Getting started with LaTeX

Pierre S. Caboche

July 15, 2022

Updated: July 16, 2022

Abstract

This article condenses all the things I've learned about LaTeX while being stuck at home during the coronavirus pandemic. This guide will help you get started with LaTeX, so you can focus on the content of your document / thesis / book, etc.

This document is self-describing: if you see something in this paper and wonder *"how do we do that in LaTeX?"* then the answer is likely to be explained somewhere in there.

I am no LaTeX expert by any stretch of the imagination; but what I know, I've put in this document. I've also tried to structure the document in a way that is easy to follow (step by step, from an empty template, important features first), even with no prior experience of LaTeX.

Enjoy!

Part I
Preamble

1 Introduction

Every article tells a story, and this one is no different.

In this story, we go from being a complete LaTeX novice, to writing documents like this one. Starting from nothing, we learn how to install a LaTeX environment (for Windows or Linux), install TeXstudio, create a new document, render it, then add all the necessary components to make a good report: sections, Table of Contents, cross-references, bibliography, index…

So this is the story about a LaTeX document, talking about itself: many of the techniques used to produce this document are featured in this document, including some code samples.

In any case, sit back, relax, and enjoy the ride through our discovery of LaTeX…

2 Legal

Last revision: 16 July 2022

- Singular terms shall include the plural forms and vice versa.
- this Document is Copyright 2022 Pierre Caboche. All rights reserved. No unauthorised distribution allowed.
- the LaTeX Code which produced this Document is Copyright 2022 Pierre Caboche. All rights reserved.
- this Document also features Source Code (also known as Code Snippet), subject to different licenses.
  - if no license is mentionned, it is to be assumed that the Source Code is *proprietary*, and the property of its author and copyright holder.
  - if a license is mentionned (in a comment or through other means), you need to refer to the full description of the license.

    Below is an example of comment, indicating that the featured Source Code is subject to the BSD License, with mention of the copyright year and copyright holder:

    ```
    # This source code is under the BSD License
    # Copyright 2022 Pierre S. Caboche
    ```

- all of the Content (including Document, LaTeX Code, Source Code) IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ”AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
- the Copyright Holders reserve the right to amend this Legal Notice at any time, without prior notification. The latest version of the Legal Notice supersedes and replaces all prior versions of it.

BSD license

The code snippets written in LaTeX and featured in this document are governed by the BSD license.

Wherever the BSD license is mentionned, the following applies:

Table of Contents

Part II
LaTeX

In this part, we will explain what LaTeX is, when to use LaTeX (and why), and compare it with other software.

3 What is LaTeX?

LaTeX is a software system for document preparation.
Instead of using formatted text like in a WYSIWYG (”What You See Is What You Get”) word processors (such as LibreOffice Writer), the user writes LaTeX code (in plain text) which then can be processed to generate a document.

This approach has many advantages. For example, LaTeX allows to define programmable macros to help with recurring tasks (formatting and others); LaTeX provides thousands of packages that cover a variety of domains; and because LaTeX code is written in plain text, it can be managed just like any other source code (so you can use tools like GIT or SVN for collaboration and version control).

For all these reasons, LaTeX is widely used in academia, and in many fields like: mathematics, computer science, engineering, physics, chemistry, etc. LaTeX is also used for the preparation and publication of books.

Compared to WYSIWYG word processors, LaTeX may have a steeper learning curve but it can also do much more, and can greatly improve productivity in certain cases (e.g. technical articles, which require consistency in styling, the handling of multiple references, etc.).
This article will help you get started with LaTeX, covering a number of subjects needed to write a paper like this.

As a rule, this document is self-describing, i.e.: this document was written in LaTeX, and every technique used to produce this document is explained somewhere in this document.

3.1 Differences between TeX, LaTeX, and others

The original TeX was created in the late 1970s by Donald Knuth, who needed a new typesetting program.

At that time, Knuth was revising the second volume of his book “The Art of Computer Programming”*, got the galleys from his publisher, and was very disappointed in the result. The quality was so far below that of the first edition that he couldn’t stand it. Around the same time, he saw a new book that had been produced digitally, and thought he could produce a digital typesetting system. So he started to learn about typography, type design, and the rules for typesetting math* (TUG, 2021)[1], and thus started his work on TeX.

