are LuaTEX and LuaLATEX. Lua is a lightweight programming language designed to be embedded into other applications. LuaTEX does this for TEX, allowing access to the TEX internals and manipulating them with a more natural programming language. This also allows the use of OTF fonts, which means that you can load any font installed on your machine in LuaLATEX. Together with its superior Unicode support, this makes LuaLATEX the preferred choice for modern LATEX documents. If you don't care about any of the new features, need to stick to pdfLATEXdue to external restrictions or want its slightly faster typesetting runs, don't worry though; pdfLATEX is still a perfectly fine typesetting system.

I also want to mention XƎLATEX, which is an extension developed for Mac OS and available on other operating systems with reduced functionality. Its main feature originally was the extended font support, when LuaLATEX was not available yet. Nowadays however, I would not recommend using XƎLATEX unless you really need it for some reason.

3 Installation and Setup

In order to write Documents using LATEX, you need several programs on your computer. Most importantly, you need the LATEX program itself. Secondly, you need a text editor. Any text editor works, but there are specialized so-called LATEX editors that have useful features like code highlighting and completion, an integrated PDF viewer and more. I recommend using TEXstudio for this. Thirdly, you will probably not want to type out your bibliography database file yourself, as this is rather tiresome and it is difficult to search through the raw text file. There are editors for this, too, that offer a more natural handling of the database and manage the text file for you in the background. I recommend using JabRef for this.

3.1 Installing LaTeX

There are two major distributions you can choose from in order to install LaTeX on your computer; MiKTeX and TeX Live, which you can download from their websites[3,4]. Both of them contain the usual TeX engines (plain TeX, pdfLaTeX, LuaLaTeX, etc.) and most packages on the *Comprehensive TeX Archive Network*[5] (*CTAN*). Here are the most important differences between them, some more are listed in [miketx-vs-texlive][6].

- MiKTeX is only available for Windows. This also means that it can focus on Windows more and has a more natural feel for Windows users. TeX Live is also available for Linux and MacOS.
- By default, TeX Live installs all of its packages (which entails several gigabytes of small file downloads) while MiKTeX only installs a small selection of packages and adds more as needed on the fly. You can also get a full MiKTeX installation right away or select what to install in TeX Live, but that requires manual intervention.
- TeX Live only contains packages that are free software. This rarely makes a difference as all important packages are free and you can easily add non-free packages yourself.

In the end, you can just pick one if you are on Windows, otherwise you have to use TeX Live.

3.2 Installing TeXstudio and JabRef

You can download TeXstudio and JabRef from their websites[7,8] and install them as usual. To get the most from these programs, some configuration is in order. You can either load the settings I recommend by importing the files in the `preferences/` folder accompanying this document by

- placing `texstudio.ini` into the folder `%appdata%/texstudio/` and
- loading `jabref_config.xml` using the ↙ button in the JabRef settings

or you can perform all of them manually. Here is a list of all the changes I performed before exporting the settings files mentioned above. Note that some of these settings are specific to the operating system (I used Windows) or how you installed the programs.

TeXstudio In the menu, open Options, Configure TeXstudio... and select Show Advanced Options in the bottom left corner. On the left side you will find a list of categories in which the preferences ar grouped.

General

- Set Style to “Adwaita Dark (txs)”.
- Set Max. Recent Documents to “12”.

[3] `https://miktex.org/download`
[4] `https://www.tug.org/texlive/acquire-netinstall.html`
[5] `https://ctan.org/`
[6] `https://tex.stackexchange.com/q/20036`
[7] `https://texstudio.org/`
[8] `https://www.jabref.org/`

- Set Max. Recent Master Documents to "7".

Commands
- Uncheck all buttons for repeating compilation commands.

Build
- Uncheck all buttons for repeating compilation commands.
- Set Default Bibliography Tool to "txs:///biber".
- Uncheck Check and update bibliography before compiling.
- Change Show stdout to "Always (If not redirected > /dev/null)".

Editor
- Change Indentation Mode to "Keep Indentation".
- Check Replace Intentation Tab by Spaces.
- Check Replace Tab in Text by Spaces.
- Check Remove Trailing Whitespace on Save.
- Change Show Line Numbers to "All Line Numbers".

Adv. Editor
- Change Tab Width to "2".
- Change Line Wrapping to "No Line Wrap".

Completion
- Uncheck Automatically start completer when typing LaTeX-Commands.

Preview
- Uncheck Show preview as tooltip on formulas in editor.

SVN/GIT
- Select "Use GIT".

You may also want to hide the "Central" button bar of the TEXstudio window.

JabRef In the menu, open Options, Preferences. Again, there is a list of categories of preferences on the left side of the window.

General
- Set Default library mode to "biblatex".

Citation key generator
- Set Default key pattern to "[auth:lower][year]".
- Set Book key pattern to "[auth:lower][year][veryshorttitle]".

Appearance
- Select Dark theme.

3.3 Allowing TEX to Call External Programs

Sometimes you want TEX to call an external program, for example to prepare your bibliography, to convert an image or to parse some program code you want to show. On the other hand, it is a substantial security risk if you compile a document you did not write yourself and allow that document to execute *any code* on your computer. Because of this, both MiKTEX

and TEX Live restrict TEX to only be able to call a select few external programs, like `biber` or `makeindex`.

Escaping to the shell from TEX is also referred to as `\write18`, after the TEX primitive used to do it. If you want to lift the shell escape restrictions, you can run LATEX with the option `--shell-escape`. You can add it to the command your editor uses to compile documents in its preferences, but you should then be careful when compiling foreign documents. Instead, it may be better to only allow shell escape on a per-document basis, see below.

3.4 Per-Document Configuration Using Magic Comments

You can specify how an editor should handle a file with so-called *magic comments*. These are lines of text, usually at the beginning of the file, that start with *% !*. To TEX, these are normal comments and they are thus ignored. Editors, however, can parse these lines and adapt their behavior accordingly.

Not all editors support the same set of magic comments or any at all. Here is a list of magic comments supported by TEXstudio.

% !TeX encoding = utf8 specifies the character encoding of the file to be UTF-8. Note that this does *not* actually change the encoding or change how TEX reads the file. It only affects how the TEXstudio reads the file.

% !TeX spellcheck = de_DE specifies the language TEXstudio should use for spell checking.

% !TeX program = lualatex specifies that TEXstudio should use `lualatex` to compile the document (as specified in the Commands section of the settings).

% !TeX root = filename specifies the root document of the current file to be `filename`. This is useful when separating the preamble or chapters of a document into separate files. For example, if your main document file is called `document.tex` and your preamble is in a separate file `preamble.tex`, you should add the magic comment *% !TeX root = document* to the preamble.

% !TeX TXS-program:bibliography = txs:///biber is a TEXstudio-specific magic comment. It allows you to change any of the commands set in the Commands section of the settings, this one selects `biber` as the bibliography tool. As another example, if you want to compile the current document with LuaLATEX with shell escape enabled, you could use *% !TeX TXS-program:compile = txs:///lualatex/[--shell-escape]*.

3.5 Keeping Things up to Date

Most of the thousands of packages that make up the LATEX ecosystem are continuously maintained and improved. In order to receive bug-fixes and be able to use the latest features, you should regularly update your LATEX distribution, at least every couple of months. You can do this in your distribution's package manager (called *MiKTeX Console* in MiKTEX and *TeX Live Manager* in TEX Live). If you are using TEX Live, you will additionally have to install a new version every year.

While this does not usually happen, there is always the possibility that an update will break something. This is especially true if you use some evil hack exploiting the undocumented internals of a package (which is more common than one might expect). Thus, if you have a deadline coming up (say you are in the last phases of writing a thesis or a paper) and everything is working fine, you should *not* update until after that project is finished.

3.6 A Word on Overleaf

Overleaf[9] is a service for writing and compiling LaTeX documents online. While it has its merits, especially if you need to collaborate with other people on the same document or for people who are new to LaTeX and want to try it without having to set it up on their own machine first, I generally recommend working on a local installation rather than using Overleaf. Especially for large projects it is much easier if you have direct access to and control over all of the files (including temporary files) belonging to that project.

One obvious example of this is being able to just open and modify an image that is included by your document, or to examine the auxiliary files produced by LaTeX when something does not work as desired. Another possible downside of Overleaf may be that its LaTeX installation (as well as other software you may want to use, e.g. Inkscape) is always lagging behind the current release a bit, so new features and bug-fixes are not immediately available there. The advantages and disadvantages of using Overleaf will vary depending on your project, of course, but in general a local installation is to be preferred.

4 Very, Very Basic Stuff

4.1 Special Characters

Most of the characters you type into your LaTeX file will appear as that character in the finished document. Some of them have a special meaning, though.

- `\` (escape character)
- `{` (begin grouping)
- `}` (end grouping)
- `$` (math shift)
- `&` (alignment tab)
- `#` (macro parameter)
- `^` (math superscript)
- `_` (math subscript)
- `%` (comment character)
- There are some *active* characters that act like macros, usually `~` and `"` are relevant.

In order to print these characters to your document, you will have to use appropriate macros.

Spaces and end of line characters are a bit special, too, see section 4.5.

[9] `https://www.overleaf.com/`

4.2 What You Absolutely Need To Know About Macros

- A macro is a command that "does something".
- Macros names consist of a backslash (\) followed by either an arbitrary number of letters (this is called a *control word*) or by exactly one non-letter (this is called a *control symbol*). Either is referred to as a *control sequence*.
- If a control word is followed by space characters, they are ignored (cf. section 4.5).
- Macros can take arguments.
  - Mandatory arguments are placed in braces, `\section{Something Cool}`, for example, starts a new section with the heading "Something Cool".
  - Optional arguments are placed in brackets and may be omitted, for example `\section[Something]{Something Cool But Really Long}` starts a new section with the heading "Something Cool But Really Long" which appears as "Something" in the table of contents.
- `\begin` and `\end` are special macros that start and end *environments*. They take one mandatory argument, the environment name. Environments affect everything between these two macros, for example the `center` environment centers everything between `\begin{center}` and `\end{center}` (and adds some vertical space, too).
- Sometimes there are starred versions of macros or environments that behave slightly differently, for example `\section*{Something Cool}` will produce an unnumbered section.

4.3 What You Absolutely Need To Know About Groups

Usually, options and macro definitions are set locally, i.e. their effect stops at the end of the current group. A group is delimited by braces (`{` and `}`) or by `\begingroup` and `\endgroup`. Environments also form groups. As an example, consider `\scshape`. It switches to small caps, but when it is enclosed in a group, the font switches back after the end of the group.

```
This text {will be \scshape in small caps. But not} for long.
```

This text will be IN SMALL CAPS. BUT NOT for long.

Note that when a macro consumes an argument that is enclosed in braces, the braces are removed. In order to group the argument's content, you will need to add an extra layer of braces (unless the macro takes care of this for you).

4.4 The Structure of a Document

Let's look at a minimal LaTeX document.

```
\documentclass{article}
```

```
\begin{document}

Hello World!

\end{document}
```

Any LaTeX document has to start with `\documentclass`, include `\begin{document}` and end with `\end{document}`. The part before `\begin{document}` is called the *preamble* of the document. Anything after `\end{document}` will be ignored.

The document class provides the framework for your document, it sets up the main layout and makes essential macros for sectioning available (cf. section 11). The preamble cannot contain any typeset material. In it, you should load packages (cf. section 12), change options and create or modify macros (if you know what you are doing; cf. section 14).

The document proper is written between `\begin{document}` and `\end{document}`. Any text you write here will be printed onto the pages of your document.

4.5 Spaces and Newlines

- A space in the source maps to a space in the document.
- Several spaces are like one space.
- Single newlines are like a space.
- Spaces at the start of lines are ignored.

```
This␣means␣␣that␣␣␣␣all␣␣␣␣␣␣these␣␣␣␣␣␣␣␣␣spaces
don't␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣even␣␣␣␣␣do␣␣␣anything.
```

This means that all these spaces don't even do anything.

- An empty line (or a line containing only spaces) starts a new paragraph.
- Several empty lines are like one.

```
Here is a paragraph that is
broken up into lines by \TeX.
When the paragraph is finished,
I leave a blank line.

This starts a new paragraph that is
again broken up into lines by \TeX.
Oh, the circle of life\dots
```

Here is a paragraph that is broken up into lines by TeX. When the paragraph is finished, I leave a blank line.

This starts a new paragraph that is again broken up into lines by TeX. Oh, the circle of life…

Since control words (i.e. pretty much all macros) gobble up subsequent spaces, you have to be careful if you want to insert a space directly after a macro. `\LaTeX is great!` would print "LaTeXis great!".

- You can use the macro `\␣` (a backslash followed by a space) after the macro, which inserts a space: `\LaTeX\ is great!`
- You can put an empty group after the macro, so the space is no longer the next thing: `\LaTeX{} is great!`
- You can enclose the macro in a group: `{\LaTeX} is great!`

Note that the last option will also enclose any local definitions or assignments the macro performs in the group, preventing them from affecting the rest of the text.

4.6 Switching To and From Math Mode

You can switch to and from inline math mode using `\(` and `\)`.

```
In a right triangle \(a^2 + b^2 = c^2\) holds.
```

In a right triangle $a^2 + b^2 = c^2$ holds.

For display math mode, use `\[` and `\]`.

```
\[
  \sum_{k = 1}^\infty 2^{-k} = 1 = (\sin x)^2 + (\cos x)^2
\]
```

$$\sum_{k=1}^{\infty} 2^{-k} = 1 = (\sin x)^2 + (\cos x)^2$$

For displayed math there is a number of environments with different spacing, alignment and numbering properties, see section 13.6.

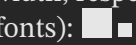
In math mode the spacing rules are different and considerably more complicated than in text mode. Space characters in the source don't have any effect (except for terminating control words and improving readability). There must not be any blank lines in math mode! Letters are always typeset as variables, unless you specify otherwise. Many macros may only be used either in text or in math mode.

You will see many people using the primitive TEX math shift `$` instead of the LATEX macros `\(` and `\)` to switch to and from inline math mode. While this does the same thing, it is clearer and more consistent to use the latter (cf. section 5.1). You must *never ever* use the TEX switch for display math (`$$`)!

4.7 Dimensions

Whenever TEX expects a length, you can specify it in one of several units. Here are the most important ones (a complete list is given in [tex-units][10]).

[10] https://tex.stackexchange.com/a/41371

- `mm` and `cm` mean what you would expect. If you prefer (or are forced to use) stupid units, there is also `in` for inches.
- `pt` for typographic points. 72.27 pt = 1 in. The font size in this document is set to 11 pt.
- `em` and `ex` are relative units that depend on the current font. It is often best to specify dimensions using these units, so that the choice still makes sense when your font (size) changes. 1 em is the width of a `\quad` and about the width of the letter M. 1 ex is about the height of the letter x. The intention is for `em` to be used for horizontal lengths and `ex` for vertical lengths, though that is not enforced in any way. Here are a square of 1 em and 1 ex width, respectively (remember that they may have different proportions with different fonts): 

There are also variables that contain dimensions. Their names look like macros, but they contain a length instead of an expansion. Here are the most important ones.

- `\textwidth` and `\textheight` for the size of the type area.
- `\columnwidth` for the width of the current column of text (in single-column documents this is the same as `\textwidth`).
- `\linewidth` for the width of the current line of text.

When TEX expects a length, instead of providing it explicitly, you can use a dimension variable. You may prefix the variable with a numeric factor, which will result in the product of the factor and the length stored in the variable being used.

Both in the numeric values of explicit dimensions and in factors preceding dimension variables, all leading zeros may be omitted.

Let us list some examples for clarity.

- `\hspace{.5em}` will create a horizontal space of 0.5 em width.
- `\vspace{4ex}` will create a vertical space of 4 ex height.
- `\includegraphics[width=.6\linewidth]{cute_cat}` will include the requested picture scaled to a width of 60 % of the line width.

4.8 Compiling Enough Times

TEX reads your TEX file top to bottom and creates your document page by page. This means that, when outputting a page, it does not know what will be on subsequent pages or if there will even be any. Because of this, auxiliary files are needed in order to facilitate things like a table of contents and cross references. When running LATEX, information necessary for these document elements is written to these files and left there for the next LATEX run. On the next run, this information is now present right from the start.

A consequence of this is that you often have to run LATEX *twice* before your table of contents or your references get updated. Many editors try to detect this and compile enough times automatically. However, this does not always work reliably. Furthermore, it can be quite irritating to have to wait for LATEX to finish several times when you just wanted to quickly compile while writing, e.g. to check the appearance of some math expression. Because of this, I prefer to disable this automatic guessing of necessary compilations and to just do it manually before publishing.

Another factor that necessitates extra compilation runs are external tools. Some packages use external programs to prepare some information for the next LaTeX run. Examples are `biber` for bibliographic data and `makeindex` or `xindy` for index/glossary preparation. With them, you need to run LaTeX first, then these tools, and then LaTeX twice more.

5 First Principles

In my opinion, the two main things that make LaTeX a superior typesetting system for authors are the following:

- All its endlessly flexible features are readily available just via typing.
- It allows (and in fact encourages) the separation of design from content.

The first point means that you can type your text without having to raise your hands from the keyboard to tediously click through menus and find just the right symbol or function. This is not only useful for technical writing where mathematical typesetting is needed. Texts in the humanities or non-academic writing, too, can be riddled with citations, cross references and special formatting of single words or phrases. Or maybe you want to have an index for your document? No problem, you just need to type the appropriate commands.

The second point frees you to write your text without having to decide the whole layout in advance. You can easily set a placeholder definition for, say, brand names and decide later on if you want to typeset them in italic or sans serif font, in a different color or with a box around them. Or maybe you don't want to use no special markup at all? You don't have to decide right away, that can wait until right before publication. This also makes it incredibly easy to get a consistent style across different documents or even with different authors. In addition, all the preferences are stored in a text file, can easily be copied and modified even long after the document was first written.

There are more advantages of LaTeX over other typesetting software, like the fact that it has great backwards compatibility, available on all platforms and that it makes it rather hard to commit the worst typographical sins. The fact that the document is stored as a simple text file is also a big advantage as the whole file is human-readable and can easily be managed using common version control software like *Git*[11]. For the process of writing, however, the two points listed above are, I think, most relevant. The magnitude of these is hard to convey to someone who has no experience with LaTeX and is maybe intimidated by its non-WYSIWYG user interface, and many a LaTeX user may not have learned how to take full advantage of them. I will try to outline some principles here, that should help you get the most out of LaTeX for your writing process.

5.1 Never Use TeX primitives

In plain TeX, writing a document involved a lot of manual adjustments and small programming tasks. With LaTeX, virtually all of this has already been done for you. Sophisticated

[11] `https://git-scm.com/`

document classes and packages do that work in the background and you can tweak them through a well documented user interface. Do that! Don't try to reinvent the wheel.

LaTeX with its thousands of packages has a macro for everything! Even when you think the plain TeX version seems to do the same thing (Why not use `\bf`, it does the same thing as `\bfseries` and is shorter, right?), don't (Well, it doesn't. `\bfseries` is better and you should *never* use `\bf`!).

The one exception many people make and that you will find in most code examples online is the way to switch to and from inline math. The LaTeX way is to use `\(` and `\)`, the TeX way is to use `$` for both. The reason for this exception is that they are functionally equivalent and people prefer typing one character over typing two. (Also, some people claim that the source is much more readable because `\(` and `\)` disappear in the text much more easily than `$`, but I would argue that that is not true in the age of code highlighting and a non-issue anyways, provided you use enough spaces in math mode.) So, what are the advantages of using `\(` and `\)`?

- It is consistent. You use LaTeX syntax and avoid TeX syntax for everything else, right?
- There is a different command for entering math mode than for exiting it. This also allows clearer error messages when you have a math shift too many or too few.
- `\(` and `\)` are macros. This means that you can easily modify them to do additional things like, say, draw a box around each inline formula or print them in a different color. Try modifying `$`!

So, I recommend going the LaTeX way, but I won't hit you if you don't.

5.2 Use Macros and Use Them Semantically

Don't be afraid to write simple macros! Have a lot of street names in your text and you aren't sure if you want to use a different font for those? Write a macro `\street` (it doesn't have to do anything right now). You have names of clubs and city offices and you want to print both of them in slanted font? Write two macros `\club` and `\office`. Have names or values in your text that are subject to change? Make them macros.

So, create new macros liberally and remember that it is important to name them semantically: The name of the macro should reflect what it *means*, what it *represents* and not how it will affect the output. Macros that have the same effect on their arguments but represent different things should have different names. This makes the document source more understandable to future readers, makes the macro names easy to remember as you write your document and enables you to easily change the formatting of just one thing and leave others unaffected.

As an example, the `\times` macro prints the symbol $\times$ in math mode. I prefer to use the symbol $\cdot$ that can be created with `\cdot` for multiplication. Now, many people just write `\cdot` instead of `\times`, but it is better to change the definition so that `\times` produces $\cdot$. You can store the original definition of `\times` in a different macro, say `\xtimes` or `\vectimes`, depending on how you plan on using it.

Another example is the notation of vectors. By default, `\vec{s}` prints $\vec{s}$, but I prefer bold symbols instead of the arrow to denote vectors. So, I change the definition of `\vec` so that `\vec{s}` now prints $\boldsymbol{s}$.

6 Taking Care of Spaces

6.1 Normal and End-of-Sentence Spaces

In some languages, including English, spaces between sentences are commonly printed wider than normal inter-word spaces. (Carefully look at the spaces in this document for an example.) TEX tries to detect which spaces are end-of-sentence spaces, but of course that is not entirely trivial. Every period is taken to be the end of a sentence, unless it is preceded by an uppercase letter. While this rule gets most cases right, sometimes you have to correct it.

- If the sentence ends with a capital letter, you have to add `\@` before the period

```
I live in the US\@. I was not born there, though.
```

- If the period is not the end of the sentence, you can
  - add `\@` after the period

```
Miller et al.\@ solved the problem.
```

    or
  - manually insert a normal space after the period using `\␣`

```
Miller et al.\ solved the problem.
```

  Most times this happens, the period denotes an abbreviation and you don't want a line break to follow it (cf. section 6.2). In those cases, you can
  - insert a non-breaking space after the period using `~`

```
The group is abelian, i.e.~commutative.
```

Exclamation points and question marks show the same behavior as periods. Colons, semicolons and commas, too, though to a lesser extent.

If you don't want end-of-sentence spaces to be different from normal inter-word spaces, you can remove this effect with `\frenchspacing`. For many languages, including German, this is the default. If you write in one of those languages or explicitly use `\frenchspacing`, you don't *have* to pay attention to all of this, but if you sometimes need it, I would recommend always doing it as a matter of good practice.

6.2 Using Non-Breaking Spaces

Sometimes, you don't want a line break to occur between two words. In those cases, you have to insert a non-breaking space instead of a normal one between the two words. In LATEX, this can be done using a tilde (~). The most common use cases for this are:

- Between words and numbers, e.g.
  - `room~101`

- 72076~Tübingen
- section~\ref{sec:example}
- The number~\(\cpi\) has a value of about~\(3\).

- In front of citations or short explanatory parentheses and symbols, e.g.
  - Leslie Lamport is the author of \LaTeX~\cite{lamport1994latex}
  - Have you been to the United States of America~(USA)?
  - The entropy~\(S\) of a system never declines.

- After many common abbreviations, e.g.
  - Many metals are superconductors, e.g.~Niobium and Aluminium.
  - Don't be stupid, i.e.~use \LaTeX, not Word!
  - Mr~and Mrs~Smith

There are, of course, more situations where the use of a non-breaking space is appropriate. Make a habit of always using a non-breaking space in these cases.

6.3 Spaces in Abbreviations

In English, there must not be any spaces between a period and the following letter of an abbreviation; you should write `e.g.` for “exempli gratia”.

In German, however, abbreviations require (thin, non-breaking) spaces between the different words; you should write `z.\,B.` for “zum Beispiel”. This may seem a bit unfamiliar at first but if you start paying attention to it you will notice that this is what you have always been seeing in professionally typeset German text. I suspect the habit of omitting these spaces (like the habit of only ever using one dash, cf. section 7) to be a remnant of the typewriter age when only a very limited set of glyphs were available to many authors.

Note that you should not use a lot of abbreviations in most texts. It is usually better to just write out the word or words.

6.4 Avoiding Spurious Spaces

As mentioned in section 4.5, a single newline is treated by TEX like a space. When you want to break some content over several lines in the source for readability and clarity, but not insert spaces between the lines, you need to comment the newline.

```
\colorsquare{purple}
\colorsquare{teal}

\colorsquare{purple}%
\colorsquare{teal}
```

If the last token in a line is a control word, the newline character is gobbled, just like a space, so you don’t have to comment it.

```
\purplesquare
\tealsquare
```

7 Using the Correct Dash

There are four different dashes that regularly occur in texts:

- The hyphen (German *Bindestrich*) created using -: "-"
  This dash is used to connect words as in "non-breaking space".
- The en-dash (German *Halbgeviertstrich*) which is created using --: "–"
  It has many uses, most notably ranges ("There may be 2 – 5 people per room.") and, in German writing, separating some statement from the surrounding sentence (German *Gedankenstrich*; "Es gibt viele Anwendungen – wie dieser Satz beispielhaft zeigt – für den Halbgeviertstrich."). In English writing (*not* in German), it is also used as a substitute for the hyphen when two names belonging to different persons are to be connected, as in "Ginzburg–Landau theory".
- The em-dash (German *Geviertstrich*) which is created using ---: "—"
  It isn't really used at all in German texts, but in English writing it serves many of the purposes the en-dash does in German.
- The minus sign, which is created using $-$ in math-mode: "$-$"
  Obviously, this is used to denote subtraction as well as negative numbers ("$5-7=-2$").

For comparison, here are all of them next to each other: - – — −.

Depending on the font used, these dashes may differ (apart from width) in thickness, vertical position and shape. Make sure to always use the correct dash for your application! The wrong dash will look hideous and impede reading your text.

8 Writing Clean Source Files

While many aspects of the formatting of your source file do not affect the compiled document, it is still advisable to keep it clean and easy to read. This helps you stay in control of your whole document while writing and is especially important if you or anyone else is to look at that file later on. Here are some guidelines you should follow in order to prevent your source from becoming an utter mess.

8.1 Write One Sentence per Line

For this to work, you will have to disable automatic line wrapping in your editor, which I would recommend anyway, because a changing appearance of your text with changing window sizes is confusing. Following this guideline has many advantages.

- The number of sentences per paragraph is obvious.

- The (approximate) length of sentences is immediately apparent. This is especially useful if you tend to sometimes write very long sentences (German *Bandwurmsatz*). You can quickly spot them as lines that are much longer than most.
- Consecutive sentences starting with the same word(s) are easily detected, as equal words right above each other catch the eye.
- The fact that all sentence endings are at the end of a line makes it much easier to search for periods that do not mark the end of a sentence. This may be relevant for the spacing inserted by TEX (cf. section 6.1). It also helps identifying places where you likely want to insert a non-breaking space (cf. section 6.2).
- When LATEX wants to alert you to something in your source (errors, warnings, etc.), it tells you on which line of the source text it noticed whatever it is telling you about. If one line of text contains an entire paragraph (or even more), this will not help you very much. Keeping the lines to a comprehensible length allows you to more easily find mistakes and avoid problems.
- Automatic tools for comparing different versions of your document, like `diff`, compare files on a line-by-line basis. This would not be very helpful if lines contained entire paragraphs.

Note that I diverge from this guideline in some code examples in this document. This is owed solely to the space restrictions of a page and should not be taken as something to emulate in a real TEX file.

8.2 Choose Clear Label Names

When labeling document elements (using `\label`), the label name you choose should be something clearly identifying the object it refers to. Ideally, you should be able to remember the name of anything you want to reference. You should certainly be able to remember what a label refers to when reading its name.

In order to achieve this, it is useful to follow the following naming convention.

- Every label consists of two parts. First, the type of element it refers to, and second, a description of the element, separated by a colon: ⟨*type*⟩:⟨*description*⟩
- The ⟨*type*⟩ is a shortened form of the thing you are labeling. For example, I use `chap` for chapters, `sec` for sections, subsections and subsubsections, `fig` for figures, `tab` for tables and `eq` for equations.
- The ⟨*description*⟩ consists of letters and hyphens and is verbose enough to clearly identify the object. I usually don't abbreviate words in the ⟨*description*⟩.

For example, this section (8.2) has the label `sec:label-clearly` and section 13.4 has the label `sec:number-unit-macros`. Figure 1 on page 51 has the label `fig:tex-lion` and table 1 on page 52 has the label `tab:natural-constants`. Even without looking at the objects I just referenced, you can probably guess what you will find there, which is exactly the point.

8.3 Liberally Use Spaces in Math Mode

Math expressions often consist of mainly macros and single letters. Since spaces in math mode do not affect the output, you could theoretically live without using any spaces there. This will however lead to a terrible mess nobody wants to read. It is much better to liberally use spaces. Insert single spaces between every element of the math expression (variables, operators, etc.) that do not form a unit (the exponent should be attached directly to its base). When the expression gets really long, it may be useful to break it across lines. Try and find out what helps, go with more spaces when in doubt.

```
\(\sqrt{ab}\leq\frac{a+b}{2}\) is way worse than
\(\sqrt{a b} \leq \frac{a + b}{2}\)
and
\[
  \frac{n}{\sum_{i=1}^n\frac{1}{a_i}}=\sqrt{\sum_{i=1}^na_i^2}\quad\iff
    \quad a_i=a_j\;\forall\;i,j\in\{1,\dots,n\}
\]
is way worse than
\[
  \frac{n}{ \sum_{i = 1}^n \frac{1}{a_i} }
  = \sqrt{ \sum_{i = 1}^n a_i^2 }
  \quad\iff\quad
  a_i = a_j \;\forall\; i, j \in \{1, \dots, n\} .
\]
```

$\sqrt{ab} \leq \frac{a+b}{2}$ is way worse than $\sqrt{ab} \leq \frac{a+b}{2}$ and

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{a_i}} = \sqrt{\sum_{i=1}^{n} a_i^2} \quad \iff \quad a_i = a_j \; \forall \; i, j \in \{1, \dots, n\}$$

is way worse than

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{a_i}} = \sqrt{\sum_{i=1}^{n} a_i^2} \quad \iff \quad a_i = a_j \; \forall \; i, j \in \{1, \dots, n\}.$$

9 Avoiding Typographical Sins

While LaTeX makes it pretty hard to make certain typographical mistakes, especially regarding consistent formatting, there are still some errors regularly made by inexperienced users. We already mentioned spaces and dashes. Here are some guidelines to help you avoid other issues.

9.1 Trust the Defaults

LaTeX and its packages usually have pretty good defaults set for all options. If you know what you are doing or you have to follow some requirements, e.g. from your school, go ahead and change them. But if something just feels different than in some other typesetting software, it's usually better to leave it be. When you are finished writing your document, you can still think about changing these options in the preamble, if necessary.

Page margins are a typical example for this. KOMA-Script has default margins that many new users perceive to be rather large. This is on purpose and actually common in good typography. In fact, the KOMA-Script manual explains this default in some length. If your school forces you to use some horrible settings like 2 cm of margin on each side, fine. Otherwise, just let it be. If you still want to reduce the margin, using KOMA-Script options (like `DIV` or maybe `\areaset`) is usually much preferable to a crude totally manual definition.

9.2 Never Use Manual Line Breaks

When writing text, you should let TeX do the line breaking for you. TeX works very hard to break your paragraphs into lines nicely and usually succeeds in doing so. If you get a bad line break (a word sticks out or the spaces get stretched a lot), then there is probably no good way of breaking that line. The best way to deal with that is usually to rephrase the sentence in question.

Note that I am breaking this rule in some of the code examples in this document. This is purely to conserve space (an empty line would take up more) and should not be taken as something to emulate (the examples in question would not appear like that in a real document).

9.3 Never Use Manual Page Breaks

The reasoning of the previous section applies analogously.

9.4 Avoid Widows and Orphans

When a page break occurs right before the last line of a paragraph, this line appears isolated at the top of the next page. This is called a *widow* (German *Hurenkind*). The opposite case, when a page break occurs right after the first line of a paragraph and leaves that line isolated at the bottom of the page, is called an *orphan* (German *Schusterjunge*). Both of these should absolutely be avoided.

Luckily, this is rather easy in LaTeX using the nowidow package as described below in section 12.1.

9.5 Don't Make Your Tables Ugly

Inexperienced users often try to recreate the look of tables they are used from other typesetting software in LaTeX, emulating inadequate layouts. This leads to results like this.

Branch	Language	Word for Bread	Word for Water
Germanic	English	bread	water
	German	Brot	Wasser
	Norwegian	brød	vann
Romance	Spanish	pan	agua
	French	pain	eau
Finno-Ugric	Estonian	leib	vesi
Slavic	Slovak	chlieb	voda

There are many bad things about this table. The main one is that there are way too many lines. They make the table considerably less comprehensive. Don't do that, don't imprison your data [data-prison][12]!

The booktabs package helps produce beautiful, comprehensive tables. Since it cannot be said often enough, I will quote the two most important guidelines listed in its documentation:

1. Never, ever use vertical rules.
2. Never use double rules.

For some examples of how to write good tables, see section 13.12.

9.6 Let Floats Float

Documents usually consist mainly of text. Since the height of a line of text is much smaller than the height of the type area, it is typically not too difficult to break the document into pages at reasonable positions. However, there are some objects regularly appearing in documents that are much larger than a line of text, namely figures and tables. When just inserted in the text, they often necessitate very early page breaks leading to underfull pages or leave only one or two lines between, say, the image and the page border, which is not satisfying at all.

To solve this problem, LaTeX uses so-called floats. Basically, you can tell LaTeX to "take this box and place it wherever", the box is allowed to "float" around your document. Every such object then receives a caption describing its content and a number which can be used to refer to it in the text of the document. The environments `figure` and `table` are intended for this purpose and explained in section 13.14.

New users often feel uncomfortable with letting LaTeX decide where to place their figures and tables and constantly tell LaTeX to place the float where they placed it in the source file (using the optional argument of the float environments). Of course, this completely defeats the purpose of the environment. Let go of the fear that it will end up in some inconvenient place (that will change any time you modify your document anyway) and just let it float (i.e. don't use the optional argument). At the very end, when everything else is done and you are checking your document for possible cosmetic improvements, some adjustments can be made, though placing the float "here" (where it is in the source) really never is the best option.

[12] https://betterposters.blogspot.com/2012/08/the-data-prison.html

If you really, *really* want some figure or table to exactly be at some place in the document, just don't put it into a float environment. A `center` environment is totally sufficient. If you are using KOMA-Script, you can also use the non-floating variants `figure-` or `table-`. In these cases, a caption and number are probably not necessary, but they can of course still be added. I can, however, only stress that this is virtually never the way to go in scientific and technical writing.