> The idea behind TeX was *“to allow anybody to produce high-quality books with minimal effort, and to provide a system that would give exactly the same results on all computers, at any point in time”* (Wikipedia, 2022b)

[1] I highly recommend you look at TUG (2021) if you want to learn more about the history of TeX

The commands in TeX were basic, but allowed the creation of macros to extend the list of commands.

In the early 1980s, Leslie Lamport created LaTeX, a typesetting program written in the TeX macro language. (Wikipedia, 2022a) As such, LaTeX provides a large set of macros for TeX to interpret, and TeX is in charge of formatting the output.

LaTeX packages are centralised in a repository called "The Comprehensive TeX Archive Network" (CTAN), *"the central place for all kinds of material around TeX"*

> Broadly speaking, you can think of LaTeX as: *"TeX, enhanced with a huge collection of macros: more than 6000 packages to date in CTAN (2022)."*

In 1989, Knuth declared that TeX was feature-complete, and only bug fixes would be made (Overleaf, 2022b). Since then, new typesetting programs based on TeX appeared: pdfTeX, XeTeX, LuaTeX…

When those typesetting programs are used in conjunction with the LaTeX macros, we talk of: pdfLaTeX, XeLaTeX, LuaLaTeX…

The advantage of XeTeX (and therefore XeLaTeX) is that:

- XeTeX supports UTF-8 by default
- XeTeX can make use of the fonts that are installed on your computer (not just the standard LaTeX fonts)

We'll be using XeLaTeX, so we can use the fonts that are installed on our system.

4 Why use LaTeX?

In this part, we show some examples where LaTeX is really beneficial. Conversely, we also discuss cases where better alternatives exist.

Then we discuss how some recent technological improvements made the LaTeX experience a lot more enjoyable than it was about a decade ago.

All of this should help you understand LaTeX better, so you can make an informed decision.

4.1 When is LaTeX a good choice?

LaTeX is a good choice in the following cases…

When you have large documents
: If your document contains a lot of parts or chapters, you can break them down in files which are easier to manage, organise (i.e. changing the order of such chapters), and collaborate on (section 8).

When you need to manage cross-references, bibliography, indexes…
: We'll explain how these work in sections "17 — Cross-references", "22 — Adding a bibliography", and "24 — Adding an index".

When the style needs to be consistent throughout
: If some elements need to always be formatted in a certain way, write a macro for that. If later you need to change the style, modify the macro and the change will be reflected throughout the whole document.

When you find recurring patterns in your document
Macros are great to avoid having to perform the same task multiple times: define a macro once, then call it whenever necessary (sometimes with different parameters).

In part “IX — Introduction to Macros” we explain how macros work, and give a few usage examples.

When you have a lot of mathematical formulas
LaTeX is really good for adding (and referencing) mathematical formulas! However, I will not cover this subject (I did not use any formulas in this article).

When you need to show a lot of source code
One thing I *did* use, however, is the ability to easily add code samples to a document (with some syntax highlighting for a variety of computer languages), whether it be small snippets of code, or entire source file.

More details in part “33 — Adding code samples”.

When you find a LaTeX package that does exactly what you need
This is a generalisation of the previous point, which uses a specialised package to print some source code from a variety of languages.

With thousands of different packages, it is possible you may find one that suits your need exactly, and that package may save you a lot of time down the line. When that happens, the time spent learning how to use LaTeX is only a small investment, which will be entirely recouped almost immediately.

In this article, I aim at reducing the time spent finding your way around LaTeX, and also introduce some of those useful packages, as well as a few techniques to gain in productivity.

4.2 When is LaTeX *not* a good choice?

When your document is very simple
For simple documents, LibreOffice is usually enough; no need to draw the “big gun” LaTeX.

When you need a lot of finely-tuned customisation
Think about posters, or other (usually short) documents for which you need to move visual elements around a lot.

For that, you may use something like LibreOffice Draw (or other software).