9.7 Cleanly Separate Text and Math Mode

Math mode and text mode are fundamentally different in a number of ways. Sometimes the differences in output are rather subtle in specific cases, sometimes there may be none at all. Still, especially because of the cases where the differences are subtle, it is important to cleanly separate math and text mode in the source.

Let us look at some examples.

```
\enquote{\(a, b, c\) and \(d\) are real numbers.} is wrong.\\
\enquote{\(a\), \(b\), \(c\) and \(d\) are real numbers.} is right.
```

"*a*, *b*, *c* and *d* are real numbers." is wrong.
"*a*, *b*, *c* and *d* are real numbers." is right.

Naturally, there are numbers that should be in text mode, like dates and years, but when a number is really part of a math expression, it should be in math mode even if there is no other math adjacent to it.

```
Leibniz published his calculus in~1684.
The variable~\(a\) has a value of~\(3\).
```

Leibniz published his calculus in 1684. The variable a has a value of 3.

Of course, numerical values, at least those that are not small integers (say, up to four digits), should be typeset using siunitx (see section 13.4). This conveniently eliminates the need to switch to or from math mode, since the siunitx macros behave identically in both. This is especially important when a decimal marker is involved.

```
\enquote{The golden ratio has a value of about~\(1.618\).} is bad.\\
\selectlanguage{ngerman}\sisetup{locale=DE}%
\enquote{Der Goldene Schnitt beträgt etwa~\(1,618\).} ist total falsch.\\
\enquote{Der Goldene Schnitt beträgt etwa~\num{1.618}.} ist richtig.
```

"The golden ratio has a value of about 1.618." is bad.
„Der Goldene Schnitt beträgt etwa 1, 618." ist total falsch.
„Der Goldene Schnitt beträgt etwa 1,618." ist richtig.

The converse case, where you want to write text in math mode, is to be treated analogously.

```
\[
  C(n) = \begin{cases}
    \frac{n}{2} & \textnormal{if \(n\) is even} \\
    3 n + 1     & \textnormal{if \(n\) is odd}
  \end{cases}
\]
```

$$C(n) = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ 3n + 1 & \text{if } n \text{ is odd} \end{cases}$$

9.8 Use the Correct Math Font

9.8.1 Indices

Normally, when you write a letter in math mode, it is typeset as a variable, i.e. in italic font without italic correction. Thus, `Vol` in math mode means "the variable `V` times the variable `o` times the variable `l`" and not "the variable `Vol`" or anything of the sort. Besides plain readability (cf. section 8.3), this is a good reason to write `V o l` instead of `Vol`.

When you don't actually mean the product mentioned above, as is often the case in indices, you need to switch to the upright (*roman*) font using `\mathrm{`⟨*expression*⟩`}`.

```
\(W_{Vol}\) is hideous and wrong.\\
\(W_{\mathrm{Vol}}\) is right.
```

W_{Vol} is hideous and wrong.
W_{Vol} is right.

This is true for single letter indices, too, of course. Whenever you use an index, you should be clear on if it denotes a variable or not. If it does, as in the n-th Fibonacci number f_n or the z-component of the magnetic field B_z, it should be typeset as such, i.e. in the default math font. When the index instead purely denotes the name of a variable, i.e. it indicates a word rather than a variable, then it should be typeset using `\mathrm`, like the Boltzmann constant k_B or the critical temperature T_c.

This is to be distinguished from actual text in math mode, which was discussed in the previous section.

9.8.2 Mathematical Constants

Mathematical constants (like e, i and π) are a special case. Ideally, these are typeset using upright letters, in contrast to the italic symbols (like e, i and π) that are used for variables and physical constants. With LuaLaTeX and unicode-math (cf. section 12.3.1), you can define macros that give you the upright symbols.

```
\protected\def\e{{\symup e}}
\protected\def\i{{\symup i}}
\protected\def\cpi{{\uppi}}
```

With pdfLaTeX, there is no upright version of π in most fonts, so the best you can do is using upright e and i.

```
\protected\def\e{{\mathrm e}}
\protected\def\i{{\mathrm i}}
```

If you really want the upright π but have to use pdfLaTeX for some reason, [upright-pi][13] outlines some options for getting one.

9.9 Manually Add Spacing in Math Mode When Helpful

Similarly to how spaces can help make math source code more readable (cf. section 8.3), adding additional spacing to the math output can make an expression, especially a long one, more easily readable. Adding small spaces or explicit multiplication signs can go a long way sometimes. The macros you can use for this are listed in section 13.7. Usually `\,` is enough, though. Here are some examples.

```
\begin{align*}
  \psi &= \sqrt{n_{\mathrm s}} \, \e^{\i \varphi} &
  \textnormal{rather than} &&
  \psi &= \sqrt{n_{\mathrm s}} \e^{\i \varphi} \\
  \dot\delta &= \frac{2 \cpi}{\Phi_0} \, U &
  \textnormal{rather than} &&
  \dot\delta &= \frac{2 \cpi}{\Phi_0} U \\
  \beta_C &= \frac{ 2 - (\cpi - 2) \, i_{\mathrm r} }{ i_{\mathrm r}^2 } &
  \textnormal{rather than} &&
  \beta_C &= \frac{ 2 - (\cpi - 2) i_{\mathrm r} }{ i_{\mathrm r}^2 } \\
  v_{\mathrm{ph}} &= \num{0.415(18)} \, c &
  \textnormal{rather than} &&
  v_{\mathrm{ph}} &= \num{0.415(18)} c
\end{align*}
```

$$\psi = \sqrt{n_\mathrm{s}}\,\mathrm{e}^{\mathrm{i}\varphi} \quad \text{rather than} \quad \psi = \sqrt{n_\mathrm{s}}\mathrm{e}^{\mathrm{i}\varphi}$$
$$\dot\delta = \frac{2\pi}{\Phi_0}\,U \quad \text{rather than} \quad \dot\delta = \frac{2\pi}{\Phi_0}U$$
$$\beta_C = \frac{2 - (\pi - 2)\,i_\mathrm{r}}{i_\mathrm{r}^2} \quad \text{rather than} \quad \beta_C = \frac{2 - (\pi - 2)i_\mathrm{r}}{i_\mathrm{r}^2}$$
$$v_\mathrm{ph} = 0.415(18)\,c \quad \text{rather than} \quad v_\mathrm{ph} = 0.415(18)c$$

In longer expression, a multiplication dot (which comes with extra spaces) is often better than just a small space.

The decision to insert a space or not is often rather subjective, however, and depends somewhat on the font you use. The main thing you should take away from this is that it

[13] https://tex.stackexchange.com/q/54166

is okay, even encouraged to manually insert spacing in math mode when this improves readability. Whenever you look at some math expression you wrote and think that some part of it is kind of hard to read, consider adding a space or more to remedy that.

10 Maintaining a Bibliographic Database

10.1 Ensuring Integrity

One easily finds citation information in BIB file format for many scientific publications online, right in the search engine, on the website of the publication or using dedicated software like JabRef. However, these BIB entries are often of poor quality, missing information or containing errors. A consequence of this is that, in order to have a high quality database of bibliographic data resulting in a consistent bibliography, you will need to check and groom every entry by hand. It is advisable to always do this at the moment of adding an entry to the database, as doing it in batch right before publication is a tedious and tiresome task.

You will usually find the relevant information on the website of the publication or in the publication itself. If you have access to a digital version of the source you are citing, I recommend saving it alongside the database. This allows you to easily access the full text of the source without having to go online and download it.

10.2 Using the Correct Entry Fields

The old system for producing bibliographies as well as the backend program preparing the entries for LaTeX are called BibTeX. For quite some time, the newer and better biblatex with its backend `biber` has been around and should be preferred. (Sadly, many journals still have not left the ancient times of BibTeX behind and require you to use it.) The most notable differences are a new interface for including the bibliography in the LaTeX document itself as well as additional entry types and fields. Section 13.16 lists the most important macros for creating a bibliography.

Some entry fields that had to be used with BibTeX are deprecated when using biblatex and should be avoided. Here is a list of the most common ones.

`year` and `month` were used to input the publication date of the entry. Use `date` instead. Input dates in the form `yyyy-mm-dd`, `yyyy-mm` or `yyyy`.

`journal` was used to input the title of the journal an article was published in. Use the newer `journaltitle` instead.

`school` was used to input the institution a thesis was done at. Use `institution` instead.

The old field names are still supported as aliases for the new ones, but it is advisable to just use the latter when you have a choice.

One case that often causes confusion is the difference between the entry fields `issue` and `number`. Scientific journals are traditionally published in *issues*, with all the issues of a year forming a *volume*. Some journals call their issues *numbers* instead. The naming used by the

respective journal is irrelevant for the field you should use when entering the information into your database, though.

- If the issue is identified using a numeric value, use the `number` field.
- Otherwise, i.e. if the issue is identified using words (e.g. "Spring Edition"), use `issue`.

There are also some new fields that did not exist before. Most notably `eid` (*electronic identifier*), which you should use for the article number when an article has one instead of page numbers. Oftentimes, you will find this number in the `pages` field when you download an entry from the web, which is wrong.

10.3 Correctly Inputting Data into the Fields

With biblatex and `biber`, you can input most data in the bibliography database exactly as you would write normal LaTeX code. As usual, special characters like "ä" and "é" can be used without problems but you have to escape characters that are special to TeX, like "$" and "%".

One field that is frequently used incorrectly is `author`. It expects a list of names, separated by `and`, where each name has the format `last names, first names`. For example, a valid field assignment may be

```
author = {Kamerlingh Onnes, Heike and Josephson, Brian David},
```

If you are ever confused about how to input certain data, have a look a the biblatex manual. It lists all the entry types and fields and describes what kind of input they expect.

10.4 Which Entry Fields to Provide

The biblatex manual describes in detail which entry types accept which entry fields and which of these are required. When set up correctly (cf. section 3.2), JabRef will also make this information apparent. However, some optional fields are more important than others. Most crucially, you should be consistent in which information you display in your bibliography. biblatex can always be configured to omit certain fields (e.g. I have it configured to only print the year of the date specified in `date`), but it can never print information that is not in the database.

The following are the fields I recommend always entering in your database. This should be fine for works in physics and adjacent fields. Customs may differ for other fields.

Journal Articles using the `@article` entry type

- Always include `author`, `title`, `journaltitle`, `date`, `volume`, `number`/`issue` and `pages`/`eid`.
- I strongly recommend also including the `doi`. While I would not print it in the bibliography itself, it allows biblatex to make the corresponding bibliography entry a hyperlink leading to the web page of the article. This makes it really convenient for readers of your document to quickly find the cited article. Additionally, it allows your database editor to provide that link as a convenience for you.
- I also like including the `abstract` in the database, so I can read it without having to actually open the article.

Books using the `@book` entry type
- Always include `author`, `title`, `date` and `publisher`.
- If the book has one, include `doi`, as motivated above. Otherwise, include `isbn`.

Online Resources using the `@online` entry type
- Always include `author/editor`, `title`, `date`, `url` and `urldate`.
- Note that `date` should contain the date of publication while `urldate` should contain the date you accessed the resource.

Any Entry
- If applicable, also include fields like `subtitle` and `origlanguage`.
- Try to use some common sense when deciding which fields to require. Which information is helpful in identifying and finding the resource you are referencing?

11 Document Classes

11.1 The Base Classes

There are three base document classes included in any LaTeX installation: article, report and book. They differ in a couple of default layout settings. The most important difference between them, however, is the sectioning commands they provide.

- article provides `\section`, `\subsection` and `\subsubsection` as well as `\paragraph` and `\subparagraph`. The latter two are not needed very often, though.
- report additionally provides `\chapter` (above `\section`).
- book additionally provides `\part` (above `\chapter`).

Accordingly, article is for short documents with closely related sections (e.g. papers, summaries or references), while report is for medium length documents with topically separated chapters (e.g. academic theses) and book is for long documents that need to be divided into even larger parts.

Use these differences to choose the right document class for your project. If you find yourself wanting to dramatically change the appearance of the headings (say you are using article and want every section to start on a new page or you need a `\subsubsubsection`) then you are probably using the wrong document class (just use report's sectioning commands from `\chapter` to `\subsubsection` instead).

11.2 KOMA-Script

While the three base classes provide some descent default configuration, they do not expose a lot of the layout decisions to the user in the form of options and commands. Many authors will wish to change these defaults, especially since the preset adheres to an American typographical tradition and differs somewhat from European customs. In order to perform adjustments, the user has to either directly modify the internal macros of the document classes or load packages that do just that.

The KOMA-Script bundle offers an alternative approach. It provides the document classes scrartcl, scrreprt and scrbook as replacements for the base classes. These have the same general properties as the respective base classes with some slightly different default layout choices, but they provide a multitude of options and commands to modify the appearance of the document and conveniently access additional functionality. Using KOMA-Script, one should not be needing any further packages to modify the general appearance of the document (type area, headers and footers, section headings, etc.). As a bonus, the default settings are somewhat more satisfying to the European eye (KOMA-Script's author is German) which means less necessity for adjustments in many cases.

The bundle also contains some additional document classes (most notably scrlttr2 for writing letters) as well as packages (most notably scrlayer-scrpage) that can be used with the KOMA-Script classes as well as with others. Though studying the entire extensive manual [koma-script] is not necessary, chapters 2, 3 and 5 are probably a good read for any author using KOMA-Script.

11.3 Other Document Classes

There are, of course, many more document classes to choose from. I want to mention only some of them here.

- memoir is a document class that aims to replace all three base classes at once (mostly report and book, though) with many additional options. While I have personally never used it and, accordingly, can't help with questions regarding it, it is certainly a large improvement to the base classes.
- moderncv is a document class for writing curricula vitae with a pretty slick look. Sadly, it does not have a manual (yet), but it comes with some examples one can use to get started.
- beamer lets you create presentations using LaTeX. It is based on PGF (in fact, it has the same author), provides capabilities for easily creating overlays and comes with many prepared layouts. Make sure to schedule some time for reading in its excellent documentation before starting to work on your first presentation.

12 Packages

In the decades since the inception of LaTeX, thousands of authors have written macros for all kinds of tasks and published them in the form of packages. The place where all of the public packages are collected is the *Comprehensive TeX Archive Network*[14] (*CTAN*). Both MiKTeX and TeX Live include most of them, but you can also easily install additional packages manually.

You will, of course, never need many of these. I will give an incomplete list of frequently used ones here. If you want to do something more complex than simple text formatting that is not covered by these packages, ask your local LaTeX sage and search online before attempting to write the macro yourself. Chances are, there is a package for that.

[14] https://ctan.org/

12.1 Always Include These

Irrespective of the engine you are using, you should also always load the following packages.

- babel for localization. This package provides language selecting and switching capabilities and, depending on the language, defines some useful shorthands. You should always give the main language of your document as an option to your document class, not only to babel. This will ensure that other packages that care about the language also get it. You really only need to tell babel directly about document languages if there are more than one.

- microtype; the documentation's subtitle really says it all: *Subliminal refinements towards typographical perfection*. Just load it, you don't have to do anything more, and your document will be prettier with many fewer bad boxes.

- nowidow to avoid widows and orphans (cf. section 9.4). It allows you to select how many lines of a paragraph TEX has to keep together when breaking it across pages. I usually load it with the following options:

```
\usepackage[defaultlines=2, all]{nowidow}
```

If you are compiling with pdfLATEX, you should also load the following packages.

- fontenc with the font encoding T1.

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

- textcomp, which provides access to additional glyphs in text mode. Importantly, `\textmu` ("μ"), which is needed for typesetting units such as μm, is only available after loading textcomp.

- Depending on the PDF viewer, there may be some issues with copying or searching through text with ligatures. This can be remedied by adding the following lines to the preamble:

```
\input{glyphtounicode}
\pdfgentounicode=1
```

- After selecting the T1 font encoding, you need to load a compatible font. If you just want to use the default font (it is a good font), load lmodern. If you want to select a different one, see section 12.3.

When compiling with LuaLATEX (or XELATEX), these packages are not necessary. In fact, you will get errors if you try to load them with those engines.

12.2 Programming

For modifying or writing your own macros, etoolbox provides a collection of useful wrapper macros to take advantage of the ε-TEX primitives. If you seriously want to develop new macros or even packages, you are probably better off using the LaTeX3 programming layer (cf. section 14.6) but it may still be useful when quickly modifying some LaTeX 2ε code.

12.3 Font Selection

12.3.1 With LuaLaTeX

With LuaLaTeX (or XeLaTeX), you have the possibility to load OTF fonts, so you pretty much use any font you have installed on your machine or for which you can find an OTF file. The interface for this is provided by the fontspec package for text and unicode-math for math fonts. I recommend loading unicode-math with the `math-style=ISO` option.

```
\usepackage[math-style=ISO]{unicode-math}
```

Keep in mind that, if you need math expressions, your math font should match your text font. Since choosing fonts is hard, I recommend sticking with math and text fonts that come together. The *The LaTeX Font Catalogue*[15] is a great resource for this. It lists fonts available in LaTeX organized into categories with text examples.

Many fonts are still available as packages that load the desired fonts using the appropriate packages and options. For example, you can load the STIX 2 fonts used in this document just by loading the stix2 package. It will detect the use of LuaLaTeX and internally load stix2-otf, not stix2-type1, in order to set up the fonts.

The default TEX font, designed by Knuth himself, is called Computer Modern, its scalable clone is called Latin Modern and still the default with fontspec if you don't select anything explicitly. It is a very good font for scientific and technical writing, especially because it contains all the important math fonts. If you don't really care about your document's font, just stick with this one.

12.3.2 With pdfLaTeX

Because LaTeX is old, older than formats like TTF or OTF, you cannot just load a system font in pdfLaTeX like in other software. The process of preparing a font for use in LaTeX is a bit involved, so it's better to stay out of that and stick with the fonts that other people prepared for you. If you really want to use one of your system fonts, use LuaLaTeX instead (see above).

Especially here, *The LaTeX Font Catalogue* is a good place to search for a good font for your document. Have a look and pick a font you like, but keep in mind that you should pick one with math support if you use any math. If you want to stick with Latin Modern, load the lmodern package in order to get its T1 version.

[15] https://tug.org/FontCatalogue/

12.4 Text

- xcolor provides extended color handling capabilities. If you want to use colors, load it.
- hyperref is for creating hyperlinks, inside of your document as well as to other files or websites. Just loading it turns all internal references in your document into hyperlinks. I usually load it with the options `hidelinks` (which suppresses the box that is displayed around links by default) and `bookmarksnumbered` (which includes sectioning numbers in the generated PDF bookmarks).

  With very few exceptions, hyperref should be the *last* package you load!
- array gives you an improved version of LaTeX's `tabular` and `array` environments. If you have any of those, load it.
- booktabs provides the commands needed to create beautiful tables (cf. section 9.5).
- enumitem extends the list environments with many useful options and lets you easily define new or modify the default ones.
- siunitx is the package for typesetting numbers and units. If you only ever need integers below 1000 and no units, you're fine. Otherwise, use this package! You should specify a typographic convention using the `locale` option.

```
\usepackage[locale=DE]{siunitx}
```

- mhchem lets you easily typeset chemical formulas. You have to specify a version when including it, at the time of writing the current version is 4.

```
\usepackage[version=4]{mhchem}
```

- csquotes provides context sensitive quotation commands. Whenever you want to set anything in quotation marks, use this package.
- biblatex is the package to use for citations and bibliographies. Do *not* use BibTeX, biblatex's predecessor; it is outdated and deprecated.
- glossaries and its extension glossaries-extra can be used to create glossaries. Sadly, its documentation can be a bit confusing and it can be hard to find what you are looking for, but it is unrivaled in functionality.
- acro is a slim alternative to glossaries for the case of simple lists of acronyms. For English texts, this may be the better choice if you are unfamiliar with glossaries.

  If you are writing in German, everything gets much more complicated due to declension. I created a system to handle this using glossaries-extra for my master's thesis, but it's a bit involved. If you are using a lot of abbreviations in a German document and are interested in this, I can show you though.

- verbatim provides an improved reimplementation of LaTeX's `verbatim` and `verbatim*` environments. If you need these, load it.

12.5 Graphics Inclusion and Creation

pdfLaTeX and its descendants can include images in the PDF, PNG and JPG formats. SVG files can also be conveniently included via the PDF route with the added feature of LaTeX typesetting the text portion of the drawing. Furthermore, you can create vector drawings yourself directly with LaTeX.

Here's the packages you should be aware of for this:

- graphicx enhances LaTeX's native graphics inclusion capabilities. Whenever you want to include an image, load this package.
- svg provides a system to automatically export SVG drawings to PDF and TEX files using Inkscape and include those in your document. [svg-howto][16] explains how to set it up.
- tikz (stylized as Ti*k*Z) is a simple but powerful interface for creating drawings with LaTeX. It is the basis for many other packages that create some kind of drawing. Its back end is called PGF.
- tcolorbox is the package to go to for creating (colored) boxes. This is also useful for creating posters in LaTeX.

12.6 Mathematics

Some of the packages mentioned above also affect math mode macros and environments (e.g. array) or provide commands that can be used both in text and in math mode (e.g. siunitx). Additionally to those, the following packages are often used.

- amsmath (by the American Mathematical Society, $\mathcal{AMS}$) brings many enhancements to standard math typesetting. You should load it (or mathtools, see below) whenever you have any math content in your document. (Other packages from the $\mathcal{AMS}$ family, like amssymb and amsthm, may also be useful.)
- mathtools further enhances amsmath and standard LaTeX capabilities. It automatically loads amsmath, so you don't have to do it if you use mathtools (and don't want to pass any options to amsmath).

If you use pdfLaTeX, the following packages are often needed.

- bm (for *bold math*) provides access to bold math symbols. There are other ways to get bold math, but I recommend this one. (With unicode-math, you can just use `\symbf`.)
- dsfont gives you access to double stroke (*blackboard*) math letters (like "$\mathbb{Z}$") when using pdfLaTeX. (With unicode-math, you can just use `\symbb`.)

[16] `https://tex.stackexchange.com/a/523685`

12.7 Code Listings

If you need to print source code listings, like I regularly do in this document, and you want capabilities exceeding those of the `verbatim` environment (like syntax highlighting or line numbers), there are two packages you may choose from.

- listings is a syntax highlighter implemented purely in LaTeX. It requires no setup besides loading the package and its functionality is sophisticated enough for most simple codes listing needs. For obscure languages, you will have to define the highlighting scheme yourself, which is rather easy, though. For more advanced highlighting schemes, however, you may find certain limitations.
- minted uses *Python*'s[17] powerful *Pygments*[18] library for syntax highlighting, with over 500 languages and many color schemes predefined. This means a bit more setup, as you need to install Python with Pygments and enable shell escape, but it can handle more edge cases and may more easily give you the desired results if you are already familiar with Python and Pygments.

12.8 Other

- geometry lets you explicitly set the dimensions defining the page layout. Especially when using KOMA-Script, you should not do this! It is really only okay when you have to follow rules that demand a bad layout (e.g. 2 cm border on all sides).
- scrlayer-scrpage belongs to the KOMA-Script bundle and allows you to set up complex page layouts. Most of the time, you only want to modify the content of your page headers and footers, though.
- pdflscape allows you to rotate single pages of your document. In contrast to the lscape package it is based on, the pages appear rotated in PDF viewers.
- rotating allows you to rotate single objects. Most notably it provides the rotated floating environments `sidewaystable` and `sidewaysfigure`.
- caption is for customizing captions of floating environments. While KOMA-Script has those capabilities as well (and you should use the KOMA-Script options rather than the ones provided by caption when using both), you also want caption's `hypcap` option. This option makes hyperlinks to, say, a figure point to the top of the figure instead of its caption. If you have any floating environments in your document, I recommend loading caption with this option.

```
\usepackage[hypcap]{caption}
```

- subcaption provides macros and environments for typesetting subfloats (e.g. subfigures) and plays well with caption (it is by the same author).

[17] `https://python.org/`
[18] `https://pygments.org/`

13 Important Macros and Environments

Here is a list of the macros and environments that you are likely to need when writing documents using LaTeX. It is not complete in any way. If you feel that I left out any important ones, feel free to let me know.

13.1 Font Switching

The following macros are font switches, i.e. they change the font immediately until the end of the current group.

`\normalfont` switches to the document's default font.

`\rmfamily`, `\sffamily`, `\ttfamily` switch to the roman (serif), sans serif and typewriter font (monospace; German *dicktengleich*), respectively.

`\upshape`, `\itshape`, `\slshape`, `\scshape` switch to upright, italic, slanted and small capital letters, respectively.

`\mdseries`, `\bfseries`, `\lfseries` switch to normal (medium), bold and light font weight, respectively.

Note that many fonts do not support all (possible combinations) of these.

There are commands that take the text to be typeset as an argument for every one of these switches: `\textnormal`, `\textrm`, `\textsf`, `\texttt`, `\textup`, `\textit`, `\textsl`, `\textsc`, `\textmd`, `\textbf` and `\textlf`. These basically have the same effect as using a group with the appropriate switch at the beginning, but additionally take care of italic correction where appropriate.

```
King {\itshape George III} had a crown. \\
King \textit{George III} had a crown.
```

King *George III* had a crown.
King *George III* had a crown.

As a rule, it is better to use the `\textxx` commands when switching inside of a paragraph, usually for just a few words. When switching for whole paragraphs or the content of an environment, go with the switches.

Remember that you should not use these commands directly in your document, but only in definitions in your preamble (cf. section 5.2)! A very common use for switching fonts is to *emphasize* a word or phrase, which is usually done using the italic font. LaTeX already provides a semantic command for that: `\emph{`⟨*text*⟩`}`. It is constructed such that it usually switches to italic font, but instead switches to upright font if the surrounding text is already italic.

13.2 Counters and Numbering

LaTeX counters are used to, well, count things. They are basically like an integer variable in other programming languages and can take values ranging from -2^{31} to $2^{31}-1$. You can manipulate them with the following macros.

`\newcounter{`⟨*name*⟩`}` creates a new counter called ⟨*name*⟩. It initially has the value 0.

`\stepcounter{`⟨*name*⟩`}` increases the value of the counter ⟨*name*⟩ by 1.

`\addtocounter{`⟨*name*⟩`}{`⟨*incr*⟩`}` increases the value of the counter ⟨*name*⟩ by ⟨*incr*⟩.

`\setcounter{`⟨*name*⟩`}{`⟨*value*⟩`}` sets the value of the counter ⟨*name*⟩ to ⟨*value*⟩.

All assignments performed by these macros are global, i.e. not confined to the current TEX group!

Typical applications for counters are numbering sections, figures, or footnotes. The values of these document elements are stored in the counters `section`, `figure` and `footnote`. When these numbers are printed, you may desire a different format than just printing the Arabic digits. For this purpose, the following commands are provided.

`\arabic{`⟨*name*⟩`}` prints an Arabic numeral: 1, 2, 3, 4, 5, 6, 7, 8, 9

`\roman{`⟨*name*⟩`}` prints a lower case Roman numeral: i, ii, iii, iv, v, vi, vii, viii, ix

`\Roman{`⟨*name*⟩`}` prints an upper case Roman numeral: I, II, III, IV, V, VI, VII, VIII, IX

`\alph{`⟨*name*⟩`}` prints a lower case Latin letter: a, b, c, d, e, f, g, h, i

`\Alph{`⟨*name*⟩`}` prints an upper case Latin letter: A, B, C, D, E, F, G, H, I

`\fnsymbol{`⟨*name*⟩`}` prints one of nine footnote symbols: ∗, †, ‡, §, ¶, ‖, ∗∗, ††, ‡‡

If you want to use the value of a counter at a place where TEX expects a number, you can use `\value{`⟨*name*⟩`}`.

The formatting for the counter `foo` is usually stored in a macro called `\thefoo`. For example, the formatted section number is typeset by the macro `\thesection`. If you want to change that format, say you want capital letters, you can redefine this macro.

```
\renewcommand*\thesection{\Alph{section}}
```

For the format of page numbers the macro `\pagenumbering` is provided which changes the numbering format and additionally resets the page number to 1. So, if you want to change the page numbering to lower case Roman numerals starting at i, you just have to say

```
\pagenumbering{roman}
```

13.3 Sectioning

For sectioning, the commands `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph` and `\subparagraph` are provided, where `\part` is only available in book (or scrbook) and `\chapter` in report and book (scrreprt and scrbook). They differ in formatting and behavior, but their syntax is the same. We shall explain it using `\section`:

`\section[`⟨*short title*⟩`]{`⟨*title*⟩`}` creates a section heading with the title ⟨*title*⟩. If the optional ⟨*short title*⟩ is given, it is used instead of ⟨*title*⟩ in the table of contents and for running titles.

Every sectioning level has a corresponding *depth value*, where sections have the depth 1. Accordingly, paragraphs have the depth 4 and parts have the depth −1. These depths are used to specify which sectioning levels should be numbered and which should appear in the table of contents using the following counters.

`secnumdepth` specifies the deepest sectioning level that should be numbered.

`tocdepth` specifies the deepest sectioning level that is to be included in the table of contents.

So, if you wanted to exclude everything below sections from the table of contents, you could set

```
\setcounter{tocdepth}{1}
```

13.4 Numbers and Units

Whenever you need to typeset numerical values other than just small integers, or when you need to typeset units, you should use siunitx! It provides the following commands (and some more handy shortcuts).

`\num[`⟨*options*⟩`]{`⟨*value*⟩`}` prints numbers.

```
\num{1234567890.0987654321} \\
\num{-3.55e-12} \\
\sisetup{locale=DE}
\num{2.656624(13)e8} \\
\num[separate-uncertainty]{2.54(8)e-3}
 ↪  \\
\complexnum{1 +- 2i}
```

1 234 567 890.098 765 432 1
$-3.55 \cdot 10^{-12}$
$2{,}656\,624(13) \cdot 10^{8}$
$(2{,}54 \pm 0{,}08) \cdot 10^{-3}$
$1 \pm 2\mathrm{i}$

`\unit[`⟨*options*⟩`]{`⟨*unit*⟩`}` prints units.

```
\unit{\kilo\gram\meter\per\square\second} \\
\unit[per-mode=fraction]
  {\kilo\gram\square\meter
   \per\ampere\per\square\second} \\
\sisetup{per-mode=symbol}
\unit{\milli\joule\per\ampere\per\second}
```

$\mathrm{kg\,m\,s^{-2}}$
$\frac{\mathrm{kg\,m^2}}{\mathrm{A\,s^2}}$
$\mathrm{mJ/(A\,s)}$

`\qty[`⟨*options*⟩`]{`⟨*value*⟩`}{`⟨*unit*⟩`}` prints a value with a unit.

```
\(\mu_0 = \qty{1.25663706212(19)e-6}{\newton\per\square\ampere}\),
furthermore, you should wait for pretty much exactly
\qty[separate-uncertainty]{8.77654(23)}{\nano\second}
before proceeding. By the way, my new room is
\qtyproduct{5 x 3}{\meter} large.
```

$\mu_0 = 1.256\,637\,062\,12(19) \cdot 10^{-6}\,\mathrm{N\,A^{-2}}$, furthermore, you should wait for pretty much exactly $(8.776\,54 \pm 0.000\,23)\,\mathrm{ns}$ before proceeding. By the way, my new room is $5\,\mathrm{m} \times 3\,\mathrm{m}$ large.

`\ang[`⟨*options*⟩`]{`⟨*value*⟩`}` prints angles.

```
\ang{1.234} \\
\ang{-3;5;8}
```

$1.234°$
$-3°5′8″$

As the format of numbers and units depends on the siunitx configuration, you may get a different output than in these examples. I loaded the package with the following options (replace × by ·, use US locale, use × when printing products):

```
\let\xtimes\times
\let\times\cdot
\usepackage[
  locale=US,
  product-symbol=\xtimes,
  ]{siunitx}
```

If you need any units that are not predefined by siunitx or want to define a shorthand for longer units, you can do so using `\DeclareSIUnit[`⟨*options*⟩`]{`⟨*unit*⟩`}{`⟨*symbol*⟩`}`.

```
\DeclareSIUnit\dBm{dBm}
```

For typesetting numeric values in tables, please see section 13.12.

13.5 Chemical Symbols

The mhchem package provides `\ce`, which can be used to easily typeset chemical formulas in both text and math mode.

```
The compound \ce{YBa2Cu3O_{7-$x$}}
is a high-temperature superconductor.
\[
  \ce{C6H12O6 -> 2 C2H5OH + 2 CO2}
\]
```

The compound $YBa_2Cu_3O_{7-x}$ is a high-temperature superconductor.

$$C_6H_{12}O_6 \longrightarrow 2\,C_2H_5OH + 2\,CO_2$$

13.6 Display Math Environments

There are a number of display math environments and it is important to use the right one for the task at hand. All of the environments presented here are provided by amsmath, other display math environments present in LaTeX 2_ε should not be used.

The following environments are intended to display single equations.

`equation` is the most basic one and displays one centered equation.

```
\begin{equation}
  a^b - b^a = 1
\end{equation}
```

$$a^b - b^a = 1 \tag{1}$$

`multline` is for equations that are too wide for one line. Use `\\` to insert line breaks. The first line in a `multline` will be left aligned, the last one right aligned and all the lines in between will be centered.

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

$$\begin{aligned} 0 = a + b + c + d + e + f + g + h + i + j + k + l + m + n + o \\ + p + q + r + s + t + u + v + w + x + y + z \end{aligned} \tag{2}$$

For displaying multiple equations at once, the following environments can be used.

`gather` displays several equations (separated by `\\[`⟨*dimension*⟩`]`), each of them centered independently. As in `tabular`, the optional argument of `\\` can be used to sometimes insert extra (possibly negative) space between equations. The last equation must *not* be terminated with `\\`.

```
\begin{gather}
  a^p \equiv a \pmod{p} \\
  a^n + b^n \neq c^n
\end{gather}
```

$$a^p \equiv a \pmod{p} \tag{3}$$

$$a^n + b^n \neq c^n \tag{4}$$

`align` does the same, but you can specify alignment points using `&`. Think of this environment like ta table with alternating `r` and `l` columns. Every pair of columns forms an equation, the equations are spaced out equally on the line. The `&` inside an equation must be placed *before* the character you want to align at, which should usually be a binary relation like `=`.

```
\begin{align}
  a + b &= c &
  c + e &= f \\
  g &= h + i &
  j &= k + l
\end{align}
```

$$a + b = c \qquad c + e = f \tag{5}$$
$$g = h + i \qquad j = k + l \tag{6}$$

`aligneat` is like `align`, but it takes the number of equations in one row as an argument and does not space out the equations horizontally, allowing you to specify that space yourself.

```
\begin{alignat}{2}
  a + b &= c &\quad
  c + e &= f \\
  g &= h + i &
  j &= k + l
\end{alignat}
```

$$a + b = c \qquad c + e = f \tag{7}$$
$$g = h + i \qquad j = k + l \tag{8}$$

`flalign` is like `align`, but it spaces the equations out to the edges of the line.

```
\begin{flalign}
  a + b &= c &
  c + e &= f \\
  g &= h + i &
  j &= k + l
\end{flalign}
```

$$a + b = c \qquad c + e = f \tag{9}$$
$$g = h + i \qquad j = k + l \tag{10}$$

There are starred variants of all of these environments that do the same thing, but with equation numbering turned off. The shortcut `\[ ... \]` is equivalent to the environment `\begin{equation*} ... \end{equation*}`.