If you can’t stand the double-entendre jokes about using LaTeX
I will not elaborate…

4.3 There’s never been a better time to use LaTeX!

Technology evolved tremendously since the turn of the century. Some of those changes made LaTeX not only more usable, but actually quite pleasant to use[2].

In this section, we will look at some of the technological advancements, and see how these changes affected the overall LaTeX experience (now vs. when I was still at university).

[2] what was I saying earlier about double-entendres?

Screens got bigger

Throughout my studies, the computer screens I owned were usually 17 or 19-inch CRT monitors, with a resolution not exceeding 1280×1024. Out of curiosity, I resized a modern TeXstudio window to that resolution. Needless to say, the UI got really crammed. (lol)

Since then, computer screens made a lot of progress, and now we can comfortably edit our documents: with the LaTeX source code on one side of the screen, and the generated document on the other. No need to continuously switch between windows on a small screen.

Disk drives got larger (and faster)

LaTeX offers a large selection of packages for all kinds of uses. This also means that if you need many different packages, you will also need enough disk space to accommodate them (typically between 500 MB and 1 GB).

Nowadays, disk space is less of an issue, but this is something to keep in mind.

CPUs got considerably faster

A fast CPU is not really required for LaTeX, but that certainly helps. LaTeX is usually very demanding, except in certain cases (e.g. drawing complex plots that demand a lot of calculations; see section 39)

A modern CPU performs those tasks with relative ease (albeit using only one of the many cores usually available) but a processor from a few decades ago might have struggled a bit…

Fortunately, a large LaTeX document is usually divided in several files (section 8), and it is always possible to temporarily comment out parts of the documents if they take too long to render.

Internet became more widely available

This is probably the most important change for LaTeX. You will usually need a good internet for two things: 1. downloading the additional packages that you need, and 2. search "*how to perform task* T *in Latex?*" (usually shortened to *Latex perform T*")

Back when I was an exchange student, the place where I was living did not have internet, and doing some research required a bit of planning: I would need to make a list of the things I needed from the internet, walk to the University campus, download the files, put them on a Iomega® Zip™ disk (remember those?), and then come back to the student room I was renting.

Working on LaTeX without internet connection would have been quite a bit of a challenge. There is no denying that WYSIWYG editors offer a better offline experience (there are buttons everywhere, to perform all sorts of tasks), which might explain their early popularity.

LaTeX offers a completely different workflow. LaTeX relies on commands to perform tasks. This not only can potentially make you massively more productive (if you find the right command for your need), but offers a range of features that go well beyond what any WYSIWYG editor can offer (as we are not limited by what the UI has to offer; we can find some packages, or write our own macros).

The downside is, you need to find the information you need, which requires an internet connection. Even this document (which you can read offline, and should cover enough subjects to get you started with LaTeX) sometimes requires you to look some the online resources for certain packages.

The technological landscape changed considerably in the past few decades. Prior to 2011, LibreOffice was not even a thing and its predecessor, OpenOffice,

was what I used through most of university.

TeXstudio (my editor of choice for working on LaTeX documents) was first released in 2009. At the time, bigger and wider screens (with a resolution of around 1440×900 or 1920×1200) were becoming more common, and we'll see how TeXstudio takes advantage of that in part "III — TeXstudio".

Working on LaTeX files prior to 2009 (i.e. on a small screen, without TeXstudio, and in some cases without internet readily available either) would have been a very different experience than what it is today!

That's why I say there's never been a better time to use LaTeX: it's become a really good experience!

4.4 LaTeX compared to other software

In this section, we will compare LaTeX with other software, by use-case.

First, I would like to talk about Markdown…

Taking notes, making documentation

Just like LaTeX, Markdown is a markup language for creating formatted text, but Markdown is much simpler. The goal of Markdown is to generate documents that can be easily viewed on the web, while have a code that is easy to read. In some implementations, inline HTML tags can be supported, to add features that do not exist in Markdown.

Markdown was designed for creating web pages (for blogging, documentation, readme files, etc.) but is also great for taking notes (it's easy to create sections, subsections, lists, rudimentary tables, etc.)

Long technical documents

The issue is, you can only fit so much information into a web page before it becomes hard to read, then you need to split the information into several pages. This might be great for Search Engine Optimisation (i.e. a long article may be divided into several parts, each indexed individually), however this might not be ideal for longer documents like this one. That's where LaTeX shines!

WYSIWYG vs. WYSIWYM

Second is LibreOffice Writer, the WYSIWYG word processor that is free and available on a wide variety of platforms. It is good, very convenient, and you can export to PDF if you want your document to be read-only. LibreOffice Writer is what I'd use for small, personal files, where aesthetics don't matter.

LaTeX takes a different approach. Instead of being WYSIWYG (*"What You See Is What You Get"*), LaTeX is WYSIWYM (*"What You See Is What You* Mean*"*): you describe the concepts (e.g. "beginning of a section", "terms to emphasise"…) and let LaTeX deal with the presentation. This is not exactly a declarative approach, but it gets close to it.

Making posters

Then there is the question of documents that require a lot of customisation (e.g. posters, flyers, brochures, etc), which is the Achilles heel of LaTeX.

For that, you would need some desktop publishing software. There are some free ones (e.g. LibreOffice Draw, Scribus), and some commercial ones (e.g. Microsoft Publisher, Adobe InDesign).

Online collaboration

And finally, there is the elephant in the room: *Microsoft® Word®*. It allows several people to collaborate online on the same document in a WYSIWYG way. For many businesses, this feature alone could justify the use of the Microsoft Office over other solutions.

Collaboration in LaTeX (or MarkDown) is achieved through version control software, i.e. LaTeX code is managed just like any other source code.

Part III
TEXstudio

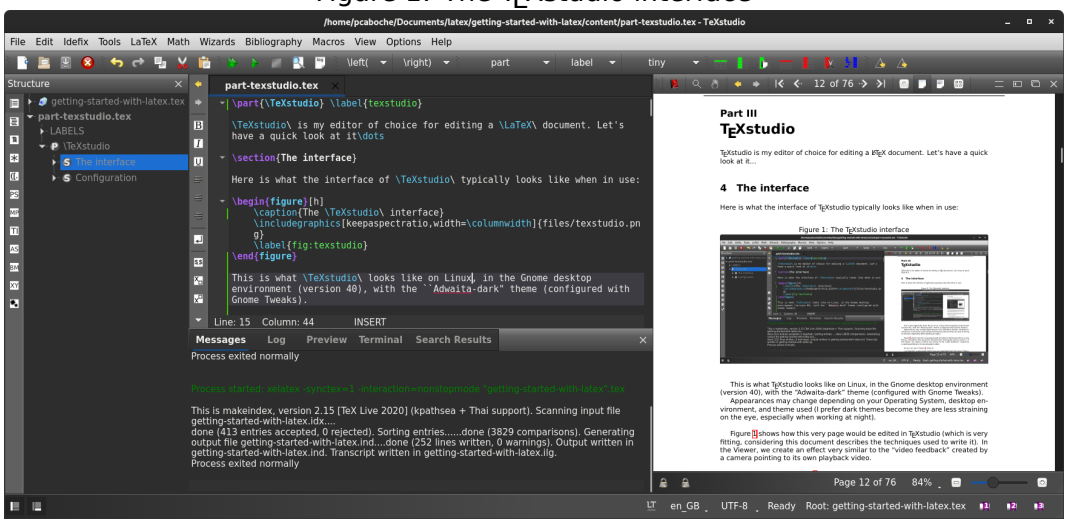
TEXstudio is my editor of choice for editing a LATEX document, an *“integrated writing environment for creating LATEX”* whose goal is to *“make writing LATEX as easy and comfortable as possible.”* (TeXstudio, 2022)

Let’s have a quick look at it…

5 The interface

Here is what the interface of TEXstudio typically looks like when in use:

Figure 1: The TEXstudio interface



This is what TEXstudio looks like on Linux, in the Gnome desktop environment (version 41), with the dark theme configured (I find dark themes less straining on the eyes, especially when writing late at night).

Appearances may change depending on your Operating System, desktop environment, and theme used.

Figure 1 shows how this very page would be edited in TEXstudio (which is very fitting, considering this document describes the techniques used to write it). In the Viewer, we create an effect very similar to the “video feedback” created by a camera pointing to its own playback video.