Inside all of these environments, except for `multline` and `multline*`, the following may be used.

`split` is for single equations that need to be broken across several lines with alignment.

```
\begin{equation}
  \begin{split}
    0 = a &+ b + c + d + e + f + g + h + i + j + k + l + m + n + o \\
      &+ p + q + r + s + t + u + v + w + x + y + z
  \end{split}
\end{equation}
```

$$\begin{aligned} 0 = a + b + c + d + e + f + g + h + i + j + k + l + m + n + o \\ + p + q + r + s + t + u + v + w + x + y + z \end{aligned} \tag{11}$$

`\tag{`⟨*text*⟩`}` may be used to manually specify the tag used for the current equation, either to replace the automatic equation number or to tag an equation that would otherwise be untagged.

`\notag` may be used to leave an equation untagged that would otherwise be tagged.

```
\begin{gather}
  a + b = c \\
  \tag{foo}
  d + e = f \\
  \notag
  g + h = i
\end{gather}
```

$$a + b = c \tag{12}$$
$$d + e = f \tag{foo}$$
$$g + h = i$$

`\intertext{`⟨*text*⟩`}` may be used to insert text between two equations (without losing alignment). It may only be used directly following a `\\`.

`\shortintertext{`⟨*text*⟩`}` (provided by mathtools) is like `\intertext` but uses less vertical space. It is useful if the ⟨*text*⟩ to be inserted is just a few words long.

```
\begin{align}
  a + b &= c \\
\shortintertext{and also}
  d &= e + f .
\end{align}
```

$$a + b = c \tag{13}$$
and also
$$d = e + f. \tag{14}$$

Additionally, the following environments are available to be used in math mode. While they work in inline math mode, too, they are usually only appropriate in one of the environments listed above.

`cases` is for, well, listing different cases, as is commonly done in mathematics.

```
\begin{equation}\label{eq:little-fermat}
  a^{p - 1} \bmod p
  = \begin{cases}
    0 & \textnormal{if \(p \divides a\)} \\
    1 & \textnormal{otherwise}
  \end{cases}
\end{equation}
```

$$a^{p-1} \bmod p = \begin{cases} 0 & \text{if } p \mid a \\ 1 & \text{otherwise} \end{cases} \tag{15}$$

`gathered`, `aligned` and `alignedat` are like the variants without `ed`, but for use inside the environments above. Their width is the natural width of their content as opposed to the full width of the line.

```
\begin{equation}
  \left.
  \begin{aligned}
    (a + b)^2 &= a^2 + 2 a b + b^2 \\
    (a - b)^2 &= a^2 - 2 a b + b^2 \\
    (a + b) (a - b) &= a^2 - b^2
  \end{aligned}
  \quad \right\} \qquad \textnormal{leges binomiales}
\end{equation}
```

$$\left.\begin{aligned} (a+b)^2 &= a^2 + 2ab + b^2 \\ (a-b)^2 &= a^2 - 2ab + b^2 \\ (a+b)(a-b) &= a^2 - b^2 \end{aligned}\right\} \qquad \text{leges binomiales} \tag{16}$$

In order to get an overview of the macros and environments available for mathematical typesetting, it is worthwhile to at least skim through the amsmath and mathtools documentations.

13.7 Spaces in Math Mode

You can manually add spaces in math mode (cf. section 9.9) using the following macros.

`\,` adds a small math skip.
`\:` adds a medium math skip.
`\;` adds a thick math skip.
`\quad` adds a skip of 1 em width.
`\qquad` adds a skip of 2 em width.

13.8 Quotations

LaTeX 2_ε comes with the following two basic quoting environments predefined.

`quote` indents the contained text on both sides. Paragraphs are separated by a vertical space.

`quotation` indents the contained text on both sides. First lines of paragraphs are indented.

These environments do not add any quotation marks to the text.

While it is of course possible to manually insert quotation marks in your text, you should never do that. Instead, use the capabilities of the csquotes package. It provides context sensitive quotation commands and environments, i.e. it automatically switches between outer and inner quotation marks and chooses the right ones for your language. The basic commands are `\enquote{`⟨*text*⟩`}` for regular quotations and `\foreignquote{`⟨*language*⟩`}{`⟨*text*⟩`}` for quotations in foreign languages (using babel). With the package option `autostyle`, foreign language citations receive the quotation marks appropriate for their language.

```
\enquote{\enquote{I am English.}, said the man.} \\
\foreignquote{ngerman}{\enquote{Ich bin deutsch.} sagte der Mann.} \\
\foreignquote{spanish}{\enquote{Soy español.}, dijo el hombre.} \\
\foreignquote{norsk}{\enquote{Jeg er norsk.} sa mannen.}
```

“‘I am English.’, said the man.”
„‚Ich bin deutsch.‘ sagte der Mann.“
«“Soy español.”, dijo el hombre.»
«‹Jeg er norsk.› sa mannen.»

The package provides a number of additional quotation commands and environments, like `\textquote`, `\blockquote` and `displayquote`, as well as some auxiliary commands for ellipses, insertions and deletions (`\textelp`, `\textins`, `\textdel`). Have a look at the manual for details.

13.9 Verbatim Text

Sometimes, you want to print some text exactly as in the source file, without any macro expansion or other TEX functionality. This verbatim output can be achieved with the following commands. The generated text is printed in typewriter font.

`\verb`⟨*char*⟩⟨*content*⟩⟨*char*⟩ is for inline verbatim text. The ⟨*char*⟩ may be any character (well, almost, non-native characters like ä, é or ß won't work) that does not appear in ⟨*content*⟩. For example, `\verb|\foo{bar}|` prints “`\foo{bar}`”.

The starred version of this macro does the same thing, but it prints visible spaces, i.e. it outputs the character ␣ instead of a space. For example, `\verb|foo bar|` prints “`foo bar`”, but `\verb*|foo bar|` prints “`foo␣bar`”.

`verbatim` does the same but as an environment. Everything between `\begin{verbatim}` and `\end{verbatim}` is printed as is.

```
This is some normal text.
The usual \TeX\ rules apply.
\begin{verbatim}
This is verbatim text.
Normal \TeX\ rules do not apply.
\end{verbatim}
This is more normal text.
```

This is some normal text. The usual TEX rules apply.

```
This is verbatim text.
Normal \TeX\ rules do not apply.
```

This is more normal text.

`verbatim*` does the same thing, but it prints visible spaces.

13.10 Footnotes

`\footnote[`⟨*number*⟩`]{`⟨*text*⟩`}` produces a footnote mark and places ⟨*text*⟩ as a footnote at the bottom of the page. If ⟨*number*⟩ is given it is used for the footnote mark, otherwise the counter `footnote` is incremented and used.

Sometimes it is useful or necessary to place the footnote mark and text separately. This way you can place footnotes in places where `\footnote` is not allowed, e.g. in tables. It may also be useful if you want to place several marks referring to the same footnote or if you just don't want to write the whole footnote in the middle of a sentence for clarity. This can be done with the following commands.

`\footnotemark[`⟨*number*⟩`]` produces only the mark.

`\footnotetext[`⟨*number*⟩`]{`⟨*text*⟩`}` produces only the footnote itself.

The format of the footnote mark is controlled by `\thefootnote`. If you want symbols (∗, †, ‡) instead of numbers (1, 2, 3), you can redefine it like this (cf. section 13.2):

```
\renewcommand*\thefootnote{\fnsymbol{footnote}}
```

13.11 Lists

There are three basic list types in LaTeX.

`itemize` for bullet lists (items are labeled with symbols),

`enumerate` for enumerated lists (items are labeled with numbers) and

`description` for descriptive lists (items are labeled with text).

All of them use the same macro to indicate a new item.

`\item[`⟨*label*⟩`]` starts a new item and needs to be the first thing in any list. If ⟨*label*⟩ is given, it is used as the label, otherwise the default label is used (in `enumerate`, an appropriate counter is stepped first). Items in `description` lists should always use the optional argument.

Lists can be nested up to four levels deep. `itemize` and `enumerate` use different labels for every level. You can change the default labels by redefining the macros

- `\labelitemi` through `\labelitemiv` for `itemize`,
- `\theenumi` through `\theenumiv`, `\labelenumi` through `\labelenumiv` as well as `\p@enumi` through `\p@enumiv` for `enumerate` and
- `\descfont` for `description`.

If you want more control over the appearance of your lists, especially of the spacing, you should load the enumitem package. It provides commands for cloning and modifying the default lists and makes all the list environments accept an optional argument for changing these settings on an individual basis. For more details, have a look at the package documentation and maybe [enit-hspacing].

13.12 Tables

Tabular material, i.e. material organized into rows and columns, can be typeset using the `tabular` environment (text mode) and the `array` environment (math mode). The array package provides an improved implementation of both and its use is assumed in the remainder of this section. Since `tabular` and `array` are basically equivalent in use but the former is much more common, I will only describe the use of `tabular` here.

The `tabular` environment takes one optional and one mandatory argument, like this: `\begin{tabular}[`⟨*position*⟩`]{`⟨*column specification*⟩`}`

⟨*position*⟩ can be `t` for top alignment, `b` for bottom alignment or be omitted for middle alignment of the table.

⟨*column specification*⟩ should contain one letter for each column in the table. By default, the column types `l`, `c` and `r` for left, center and right aligned single line columns as well as `p{`⟨*width*⟩`}`, `m{`⟨*width*⟩`}` and `b{`⟨*width*⟩`}` for top, middle and bottom aligned fixed width columns (justified with automatic line breaks) are provided.

Columns are separated by `&`, rows by `\\[`⟨*dimension*⟩`]`. The optional argument of `\\` can be used to sometimes add extra vertical space between rows.

```
\begin{tabular}{l c r p{3em} m{3em} b{3em}}
  left & center & right & par & mid & bot \\
  x & x & x & x & x & x \\
  bla bla bla & bla bla bla & bla bla bla &
  bla bla bla bla bla & bla bla bla bla bla & bla bla bla bla bla \\
\end{tabular}
```

left	center	right	par	mid	bot
x	x	x	x	x	x
					bla bla
				bla bla	bla bla
bla bla bla	bla bla bla	bla bla bla	bla bla	bla bla	bla
			bla bla	bla	
			bla		

Additionally, the following characters may appear in the ⟨*column specification*⟩:

`|` inserts a vertical line. Don't use it (see below).

`@{`⟨*stuff*⟩`}` inserts ⟨*stuff*⟩ between the adjacent columns in every row instead of the default space of `2\tabcolsep` width. Use `@{}` to remove the column separation.

`!{`⟨*stuff*⟩`}` inserts ⟨*stuff*⟩ between the adjacent columns in every row in addition to the default space of `\tabcolsep` width on either side. `!{}` won't change anything.

`>{`⟨*stuff*⟩`}` inserts ⟨*stuff*⟩ at the start of every cell in the successive column.

`<{`⟨*stuff*⟩`}` inserts ⟨*stuff*⟩ at the end of every cell in the preceding column. Notably, `>` and `<` can be used to switch to and from math mode in every cell of a `tabular` column.

`*{`⟨*n*⟩`}{`⟨*column spec*⟩`}` repeats the ⟨*column spec*⟩ (which can contain everything the full specification can) ⟨*n*⟩ times.

```
\begin{tabular}{c @{--} c @{\hspace{1cm}} c !{--} c
                >{X}c c<{X} >{\(}c<{\)}
                *{3}{>{\color{teal}}c}}
  x & x & x & x & x & x & x & x & x & x \\
  a & b & c & d & e & f & g & h & i & j \\
\end{tabular}
```

x–x x – x Xx xX *x* x x x
a–b c – d Xe fX *g* h i j

Sometimes a fixed width column with non-justified text is desired. For this, you can define the following additional column specifiers (essentially taken from [ragged-colspec][19]).

```
\newcolumntype{L}[2]{>{\raggedright\let\newline\\\arraybackslash\strut}#1{#2}}
\newcolumntype{C}[2]{>{\centering\let\newline\\\arraybackslash\strut}#1{#2}}
\newcolumntype{R}[2]{>{\raggedleft\let\newline\\\arraybackslash\strut}#1{#2}}
```

They take two arguments, e.g. `L{`⟨*prototype*⟩`}{`⟨*width*⟩`}`, where the ⟨*prototype*⟩, that determines the vertical alignment of the column cells, should be one of `p`, `m` or `b`.

As explained in section 9.5, you should not use many rules in your table and no vertical ones at all. The only rules you should ever need are `\toprule`, `\midrule`, `\bottomrule` and sometimes `\cmidrule`. The rest of the structure of your table can be expressed by careful alignment of its cells.

```
\begin{tabular}{l l c c}\toprule
  Branch      & Language  & Word for Bread & Word for Water \\\midrule
  Germanic    & English   & bread          & water          \\
              & German    & Brot           & Wasser         \\
              & Norwegian & brød           & vann           \\
  Romance     & Spanish   & pan            & agua           \\
              & French    & pain           & eau            \\
  Finno-Ugric & Estonian  & leib           & vesi           \\
  Slavic      & Slovak    & chlieb         & voda           \\\bottomrule
```

[19] https://tex.stackexchange.com/a/12712

```
\end{tabular}
```

Branch	Language	Word for Bread	Word for Water
Germanic	English	bread	water
	German	Brot	Wasser
	Norwegian	brød	vann
Romance	Spanish	pan	agua
	French	pain	eau
Finno-Ugric	Estonian	leib	vesi
Slavic	Slovak	chlieb	voda

If you want to list numeric values in a table, use siunitx' `S` column specifier. You need to tell siunitx how much space to allocate for your numbers in every column via the option `table-format`, either using `\sisetup` or in the optional argument of `S`. Specify the `table-format` by writing a number, but instead of the digits of a part of the number you write the maximum number of digits in that part. For example `table-format=-3.2` would allocate space for three digits before and two digits after the decimal point in addition to space for a sign. `table-format=1.3(1)e2` would allocate space for one digit before and three after the decimal point, one digit of uncertainty and two digits in the exponent.

```
\begin{tabular}{c S[table-format=2.3]
                  S[table-format=1.2e-1]
                  S[table-format=-2]}\toprule
  sample & {weight in \unit{\kilo\gram}} &
           {resistivity in \unit{\ohm}} &
           {voltage in \unit{\micro\volt}} \\\midrule
  foo & 1.27  & 3e3    & -3  \\
  bar & 13.2  & 1.7e-3 & 0   \\
  baz & 0.138 & 5.66e9 & -74 \\\bottomrule
\end{tabular}
```

sample	weight in kg	resistivity in Ω	voltage in μV
foo	1.27	$3 \cdot 10^{3}$	−3
bar	13.2	$1.7 \cdot 10^{-3}$	0
baz	0.138	$5.66 \cdot 10^{9}$	−74

For the rare occasion that you need to connect cells horizontally, the `\multicolumn` macro is provided. It takes three arguments: `\multicolumn{`⟨*n*⟩`}{`⟨*colspec*⟩`}{`⟨*content*⟩`}`. ⟨*n*⟩ is the number of columns to connect, ⟨*colspec*⟩ is the specification for the connected column and ⟨*content*⟩ its content.

You never want to connect cells vertically.

13.13 Including Images

If you need to include any graphics in your document, load the graphicx package. There are really only two macros you absolutely have to know about.

`\includegraphics[`⟨*options*⟩`]{`⟨*filename*⟩`}` places the graphic ⟨*filename*⟩ in the document. The file format must be PDF, PNG or JPG, but you don't have to include the file format in ⟨*filename*⟩.

The most important options are `width` and `height`, which you can use to scale the image to the desired size. If you use only one of them, the aspect ratio of the graphic is conserved. A typical use case would be

```
\includegraphics[width=.8\textwidth]{cute_cat}
```

`\graphicspath{`⟨*paths*⟩`}` sets the paths LaTeX should search when looking for graphics. I usually have an `img/` folder and so I say (note the extra braces)

```
\graphicspath{{./img/}}
```

You can also specify several folders at once:

```
\graphicspath{{./img/setup/}{./img/diagrams/}}
```

If you have SVG drawings, the svg package helps you to easily include them into your document via *Inkscape*[20]. I *strongly* recommend setting the `inkscapearea` option to `page` instead of the default `drawing` when loading the package.

```
\usepackage[inkscapearea=page]{svg}
```

A bit of setup is necessary (see [svg-howto][21]), but after that, the package provides the commands `\includesvg` and `\svgpath` that can essentially be used just like the two commands above, but for SVG drawings. Maybe the biggest advantage of SVG drawings over other graphics formats is that LaTeX typesets any text in the drawing itself. This means that font and font size match the rest of the document and you can use all of the LaTeX symbols and other machinery in your SVG drawing as well. The `pretex` option allows you to insert some LaTeX code before text from the drawing, this is useful if you want to choose the `\small` font for the drawing.

```
\includesvg[width=.6\linewidth, pretex=\small]{cat_drawing}
```

[20] https://inkscape.org/

[21] https://tex.stackexchange.com/a/523685

Figure 1: The TEX lion [tex-lion].