As you can see in figure 1, there is:

- on the left: a side panel (which you can show/hide from the menu “View » Show » Side Panel”) that will show you the structure of the document, give you access to some LATEX commands shortcuts, etc.

- in the center: the editor for your LATEX code.
  At the bottom of it, you will find a panel for the messages and logs

- on the right: the generated document, which will appear after you generate the document from your LaTeX files, by clicking the “Build & View” button (or by pressing F5)

Figure 2: The “Build & View” button (F5)

TeXstudio is best enjoyed on a fairly wide screen (just like many other code editors, where you often have several views side by side).

5.1 Build & View, Go to Source

When in the editor, clicking on the “Build & View” button (figure 2) will generate the document, and highlight the object corresponding to the code where your cursor was in the editor.

Similarly, in the editor, select some code, do a right-click, select the “Go to PDF” option, and TeXstudio will show you the location of the object in the generated document (based on the position of your cursor, not where you performed your right click).

Conversely, in the document viewer, do a right-click on any part of the document, select the “Go to Source” option, and TeXstudio will open the source file, and place the cursor on the code that generated that part of the document (i.e. where you performed the right click).

These options make it very easy to navigate between the LaTeX code and the generated document.

5.2 Toggle Comments

In TeXstudio, it is very easy to comment and uncomment parts of the source code.

- select the code you want to comment or uncomment
- toggle the comments with the “Ctrl + T” shortcut

We will see the importance of comments in the writing process in section “12 — Comments”. Notably, you can put as many comments as you want in a LaTeX document, so you don’t need to discard any draft idea: just comment them out. You’ll remove the comment only when you are sure it is not relevant anymore.

5.3 Undo, Redo

Like with many editors, “Undo” in TeXstudio is performed with the “Ctrl + Z” shortcut.
However, please keep in mind that the shortcut for “Redo” is “Ctrl + Shift + Z”.

5.4 Code completion

Like with many source code editors, TEXstudio provides some code completion that is tailored specifically for LATEX, for example:

- commands
  Any string starting with \ will show you a list of possible commands
- environments
  Any string starting with `\begin{` will show you a list of possible environments
- references
  Typing `\ref{`, `\nameref{`, or `\pageref{` will show a list of available label (see "17 — Cross-references")
- etc.

5.5 Code highlights

TEXstudio will highlight:

- the syntax
- words that may be spelled incorrectly (in red squiggly underline)
- words that are repeated within a short distance of each other (in yellow squiggly underline)
- duplicate labels (in orange)
- references that point to non-existing labels (in green squiggly underline)
- etc.

5.6 And so much more!

If you explore the interface of TEXstudio, you will see that there are many features to fit the needs of different LATEX users (to the point that it can be really overwhelming at the beginning).

I have decided to focus on a few features: the ones that are most universally used, and the ones that are the most useful, especially at the beginning.

At the core, the interface of TEXstudio is very straightforward: an editor, a document viewer, the ability to jump between the two (as described in section 5.1, a feature that really improves productivity!), code completion, different types of highlights (syntax, references, typos), etc. The workflow (in which we generate a document from LATEX code) is quite different from other text processors, but we get used to it fairly rapidly.

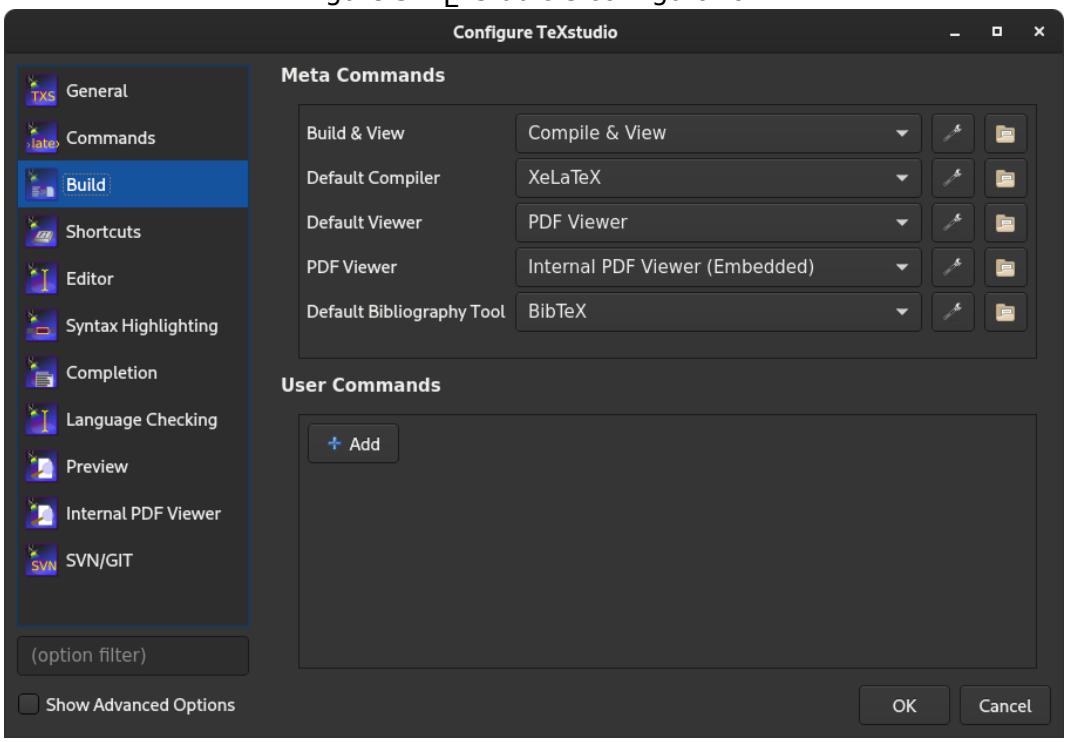
Now that we have a better understanding of what TEXstudio is, and why this is our editor of choice, our next step will be to install all of the necessary software for editing in LATEX…

6 Configuration

6.1 Configuring TeXstudio

Go to “Option » Configure TeXstudio...”. A window appears.
Select the “Build” tab

Figure 3: TeXstudio’s configuration



In this screen, we configure the following:

- we set “Default Compiler” to “XeLaTeX”
  This way, we will be able to use the fonts installed on our system
- we set “PDF Viewer” to “Internal PDF Viewer (Embedded)”
  This is the default
- we set “Default Bibliography Tool” to “BibTeX”
  Needed for section 22.2

6.2 Windowed mode

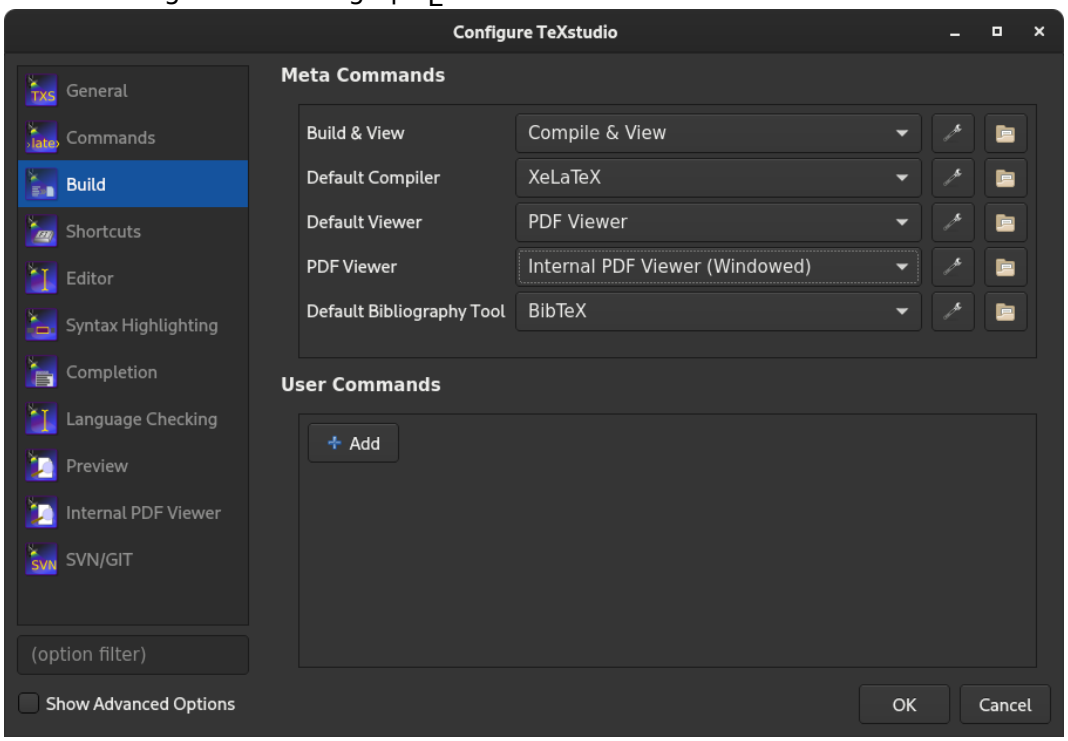
There are several reasons why you might prefer to open the PDF Viewer in Windowed mode rather than Embedded mode. For example, your screen might not be wide enough, or you may want to display the PDF Viewer in a secondary screen.