13.14 Floating Content

There are two categories of material that you usually want to put into a floating environment (cf. section 9.6): figures and tables. LATEX provides the environments `figure` and `table` for this purpose. These environments are identical except for the counter that is used to number them and the name that is used for their captions. It is easy to define additional floating environments if the need arises, usually this is done by the packages you use to generate the content you want to allow to float, though. For example, both listings and minted define floating environments for displaying listings.

```
\begin{figure}
  \includesvg[height=6.5cm]{lion}
  \caption{The \TeX\ lion~\cite{web:tex-lion}.}
  \label{fig:tex-lion}
\end{figure}
```

Every floating environment takes an optional argument denoting its preferred placement, e.g. `\begin{figure}[`⟨*placement*⟩`]`. The ⟨*placement*⟩ may be a sequence of the characters `!` (lift placement restrictions), `h` (here), `t` (top of page), `b` (bottom of page) and `p` (on a float page) in any order (the order doesn't change anything), with the default being `tbp`. Inexperienced users often feel the need to say `[h]` at every float, or even use packages that define `H` to *really* place the "float" exactly where it is in the source file. This is a bad idea (cf. section 9.6). Only check if you need to influence the float placement at the very end, right before publishing.

A floating environment should always contain a `\caption` describing its content and a `\label` that you can use to refer to it in the text (cf. section 13.15). Remember that `\label` has to be placed *after* `\caption`. To avoid errors, it is best to just place it directly after `\caption`.

Table 1: Some natural constants.

Quantity	Symbol	value		Unit
Avogadro constant	N_A	6.022 140 76	$\cdot 10^{23}$	mol^{-1}
Planck constant	h	6.626 070 15	$\cdot 10^{-34}$	J s
gravitational constant	G	6.674 30(15)	$\cdot 10^{-11}$	$\mathrm{m}^3\,\mathrm{kg}^{-1}\,\mathrm{s}^{-2}$
electron mass	m_e	9.109 383 701 5(28)	$\cdot 10^{-31}$	kg
proton mass	m_p	1.672 621 923 69(51)	$\cdot 10^{-27}$	kg
elementary charge	e	1.602 176 634	$\cdot 10^{-19}$	C

Usually, figures and tables are typeset centered in the type area. To achieve this, you can use `\centering` as the first thing in the floating environment. You can make this the default by adding the following to your preamble (etoolbox has to be loaded)

```
% Center all floats by default.
\makeatletter
  \appto\@floatboxreset{\centering}
\makeatother
```

which is what I did and why I didn't have to use `\centering` in the previous example (and the next).

As noted above, the only difference between `figure` and `table` are their name and numbering. However, it is good practice to place table captions above the table but figure captions below the figure. This accounts for the different ways in which readers read figures opposed to tables.

```
\begin{table}
  \caption{Some natural constants.}
  \label{tab:natural-constants}
  \begin{tabular}{r >{\(}c<{\)} S[table-format=1.11(2)e-2]
                  >{\collectcell\unit}l<{\endcollectcell}}\toprule
    Quantity & \multicolumn{1}{c}{Symbol} &
      {value} & \multicolumn{1}{l}{Unit} \\\midrule
    Avogadro constant & N_{\mathrm{A}} & 6.02214076e23 & \per\mole \\
    Planck constant & h & 6.62607015e-34 & \joule\second \\
    gravitational constant & G & 6.67430(15)e-11 &
      \cubic\meter\per\kilo\gram\per\square\second \\
    electron mass & m_{\mathrm{e}} & 9.1093837015(28)e-31 & \kilo\gram \\
    proton mass & m_{\mathrm{p}} & 1.67262192369(51)e-27 & \kilo\gram \\
    elementary charge & e & 1.602176634e-19 & \coulomb \\\bottomrule
  \end{tabular}
\end{table}
```

When using KOMA-Script, use the document class option `captions=tableheading` to adjust the caption spacing for this convention.

13.15 Cross Referencing

Most numbered things in LaTeX are set up to be cross referenced. Whenever you use a command that creates some numbered item, like `\section` or `\caption`, this command prepares some information to be used for cross referencing.

`\label{`⟨*label name*⟩`}` can be issued *after* any of these commands to store this information under the ⟨*label name*⟩. Always choose a clear ⟨*label name*⟩ (cf. section 8.2)! I recommend only using letters, numerals and the characters : and - in the ⟨*label name*⟩.

`\ref{`⟨*label name*⟩`}` can be used anywhere in the document to print the number associated with the labeled object, e.g. "1" for section 1.

`\eqref{`⟨*label name*⟩`}` is like `\ref`, but it formats the number like an equation number. This should, of course, only be used when referencing equations.

`\pageref{`⟨*label name*⟩`}` can be used anywhere in the document to print the number of the page where the labeled object is located.

`\nameref{`⟨*label name*⟩`}` can be used anywhere in the document to print the name of the labeled object, if hyperref is loaded. This only makes sense for objects that actually have a name (like `\section`).

Remember that you have to compile twice for all labels to be correct (cf. section 4.8).

13.16 Citations and Bibliographies

biblatex is a very powerful package and we will only list the bare minimum of what you need to know to make it work here.

- You need to load biblatex in the preamble. Most of the package configuration can be done separately from loading the package, but there are some options that have to be issued in the optional argument to `\usepackage`. The only one you usually need is `style`, which determines the bibliography and citation style used throughout the document.

```
\usepackage[style=phys]{biblatex}
```

- You can place further configurations directly in the preamble of your document, but since these frequently get a bit lengthy and are often reused in many documents, it is best to write a configuration file. A file named `biblatex.cfg` located in the compilation directory will automatically be read by biblatex and can be used for this purpose.

- You need to point biblatex to your bibliography database file using the command `\addbibresource{`⟨*bibfile*⟩`}`. Unlike many other commands pointing to external files, you need to include the file extension in ⟨*bibfile*⟩.

  ```
  \addbibresource{myliterature.bib}
  ```

  The BIB file itself is most conveniently generated using a bibliography reference manager like JabRef[22].

- In your document, you can now cite any reference present in your BIB files using citation commands, usually `\cite[`⟨*page*⟩`]{`⟨*key*⟩`}` or `\footcite[`⟨*page*⟩`]{`⟨*key*⟩`}`.

- Finally, tell LaTeX to print the bibliography using `\printbibliography[`⟨*options*⟩`]`.

Remember that in order for all citations and bibliography entries to be up to date you need to call `pdflatex`, then `biber` and then `pdflatex` twice more. Of course, you do not have to do the full cycle every time you compile while writing your document.

13.17 Hyperlinks

As mentioned before, you can turn all of your internal references into links just by loading hyperref. That is, every entry in your table of contents or your index will be a clickable link, as well as any reference created using `\ref`, `\cite` or related commands. Of course, it also provides macros for creating links manually.

For internal links, these macros are relevant:

`\hyperref[`⟨*label*⟩`]{`⟨*text*⟩`}` prints ⟨*text*⟩ and make it a hyperlink to the given ⟨*label*⟩, which must be placed elsewhere in the document using `\label`.

`\hypertarget{`⟨*name*⟩`}{`⟨*text*⟩`}` prints ⟨*text*⟩ and defines it as a possible destination to link to using `\hyperlink`.

`\hyperlink{`⟨*name*⟩`}{`⟨*text*⟩`}` prints ⟨*text*⟩ and makes it a hyperlink to the target called ⟨*name*⟩ defined elsewhere in the document using `\hypertarget`.

If you want to link to external resources, like websites, you may need the following macros:

`\url{`⟨*URL*⟩`}` prints the ⟨*URL*⟩ and makes it a hyperlink. The formatting of the printed URL is governed by the value of `\UrlFont`. In my experience, the following two settings are both good choices:

```
\renewcommand*\UrlFont{\sffamily}
\renewcommand*\UrlFont{\ttfamily\small}
```

`\nolinkurl{`⟨*URL*⟩`}` prints the ⟨*URL*⟩, just like `\url`, but without making it a hyperlink.

[22] `https://www.jabref.org/`

`\href{`⟨*URL*⟩`}{`⟨*text*⟩`}` prints ⟨*text*⟩ and makes it a hyperlink to the destination given by ⟨*URL*⟩.

Notably, any ⟨*URL*⟩ has to start with `https://` (or `http://`) in order for the hyperlink to work. If that prefix is missing, the ⟨*URL*⟩ will be assumed to point to a local file, not a web resource.

Sometimes, the automatic creation of hyperlinks is undesirable because it may lead to nested links (which doesn't make any sense). Examples are section titles (because they appear as links in the table of contents) or arguments of `\href` or `\hyperlink`. For these cases, hyperref defines starred versions of the referencing commands that do not create a hyperlink. That is, `\ref*{`⟨*label*⟩`}` will just print the value of ⟨*label*⟩, with no linking.

13.18 PDF Properties

hyperref not only enables you to use the hyperlink capabilities that come with PDF. It also gives you access to other properties of that format. PDF bookmarks are set automatically and

`\hypersetup{`⟨*options*⟩`}` sets additional options parsed by hyperref, including many keys that set properties of the generated PDF file. Useful keys include `pdftitle`, `pdfauthor`, `pdfsubject`, `pdfdisplaydoctitle`, `pdflang`, `pdfduplex` and `pdfpagelayout`.

All of these PDF fields, like the author or a bookmark title, only accept plain text as their value. Because of this, you may receive errors or warnings when you try to pass content containing LaTeX features like exotic special symbols, superscript or color to a sectioning command or to an option like `pdftitle`. The following macro helps you deal with this.

`\texorpdfstring{`⟨*TEX string*⟩`}{`⟨*PDF string*⟩`}` expands to ⟨*PDF string*⟩ when in the context of a PDF field but to ⟨*TEX string*⟩ otherwise. For example, you may write

```
\section{\texorpdfstring{\(\frac{\lambda}{2}\)}{lambda/2} Resonators}
```

in order to print "$\frac{\lambda}{2}$ Resonators" in the document but use "lambda/2 Resonators" in the PDF bookmark.

14 How Macros Really Work

This section contains a very brief description on how macros really work, but not all of the aspects surrounding this are mentioned or explained in detail here. If you want to understand all the nitty gritty minutiae of TeX, reading *The TEXbook* [Knu84] is the way to go (it actually is a good read). If you already know your way around TeX and just need a reference, the freely available *TEX by Topic* [Eij92] is for you.

Note that I will sometimes use macros and TeX primitives in the code examples in this section that are not explained in this document. If you wish to know what exactly they do, please look them up in a reference or search for them online. The examples should however be understandable even without that exact knowledge.

Also note that even though I introduce writing plain TeX macros at some length here, you should really only do that if you know what you are doing. Otherwise, it's better to stick to the higher-level programming macros of LaTeX 2_ε and LaTeX3.

14.1 Category Codes

In TEX, every character in your source file has a *category code* associated with it. There are 16 of them, but I will only focus on category codes 11 (*letter*; the alphabet, i.e. `a` through `z` and `A` through `Z`), 12 (*other*; every character that has no special role, like digits and punctuation) and 13 (*active*; these behave like macros). (For most of the other category codes, there is usually only one character with that category code, which make up the list in section 4.1.) As explained in section 4.2, a control sequence is initiated by a backslash (\) and contains either only letters or a single non-letter. More precisely, it is initiated by any category 0 character (*escape character*) and contains either only category 12 characters or a single character of any other category.

This is used in LATEX to separate internal commands that should in principal not be used or altered by document authors (though that rule is routinely broken) from user macros. The convention is to have all internal control words contain the character `@`. Since it normally has category code 12 (other), it cannot be accidentally used by a document author. In package source code, the category code is changed to 11 (letter) and it can be used in macro names like any letter of the alphabet. You can perform this change yourself using the commands `\makeatletter` and `\makeatother`, which do exactly what they say. So, for example

```
\foo@bar \@gobble
```

usually means "the control word `\foo`, the characters `@`, `b`, `a` and `r`, a space character, the control symbol `\@` and then the characters `g`, `o`, `b`, `b`, `l` and `e` followed by a new line character (which acts like a space)". When switching category codes first, though,

```
\makeatletter
  \foo@bar \@gobble
\makeatother
```

the same sequence of characters in the source means "the control word `\foo@bar` and the control word `\@gobble`".

The term for a character or control sequence with the category code attached is *token*. When TEX first reads a character on the input stream (the source file), it attaches the currently valid category codes to it, which is called *tokenization*. This happens only once; if category codes change after some character has already been tokenized, that change will not affect that token. A consequence of this is that macros that work by changing category codes, like `\verb`, cannot function inside macro arguments, because the whole argument is tokenized before the macro can change the category codes. For example,

```
\emph{Do not use \verb|\verb| in macro arguments!}
```

will fail.

Now, what about active characters, i.e. those with category code 13? These characters are treated by TEX like a macro, i.e. they can be defined, take arguments and be expanded

just like a control sequence. Typical examples of active characters are ~, which is defined to insert a non-breaking space, and ", which babel uses to provide useful shorthands in many languages.

14.2 Macro Expansion

When TEX reads your source file, it sees a long string of text, the so-called input stream. TEX starts at the front, with the first token. Let's assume this is a macro. Then TEX looks up the definition of that macro, takes the macro out of the input stream and inserts its definition instead. If the macro takes any arguments, these are taken out of the input stream together with the macro and may be inserted in appropriate places in its definition. This replacing the macro by its definition is called *expansion*. TEX then looks at the token that is now the first one in the input stream, and so on.

Importantly, TEX only ever tokenizes as many characters from the input stream as necessary. So, at the start of the input stream, there is a number of tokens, i.e. characters that TEX has already "looked at", (these are said to be "in TEX's mouth") while the rest of the input stream remains untouched. New characters are only tokenized when they are at the front of the input stream or when TEX needs to scan ahead for something, e.g. macro arguments.

Before we discuss expansion some more, let us first look at how macros can be defined.

14.3 Writing Macros: The TEX Way

Here, I describe how to define new macros using plain TEX. Note that you should only do it this way if you really know what you are doing! Otherwise, it's better to stick to the higher level possibilities explained below.

`\def`⟨*macro name*⟩⟨*parameter pattern*⟩`{`⟨*replacement*⟩`}` is the basic command for macro definition. The ⟨*macro name*⟩ can be either a control sequence or an active character. The ⟨*parameter pattern*⟩ specifies how many arguments the macro expects in which form. You can specify up to 9 arguments as `#1` through `#9`. The ⟨*replacement*⟩ can be any sequence of tokens with balanced braces, where `#1` through `#9` will be replaced with the appropriate argument on expansion.

```
\def\foo#1#2{I saw #2, but first #1.}
\foo{hello}{world}
```

I saw world, but first hello.

`\gdef`⟨*macro name*⟩⟨*parameter pattern*⟩`{`⟨*replacement*⟩`}` does the same thing as `\def`, but globally, i.e. the definition is not constrained to the current group.

```
\def\foo{one}
\def\baz{two}
\begingroup
  \def\foo{three}
  \gdef\baz{four}
  \foo              \\
  \baz              \\[2ex]
\endgroup
\foo                \\
\baz
```

```
three
four

one
four
```

`\edef`⟨*macro name*⟩⟨*parameter pattern*⟩`{`⟨*replacement*⟩`}` does the same thing as `\def`, but it fully expands the ⟨*replacement*⟩ first. You will need this if you want to store the current value of some variable for later use.

```
\def\var{one}
\def\foo{\var}
\edef\baz{\var}
\foo                \\
\baz                \\[2ex]
\def\var{two}
\foo                \\
\baz
```

```
one
one

two
one
```

`\xdef`⟨*macro name*⟩⟨*parameter pattern*⟩`{`⟨*replacement*⟩`}` does the same thing as `\edef`, but globally, i.e. the definition is not constrained to the current group.

The ⟨*parameter pattern*⟩ warrants some further explanation. Apart from the argument placeholders `#1`, `#2`, etc., it may contain any other tokens (except for braces) that must then be used in the same pattern when the macro is used. If an argument is followed by such a token, it is called *delimited*. An undelimited argument will always only consist of the next available token or brace group. Spaces in front of undelimited arguments will be ignored when TEX scans for arguments. Both delimited and undelimited arguments will lose one layer of braces on expansion.

```
\def\foo#1#2{[#1::#2]}
\foo hello world              \\
\foo h{ello} world            \\
\foo{hello}world              \\
\foo{hello}{world}            \\
\foo {hello}   {world}        \\[2ex]
\def\foo#1#2r{[#1::#2]}
\foo hello world              \\
\foo{hello}world              \\[2ex]
\def\foo#1#2\foo{\baz #1 and \baz #2}
\def\baz#1{[#1]}
\foo hello world\foo          \\
\foo{hello}world\foo          \\
\foo{{hello}}{world}\foo      \\
\foo{{hello}}{{world}}\foo
```

[h::e]llo world
[h::ello] world
[hello::w]orld
[hello::world]
[hello::world]

[h::ello wo]ld
[hello::wo]ld

[h] and [e]llo world
[h]ello and [w]orld
[hello] and [w]orld
[hello] and [world]

If you want the expansion of a macro to contain a `#`, you need to escape it in the definition using another `#`, otherwise TEX will expect it to be part of an argument placeholder.

```
\def\foo#1{%
  \def\baz##1{Try #1 with ##1!}%
}
\foo{Nutella}
\baz{banana}            \\
\baz{peanut butter}     \\
\baz{fruit yoghurt}     \\[2ex]
\foo{peanut butter}
\baz{banana}            \\
\baz{honey}
```

Try Nutella with banana!
Try Nutella with peanut butter!
Try Nutella with fruit yoghurt!

Try peanut butter with banana!
Try peanut butter with honey!

There are two varieties of macros in TEX, called *short* and *long*. Short macros (which is the default) will not accept arguments containing a paragraph break (`\par` or equivalently a blank line), long macros do not have this restriction. This may be useful when you forget a closing brace or an argument delimiter, which causes TEX to scan ahead until it finds a matching one (or the document ends). If the macro is defined as short, TEX will stop scanning when it encounters a `\par` and complain about it. As this can otherwise lead to rather surprising behavior and errors, it is best not to make macros long if they should never receive entire paragraphs as arguments anyway. To make a macro long, just prefix `\long` to its definition, i.e. use `\long\def` instead of `\def`, etc.

Sometimes, you want to create a copy of a macro under a different name.

`\let`⟨*command*$_a$⟩⟨*command*$_b$⟩ makes ⟨*command*$_a$⟩ a copy of ⟨*command*$_b$⟩. As with `\def`, both control sequences and active characters work.

```
\def\foo#1{I saw #1.}
\def\baz{\foo}
\let\foocopy\foo
\foo{fire}              \\
\baz{fire}              \\
\foocopy{fire}          \\[2ex]
\def\foo#1{Give me #1 or else.}
\foo{fire}              \\
\baz{fire}              \\
\foocopy{fire}
```

I saw fire.
I saw fire.
I saw fire.

Give me fire or else.
Give me fire or else.
I saw fire.

There is no `\glet` command, instead you have to prepend `\global`, i.e. use `\global\let` instead of `\let`, in order to make the assignment globally.

So, what about optional arguments, you may ask. Well, there really are no optional arguments at all in TEX. LATEX only creates the illusion of optional argument using a construction like this one.

```
\makeatletter
  \def\foo{\@ifnextchar[{\foo@aux}{\foo@aux[that]}}
  \def\foo@aux[#1]#2{My name is #2, please call me #1.}
\makeatother
\foo{Paul}              \\
\foo[Sam]{Samantha}
```

My name is Paul, please call me that.
My name is Samantha, please call me Sam.

The macro `\@ifnextchar`⟨*char*⟩`{`⟨*true code*⟩`}{`⟨*false code*⟩`}` checks if the next character following it is ⟨*char*⟩. If so, it leaves ⟨*true code*⟩ in the input stream, otherwise ⟨*false code*⟩. Using this, `\foo` first checks if an optional argument is present and inserts the default if there is none. Then, the auxiliary macro `\foo@aux` does the actual thing you called `\foo` for.

So, what we call *optional arguments* in LATEX are checks for an opening bracket combined with an auxiliary macro scanning for an argument enclosed in brackets. What we call *mandatory arguments* are just normal, undelimited arguments. There are two important consequences of this

- You cannot use a closing bracket in an optional argument, as TEX will interpret it as the closing bracket of that argument (unlike with braces, TEX does not know the concept of matching brackets). If you do want to use an optional argument containing a closing bracket, you will have to enclose it in a brace group. For example, with the definition of `\foo` above, `\foo[[]]{brackets}` will fail while `\foo[{[]}]{brackets}` will succeed.

- The braces around mandatory arguments are in fact optional if the argument consist of a single token. It is considered good practice to use the braces anyway, because it clarifies which tokens are macro arguments and which aren't. Especially if you are not totally sure of when you need and when you needn't use braces, you should just always

use them. As you become more familiar with LaTeX, though, you may find it tiresome to always type the braces for common macros that usually apply to only one token, and move on to writing e.g. `\vec s` instead of `\vec{s}`. This is fine, in my opinion, as long as you stay consistent and use braces when it may be confusing otherwise (for example I'd say that `\frac 12` is fine but `\frac{a_n} 2` is not). The only arguments I'd recommend *not* enclosing in braces are those that must receive exactly one token, e.g. the first argument of `\newcommand`.

14.4 Expandability and Robustness

When I discussed macro expansion above, I omitted the question of what happens when the token at the front of the input stream is not a macro. TeX may "swallow" the token to put it into the box it is currently building for output on the page (e.g. if the token is the letter `a` or the character `!`). If the token is a TeX primitive, then it depends. Some primitives are expanded (remember this means that they only change the input stream), others need to be executed, e.g. for defining macros or setting a dimension register. All the tokens that affect other things than just the input stream when handled, i.e. everything except macros and the expandable primitives, are called *unexpandable*.

Usually, this distinction does not matter much, because TeX will just do whatever the token at the beginning of the input stream demands. However, there are some contexts in which TeX works in *expansion-only mode*, most notably when writing to files or inside an `\edef` or `\xdef`. In these contexts, TeX will happily leave any unexpandable tokens be and move on to the next one.

This can lead to problems when a macro relies on something being executed before something else is expanded. Here's an example.

```
\makeatletter
  \@tempcnta=0\relax
  \def\foo{%
    \advance\@tempcnta by 1\relax
    I like the number \the\@tempcnta.%
  }
\makeatother
\foo                    \\
\foo                    \\
\foo                    \\[2ex]
\edef\baz{\foo}
\baz                    \\
\baz                    \\
\foo
```

I like the number 1.
I like the number 2.
I like the number 3.

I like the number 3.
I like the number 3.
I like the number 6.

The macro `\foo` first increases the count register `\@tempcnta` by one using `\advance` and then prints a sentence using the current value of that count register. This works fine as long as `\foo` is used in the standard typesetting mode. After saying `\edef\baz{\foo}` you may have expected `\baz` to have the expansion `I like the number 4.`, however it actually prints "I

like the number 3." and furthermore has an unintended interaction with the behavior of `\foo`. Why does this happen?

The key to understanding this is knowing that the only expandable token in the definition of `\foo` is `\the`. It consumes the count register `\@tempcnta` and expands to its current value. All the other tokens are not touched by the `\edef` expansion. `\baz` thus always prints the value `\@tempcnta` had when it was defined while increasing the value of that register every time it is used.

Commands that break down when used in an expansion-only context are called *fragile*. While the example above is rather silly, it shows that using fragile commands in an expansion-only context can lead to unexpected results. More often than not, it will cause TEX to fail immediately, which is certainly better than producing unintended results, but the error messages may be hard to connect to the underlying issue of expandability.

The solution to this problem is to make the problematic macros *robust* by *protecting* them from being expanded in the wrong place. In LATEX 2_ε, this was done using the macro `\protect` in front of fragile commands and defining it to do nothing normally, but redefining it to prevent the following token from being expanded in the appropriate places. This is explained in a bit more detail in [fragile-robust][23], but you don't really need to know about all that, because ε-TEX has protection as a native concept. It introduced the `\protected` primitive, which you can prepend to a macro definition to make it robust. Macros defined this way will not be expanded in an expansion-only context.

```
\makeatletter
  \@tempcnta=0\relax
  \def\foo{%
    \advance\@tempcnta by 1\relax
    I like the number \the\@tempcnta.%
  }
  \protected\def\betterfoo{%
    \advance\@tempcnta by 1\relax
    I like the number \the\@tempcnta.%
  }
\makeatother
\edef\baz{\foo}
\edef\betterbaz{\betterfoo}
\ttfamily
\meaning\baz            \\
\meaning\betterbaz
```

```
macro:->\advance \@tempcnta by 1\relax I like the number 0.
macro:->\betterfoo
```

It is best to make every macro either protected or fully expandable. In LATEX 2_ε this is a bit of a hassle, because the syntax for defining robust and fragile commands is not consistent.

[23] https://tex.stackexchange.com/a/4747

So, if you are just writing a few small macros, I'd say don't worry about this rule too much. In LaTeX3 however, this separation is implemented consistently and should be followed.

14.5 Writing Macros: The LaTeX 2_ε Way

LaTeX 2_ε provides the following commands to define macros instead of with `\def`.

`\newcommand`⟨*macro name*⟩`[`⟨n_{args}⟩`][`⟨*default*⟩`]{`⟨*replacement*⟩`}` checks if ⟨*macro name*⟩ is already defined. If it is, it issues an error, otherwise it defines it. ⟨n_{args}⟩ is the number of arguments the new macro should take and can be a value from `1` to `9`. If ⟨*default*⟩ is given, the first argument (`#1`) is optional and has the value ⟨*default*⟩ when it is not given.

```
\newcommand\foo[2][that]{My name is #2, please call me #1.}
\foo{Paul}              \\
\foo[Sam]{Samantha}
```

My name is Paul, please call me that.
My name is Samantha, please call me Sam.

`\renewcommand`⟨*macro name*⟩`[`⟨n_{args}⟩`][`⟨*default*⟩`]{`⟨*replacement*⟩`}` does the same thing, but instead expects ⟨*macro name*⟩ to already be defined and issues an error otherwise.

`\providecommand`⟨*macro name*⟩`[`⟨n_{args}⟩`][`⟨*default*⟩`]{`⟨*replacement*⟩`}` checks if ⟨*macro name*⟩ is taken. If so, it does nothing without complaining, otherwise it defines the macro just like `\newcommand`.

An environment `foo` is made up of two macros, `\foo` and `\endfoo`. When the environment is started using `\begin{foo}`, some bookkeeping is done and finally `\foo` is expanded. Conversely, `\end{foo}` expands to `\endfoo` with some bookkeeping. Note that this means that you cannot define both an environment `foo` and a macro `\foo`. To define environments, the following macros are available.

`\newenvironment{`⟨*environment name*⟩`}[`⟨n_{args}⟩`][`⟨*default*⟩`]{`⟨*begin code*⟩`}{`⟨*end code*⟩`}` works just like `\newcommand`, but it defines an environment. The ⟨*begin code*⟩ will be executed at the start of the environment, the ⟨*end code*⟩ at its end. Note that you can only use the arguments in the ⟨*begin code*⟩.

`\renewenvironment{`⟨*environment name*⟩`}[`⟨n_{args}⟩`][`⟨*default*⟩`]{`⟨*begin code*⟩`}{`⟨*end code*⟩`}` does the same, but for environments that were already defined (like `\renewcommand`).

All of these commands create long macros by default. To define a short one, use the starred variant (`\newcommand*`, etc.) instead. While this syntax encourages the creation of long macros, I suggest only doing so when needed (as explained above).

14.6 Writing Macros: The LaTeX3 Way

In LaTeX3, the programming layer used for writing complex macros and packages is clearly separated from the user commands to be used by document authors. This separation as well as the naming conventions mentioned below are purely by convention; under the hood, everything still runs on TeX. They do however make it much easier to create complex commands and handle data in a more natural way as well as create user commands with a flexible interface of different kinds of arguments.

Because of the extensiveness of the LaTeX3 concepts, the description provided here will not come close to enabling you to use them. For that, you will have to read the proper documentation.

14.6.1 Creating User Commands Using xparse

The xparse (which is part of the LaTeX kernel, so you don't have to load it) package lets you easily define user commands with both undelimited and delimited, mandatory as well as optional arguments with or without default values. Starred variants (or in fact other optional characters in the argument syntax) as well as *embellishments* are easily specified as well. It also introduces *argument preprocessors* that can be used to regularize user input before it is passed to the macro's code. Please have a look at the xparse documentation for a detailed description of its features and syntax. Here, I only want to show a short example showing off some of its features.

`\NewDocumentCommand`⟨*function*⟩`{`⟨*arg spec*⟩`}{`⟨*code*⟩`}` checks if ⟨*function*⟩ is already defined and defines it otherwise. The power of xparse lies in the many possible argument types you can use in ⟨*arg spec*⟩.

Let us write a function `\newcmdsyntax` that tells us what a certain `\newcommand` call would do.

```
\NewDocumentCommand\newcmdsyntax{s m o o v}{%
  \texttt{%
    \string\newcommand
    \IfBooleanT{#1}{*}%
    \string #2%
    \IfValueT{#3}{[#3]\IfValueT{#4}{[\detokenize{#4}]}}%
    \{#5\}%
  }
  defines a new \IfBooleanTF{#1}{short}{long} macro \texttt{\string #2}
  \IfValueT{#3}{%
    with #3 arguments%
    \IfValueT{#4}{%
      , where the first one is optional with the default value
      `\texttt{#4}',%
    }
  }%
  that expands to `{\ttfamily\detokenize{#5}}'.%
}
```

```
\newcmdsyntax*\foo{Hello World!}                                    \\
\newcmdsyntax*\foo[1]{I saw #1.}                                    \\
\newcmdsyntax\foo[2][that]{My name is #2, please call me #1.}
```

`\newcommand*\foo{Hello World!}` defines a new short macro `\foo` that expands to 'Hello World!'.
`\newcommand*\foo[1]{I saw #1.}` defines a new short macro `\foo` with 1 arguments that expands to 'I saw #1.'.
`\newcommand\foo[2][that]{My name is #2, please call me #1.}` defines a new long macro `\foo` with 2 arguments, where the first one is optional with the default value 'that', that expands to 'My name is #2, please call me #1.'.

While the xparse functions are intended to eventually only be used to create user interfaces to the underlying programming layer (see below), for now you can also use it as a more powerful replacement to the LaTeX 2_ε macro creation commands.

14.6.2 Programming Using expl3

The LaTeX3 programming layer is contained in the expl3 package (which is also part of the kernel and does not have to be loaded) and explained in its documentation, the built-in functions and variables are described in [interface3]. All this is much to extensive to describe here in detail, so only the very broad strokes will be mentioned.

- expl3 programming takes place in a special category code regime, which you can enter with `\ExplSyntaxOn` and leave with `\ExplSyntaxOff` (much like `\makeatletter` and `\makeatother`). In this category code regime, _ and : are letters, spaces and newlines are ignored and ~ inputs a space. This means that spaces can liberally be used to organize your code without worrying about spurious spaces (cf. the previous code example). _ and : are used to give structure to macro names.
- A distinction between *functions* and *variables* is introduced. While this distinction is purely artificial (after all, this is still TeX, a macro expansion language), it is more natural to think in these terms. The idea is to separate macros that contain data (variables) from macros that contain code (functions).
- A clear naming scheme is used for both functions and variables.
  - \⟨*module*⟩_⟨*description*⟩:⟨*signature*⟩ for functions and
  - \⟨*scope*⟩_⟨*module*⟩_⟨*description*⟩_⟨*type*⟩ for variables.

  This naming scheme effectively allows functions to be overloaded and variables to be organized into global, local and constant ones.
- Several variable types together with functions to manipulate these variables as well as powerful expansion control capabilities are provided.

If you have never attempted to program anything in TEX, these points may not mean much to you. Trust me, these concepts vastly simplify the task. As a demonstration, here is a small program that I was able to pretty much just bang out using LATEX3, that you can probably understand immediately (well, after some getting used to the naming scheme). In plain TEX, which is the way programs were written for LATEX 2ε, this would taken much more effort, and the result would be much harder to comprehend.

```
\ExplSyntaxOn
  \cs_new:Nn \primer_gcd:nn
    {
      \int_compare:nNnTF { #1 } > { #2 }
        { \__primer_gcd:nn { #1 } { #2 } }
        { \__primer_gcd:nn { #2 } { #1 } }
    }
  \cs_new:Nn \__primer_gcd:nn
    {
      \int_compare:nNnTF { #2 } = 0
        { #1 }
        {
          \__primer_gcd:ee
            { #2 }
            { \int_mod:nn { #1 } { #2 } }
        }
    }
  \cs_generate_variant:Nn \__primer_gcd:nn { ee }
  \cs_new:Nn \primer_reduced_frac:nn
    {
      \primer_canceled_frac:nne
        { #1 } { #2 } { \primer_gcd:nn { #1 } { #2 } }
    }
  \cs_new:Nn \primer_canceled_frac:nnn
    {
      \primer_frac:ee
        { \int_div_truncate:nn { #1 } { #3 } }
        { \int_div_truncate:nn { #2 } { #3 } }
    }
  \cs_generate_variant:Nn \primer_canceled_frac:nnn { nne }
  \cs_new:Nn \primer_frac:nn
    {
      \int_compare:nNnTF { #2 } = 1
        { #1 }
        { \frac{ #1 }{ #2 } }
    }
  \cs_generate_variant:Nn \primer_frac:nn { ee }
  \NewDocumentCommand \reducedfrac { m m }
    { \primer_reduced_frac:nn { #1 } { #2 } }
\ExplSyntaxOff
```

```
\begin{align*}
  \frac{27}{6}        &= \reducedfrac{27}{6}        &
  \frac{31611}{57054} &= \reducedfrac{31611}{57054} &
  \frac{27699}{3957}  &= \reducedfrac{27699}{3957}
\end{align*}
```

$$\frac{27}{6} = \frac{9}{2} \qquad \frac{31611}{57054} = \frac{41}{74} \qquad \frac{27699}{3957} = 7$$

15 Getting Help

LaTeX can be confusing at times and the vastness of its features can leave users a bit lost. Luckily, there are many resources that can help you with this.

15.1 How Can I Produce This Symbol?

Traditionally, the way to look up symbols is by looking through *The Comprehensive LaTeX Symbol List* [comprehensive]. It lists symbols sorted by category and package. If you want to look up symbols of a certain category or compare different versions of the same symbol (provided by different packages), it is still the way to go.

If you know the shape of the symbol and just have to figure out how to produce it, though, *Detexify*[24] is much more useful. There, you can just draw your symbol and it will point you to the right packages and commands. For those who prefer drawing on their phone to drawing on their computer, there even is an app.

Some more resources for finding symbols, also for use with unicode-math, are mentioned in [finding-symbols][25].

15.2 How Does This Package Work?

As a general rule, RTFM! Most common LaTeX packages come with very fine documentation indeed. You can easily find it on *CTAN*[26], but it is also installed on your local machine with each package. The `texdoc` utility is the most convenient way of accessing this local documentation [texdoc]. Just type `texdoc` ⟨*package*⟩ in your system's command line to access the documentation of any installed ⟨*package*⟩.

While reading the documentation is the ultimate way to learn about a package, sometimes it is useful to just look at an example to get started. Many packages provide minimal (and sometimes advanced) examples in or alongside their documentation, but it can still be useful to search online or ask your fellow TeXers for one. Just remember that what other people show you is not always the best way to go about things.

[24] https://detexify.kirelabs.org/classify.html
[25] https://tex.stackexchange.com/a/21
[26] https://ctan.org/

15.3 I'm Getting Lots of Errors!

Sometimes, you just need to compile again for the errors to disappear. This can happen when you change something in your document that first needs to propagate through the auxiliary files (loading hyperref is a typical example). When in doubt, just try compiling twice without changing anything in between.

When actual errors appear, it's usually a whole bunch at once. This is because when TEX finds an error, it has no way of determining where the erroneous piece of code ends, and it just plows on. This in turn often causes a whole cascade of errors before TEX finally stops. Fixing the original error will usually prevent that cascade from starting and thus remove all error messages at once.

So, when you get errors, always look only at *the first one* and fix that first. Chances are, it's the only real one. Carefully read the error message (many editors try to parse error messages and sometimes miss part of it, so you may want to look at the error message in the log file; in TEXstudio you can do that by clicking the Log button in the tab of the same name). Many times, it tells you what's wrong. Sometimes, however, you get infamously cryptic error messages and can't really know what they mean without prior experience. In these cases, searching for the error message online usually leads you to the right solution.

15.4 I Still Need Help!

Well, that's okay. The best address for finding LATEX help online is the *TEX – LATEX Stack Exchange*[27], where you will find many questions already answered and some competent experts to answer new ones. Helpful links can also be found on the websites of *The LATEX Project*[28] and *DANTE e.V.*[29] (in German). If you are looking for a LATEX reference in book form, *The LATEX Companion* [Mit+04] may be for you.

Of course, asking your local TEX sage for help is always a good option as well. I am sure they'll be glad to help.

16 Further Reading

16.1 Weblinks

[ctan] *CTAN. Comprehensive TeX Archive Network*. URL: https://ctan.org/.

[dante] *DANTE e.V.. Deutschsprachige Anwendervereinigung TeX e.V.*. URL: https://dante.de/.

[data-prison] Zen Faulkes. *The data prison*. Aug. 23, 2012. URL: https://betterposters.blogspot.com/2012/08/the-data-prison.html.

[detexify] *Detexify*. URL: https://detexify.kirelabs.org/classify.html.

[27] https://tex.stackexchange.com/

[28] https://latex-project.org/

[29] https://dante.de/

[enit-hspacing] *enumitem: Understanding the usage of asterisk and exclamation mark in setting the different lengths. Answer by user schtandard.* URL: https://tex.stackexchange.com/a/490880.

[finding-symbols] *How to look up a symbol or identify a math symbol or character? Answer by user Rebekah.* URL: https://tex.stackexchange.com/a/21.

[fragile-robust] *What is the difference between Fragile and Robust commands? Answer by user mpg.* URL: https://tex.stackexchange.com/a/4747.

[git] *Git*. URL: https://git-scm.com/.

[inkscape] *Inkscape. Draw Freely*. URL: https://inkscape.org/.

[jabref] *JabRef*. URL: https://www.jabref.org/.

[latex-fonts] *The LaTeX Font Catalogue*. URL: https://tug.org/FontCatalogue/.

[latex-project] *The LaTeX Project*. URL: https://latex-project.org/.

[learnlatex] *Learn LaTeX online*. URL: https://www.learnlatex.org/.

[miketx-vs-texlive] *What are the advantages of TeX Live over MiKTeX?*. URL: https://tex.stackexchange.com/q/20036.

[miktex] *Getting MiKTeX*. URL: https://miktex.org/download.

[overleaf] *Overleaf. Online LaTeX Editor*. URL: https://www.overleaf.com/.

[overleaf-doc] *Documentation – Overleaf. Online LaTeX Editor*. URL: https://www.overleaf.com/learn.

[pygments] *Pygments. Python syntax highlighter*. URL: https://pygments.org/.

[python] *Python*. URL: https://python.org/.

[ragged-colspec] *How to create fixed width table columns with text raggedright/centered/raggedleft? Answer by user lockstep*. URL: https://tex.stackexchange.com/a/12712.

[svg-howto] *How to include SVG diagrams in LaTeX?. Answer by user schtandard.* URL: https://tex.stackexchange.com/a/523685.

[tex-lion] *What's with the Lion?* URL: https://ctan.org/lion/.

[tex-units] *What are the possible dimensions / sizes / units LaTeX understands? Answer by user Stefan Kottwitz.* URL: https://tex.stackexchange.com/a/41371.

[tex.sx] *TeX – LaTeX Stack Exchange*. URL: https://tex.stackexchange.com/.

[texlive] *Installing TeX Live over the Internet*. URL: https://www.tug.org/texlive/acquire-netinstall.html.

[texstudio] *TeXstudio. LaTeX made comfortable*. URL: https://texstudio.org/.

[upright-pi] *Upright Lowercase π*. URL: https://tex.stackexchange.com/q/54166.

16.2 References

The literature listed here consists mainly of package documentation and other manuals and references that are available on *CTAN* and via the `texdoc` utility. If the label of the bibliography entry is not a shortened form of author and year of the entry, you can find it using this name. For example, you can find *The Comprehensive LaTeX Symbol List* (which

has the label [comprehensive]) by typing `texdoc comprehensive` in the command line or by searching for "comprehensive" on `ctan.org`. (Of course, you will only find documentation of a package using `texdoc` if the package is installed on your machine.)

[acro] Clemens Niederberger. *acro. Typeset Acronyms and other Abbreviations*.

[amsmath] American Mathematical Society and LaTeX3 Project. *User's Guide for the amsmath Package*.

[array] Frank Mittelbach and David Carlisle. *A new implementation of LaTeX's `tabular` and `array` environment*.

[babel] Johannes L. Braams and Javier Bezos. *Babel. Localization and Internationalization*.

[beamer] Till Tantau, Joseph Wright, and Vedran Miletić. *The beamer class*.

[biber] Philip Kime and François Charette. *`biber`. A backend bibliography processor for biblatex*.

[biblatex] Philip Kime, Moritz Wemheuer, and Philipp Lehman. *The biblatex Package. Programmable Bibliographies and Citations*.

[bm] David Carlisle and Frank Mittelbach. *The bm package*.

[booktabs] Simon Fear. *Publication quality tables in LaTeX*.

[caption] Axel Sommerfeldt. *Customizing captions of floating environments*.

[comprehensive] Scott Pakin, ed. *The Comprehensive LaTeX Symbol List*.

[csquotes] Philipp Lehman and Joseph Wright. *The csquotes Package. Context Sensitive Quotation Facilities*.

[Eij92] Victor Eijkhout. *TeX by Topic. A TeXnician's Reference*. 1992.

[enumitem] Javier Bezos. *Customizing lists with the enumitem package*.

[etoolbox] Philipp Lehman and Joseph Wright. *The etoolbox Package. An ε-TeX Toolbox for Class and Package Authors*.

[expl3] The LaTeX3 Project. *The expl3 package and LaTeX3 programming*.

[fontspec] Will Robertson. *The fontspec package. Font selection for XeLaTeX and LuaLaTeX*.

[geometry] Hideo Umeki. *The geometry package*.

[glossaries] Nicola L. C. Talbot. *User manual for `glossaries.sty`*.

[glossaries-extra] Nicola L. C. Talbot. *`glossaries-extra.sty`. an extension to the glossaries package*.

[graphicx] David P. Carlisle and Sebastian P. Q. Rahtz. *The graphicx package*.

[hyperref] Sebastian Rahtz and Heiko Oberdiek. *Hypertext marks in LaTeX: a manual for hyperref*.

[interface3] The LaTeX3 Team. *The LaTeX3 Interfaces*.

[Knu84] Donald Ervin Knuth. *The TeXbook*. 1984.

[koma-script] Markus Kohm. *KOMA-Script. a versatile LaTeX 2ε bundle*.

[Lam94] Leslie Lamport. *LaTeX: A Document Preparation System*. User's Guide and Reference Manual. Addison-Wesley, 1994.

[listings] Carsten Heinz, Brooks Moses, and Jobst Hoffman. *The listings Package*.

[makeindex] Leslie Lamport. *MakeIndex: An Index Processor for LaTeX*.

[mathtools] Morten Høgholm and Lars Madsen. *The mathtools package*.

[memoir] Peter Wilson. *The Memoir Class. for Configurable Typesetting*.

[mhchem] Martin Hensel. *The mhchem Bundle. Documentation for the LaTeX Packages*.

[microtype] Robert Schlicht. *The microtype package. Subliminal refinements towards typographical perfection*.

[minted] Geoffrey M. Poore and Konrad Rudolph. *The minted package: Highlighted source code in LaTeX*.

[Mit+04] Frank Mittelbach et al. *The LaTeX Companion*. 2nd ed. Addison-Wesley Professional, 2004.

[nowidow] Raphaël Pinson. *The nowidow package*.

[pdflscape] Heiko Oberdiek. *The pdflscape package*.

[pgf] Till Tantau. *The TikZ and PGF Packages*.

[rotating] Robin Fairbairns, Sebastian Rahtz, and Leonor Barroca. *A package for rotated objects in LaTeX*.

[siunitx] Joseph Wright. *siunitx – A comprehensive (SI) units package*.

[stix2] STI Pub Companies, ed. *The stix2 package*.

[subcaption] Axel Sommerfeldt. *The subcaption package*.

[svg] Philip Ilten and Falk Hanisch. *The packages svg and svg-extract*.

[tcolorbox] Thomas F. Sturm. *The tcolorbox package*.

[texdoc] Manuel Pégourié-Gonnard and Takuto Asakura. *Texdoc. Find & view documentation in TeX Live*.

[unicode-math] Will Robertson et al. *Experimental Unicode mathematical typesetting: The unicode-math package*.

[verbatim] Rainer Schöpf, Bernd Raichle, and Chris Rowley. *A New Implementation of LaTeX's `verbatim` and `verbatim*` Environments*.

[xcolor] Dr. Uwe Kern. *Extending LaTeX's color facilities: the xcolor package*.

[xindy] Joachim Schrod. *`xindy`. create sorted and tagged index from raw index*.

[xparse] The LaTeX3 Project. *The xparse package. Document command parser*.

17 Index

`\(`, 14, 17
`\)`, 14, 17
`\,`, 27, 44
`\:`, 44
`\;`, 44
`\[`, 14
`\@`, 18
`\␣`, 18
`\]`, 14
`~`, 11, 18, 57, 65

acro, 34
`\addbibresource`, 54
`\addtocounter`, 38
`\advance`, 61
`align`, 41, 42
`aligneat`, 42
`aligned`, 43
`alignedat`, 43
`\Alph`, 38
`\alph`, 38
amsmath, 35, 41, 44

amssymb, 35
amsthm, 35
\ang, 40
\arabic, 38
\areaset, 23
array, 34, 35, 47
array, 34, 47
article, 30
autostyle, 44

babel, 32, 44, 57
beamer, 31
\begin, 12
\begingroup, 12
\bfseries, 17, 37
biber, 10, 16, 28, 29, 54
biblatex, 28, 29, 34, 53, 54
\blockquote, 45
bm, 35
book, 30, 31, 38
bookmarksnumbered, 34
booktabs, 24, 34
\bottomrule, 48

caption, 36
\caption, 51, 53
captions, 53
cases, 43
\cdot, 17
\ce, 40
center, 12, 25
\centering, 52
\chapter, 30, 38
\cite, 54
\cmidrule, 48
\columnwidth, 15
\cpi, 26
csquotes, 34, 44

\DeclareSIUnit, 40
\def, 57–59, 63
defaultlines, 32
\descfont, 46
description, 46
diff, 21
dimension, 14
displayquote, 45
DIV, 23
document, 12
\documentclass, 12, 13
dsfont, 35

\e, 26, 27
\edef, 58, 61, 62
\emph, 37
\end, 12
\endgroup, 12
\enquote, 44
enumerate, 46
enumitem, 34, 47
\eqref, 53
equation, 41
etoolbox, 33, 52
expl3, 65
\ExplSyntaxOff, 65
\ExplSyntaxOn, 65

figure, 24, 51, 52
figure, 38
figure-, 25
flalign, 42
float, 24, 51
\fnsymbol, 38
font, 26, 32–33, 37
fontenc, 32
fontspec, 33
\footcite, 54
\footnote, 46
footnote, 38, 46
\footnotemark, 46
\footnotetext, 46
\foreignquote, 44
\frenchspacing, 18

gather, 41
gathered, 43
\gdef, 57
geometry, 36
\global, 60
glossaries, 34
glossaries-extra, 34
glyphtounicode, 32
\graphicspath, 50
graphicx, 35, 50
grouping, 12

height, 50
hidelinks, 34
\href, 55
hypcap, 36
\hyperlink, 54, 55
hyperref, 34, 53–55, 68
\hyperref, 54
\hypersetup, 55
\hypertarget, 54

\i, 26, 27
\includegraphics, 50
\includesvg, 50
inkscapearea, 50
\intertext, 43
\item, 46
itemize, 46
\itshape, 37

KOMA-Script, 3, 23, 25, 30, 31, 36, 53

\label, 21, 51, 53, 54
\labelenumi, 46
\labelenumii, 46
\labelenumiii, 46
\labelenumiv, 46
\labelitemi, 46
\labelitemii, 46
\labelitemiii, 46
\labelitemiv, 46
latex, 7
\let, 59
\lfseries, 37
\linewidth, 15
listings, 36, 51
lmodern, 32, 33
locale, 34
\long, 59
lscape, 36
lualatex, 7, 10

magic comment, 10
\makeatletter, 56, 65
\makeatother, 56, 65
makeindex, 16
math mode, 14, 25–28, 41–44
math-style, 33
\mathrm, 26
mathtools, 35, 43, 44
\mdseries, 37
memoir, 31
mhchem, 34, 40

microtype, 32
\midrule, 48
minted, 36, 51
moderncv, 31
\multicolumn, 49
multline, 41, 42
multline*, 42

\nameref, 53
\newcommand, 61, 63, 64
\newcounter, 38
\NewDocumentCommand, 64
\newenvironment, 63
\nolinkurl, 54
\normalfont, 37
\notag, 43
nowidow, 23, 32
\num, 39

orphan, 23, 32
Overleaf, 11

\p@enumi, 46
\p@enumii, 46
\p@enumiii, 46
\p@enumiv, 46
\pagenumbering, 38
\pageref, 53
\par, 59
\paragraph, 30, 38
\part, 30, 38
pdfauthor, 55
pdfdisplaydoctitle, 55
pdfduplex, 55
pdflang, 55
pdflatex, 7, 54
pdflscape, 36
pdfpagelayout, 55
pdfsubject, 55
pdftitle, 55
PGF, 31, 35
pretex, 50
\printbibliography, 54
\protect, 62
\protected, 62
\providecommand, 63

\qquad, 44
\qty, 39
\quad, 15, 44
quotation, 44
quote, 44

\ref, 53–55
\renewcommand, 63
\renewenvironment, 63
report, 30, 31, 38
\rmfamily, 37
\Roman, 38
\roman, 38
rotating, 36

scrartcl, 31
scrbook, 31, 38
scrlayer-scrpage, 31, 36
scrlttr2, 31
scrreprt, 31, 38
\scshape, 12, 37
secnumdepth, 39
\section, 30, 38, 39, 53
section, 38
\setcounter, 38
\sffamily, 37
shell escape, 10, 36
\shortintertext, 43
sidewaysfigure, 36
sidewaystable, 36
\sisetup, 49
siunitx, 25, 34, 35, 39, 40, 49
\slshape, 37
\small, 50
split, 42
\stepcounter, 38
stix2, 33
style, 53
subcaption, 36
\subparagraph, 30, 38
\subsection, 30, 38
\subsubsection, 30, 38
\subsubsubsection, 30
svg, 35, 50
\svgpath, 50
\symbb, 35
\symbf, 35

table, 23, 47–49
table, 24, 51, 52
table-, 25
table-format, 49
tabular, 34, 41, 47, 48
\tag, 43
tcolorbox, 35
tex, 7
texdoc, 67, 69, 70
\texorpdfstring, 55
\textbf, 37
textcomp, 32
\textdel, 45
\textelp, 45
\textheight, 15
\textins, 45
\textit, 37
\textlf, 37
\textmd, 37
\textmu, 32
\textnormal, 37
\textquote, 45
\textrm, 37
\textsc, 37
\textsf, 37
\textsl, 37
\texttt, 37
\textup, 37
\textwidth, 15
\the, 62
\theenumi, 46
\theenumii, 46
\theenumiii, 46
\theenumiv, 46
\thefootnote, 46
\thesection, 38
TikZ, 35, 71
\times, 17
tocdepth, 39
\toprule, 48
\ttfamily, 37

unicode-math, 26, 33, 35, 67
\unit, 39
\upshape, 37
\url, 54
\UrlFont, 54
\usepackage, 53

\value, 38
\vec, 18
\vectimes, 17
\verb, 45, 56
verbatim, 35
verbatim, 35, 36, 45

verbatim*, 35, 46
widow, 23, 32
width, 50
xcolor, 34
\xdef, 58, 61
xindy, 16
xparse, 64, 65
\xtimes, 17