Here is how to configure the PDF Viewer in Windowed mode (this setting will apply even after you restart TeXstudio):

- Go to “Option » Configure TeXstudio...”. A window appears.
  Select the “Build” tab, then set “PDF Viewer” to “Internal PDF Viewer (Windowed)”, as shown in figure 4

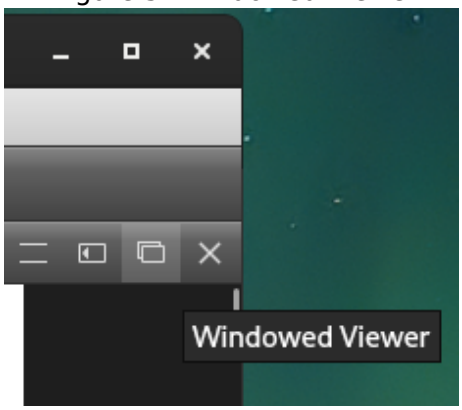
- If the PDF Viewer is already opened, then close it
- Press the “Build & View” button (figure 2), and the “PDF Viewer” will open in Windowed mode

Figure 4: Setting up TEXstudio’s Viewer in Windowed mode



If the PDF Viewer was already in Embedded mode and you want to put it Windowed mode, then press the button highlighted in figure 5 (note that this will not save the setting for after you restart TEXstudio)

Figure 5: Windowed Viewer



Part IV
Installation

Setting up a LaTeX environment

The official LaTeX Project website provides some information about LaTeX (including how to install LaTeX).

Link: `https://www.latex-project.org`

That being said, here is some information to get you started…

Windows

On Windows, I would recommend using MiKTeX, which includes (among other things):

- an *"integrated package manager"*, which will help you download the missing TeX packages, as you need them. This allows you to keep *"just enough TeX"* on your computer for your work (MiKTeX, 2022)
- *TeXworks*, a *"simple TeX front-end program (working environment)"* (TeXworks, 2022)

When it comes to TeX editors, I have a preference for TeXstudio, which we've introduced in *Part III — TeXstudio*

So on Windows, I would usually install the following:

- MiKTeX: `https://miktex.org`
- TeXstudio: `https://www.texstudio.org`

Linux

Under Linux, I tend to install `texlive` (as most Linux distribution provide it through their official repositories, as well as an editor for TeX (usually TeXstudio).

texlive
Required
TeX formatting system

texstudio
Optional, but highly recommended
A feature-rich editor for LaTeX documents

To install these packages in Fedora (requires *super user* privileges):

```
$ sudo dnf install texlive texstudio
```

Additional TeX packages need to be installed separately.

Such packages normally have a name that starts with "`texlive-`"
(e.g. `texlive-mdframed`)

Part V
Your first document

In this article, we will start with an empty document (created from a built-in template), and then slowly add features to it…

At first, we will focus on the features necessary to add *content* to our document (organising the document, add notes, table of contents, references, cross-references, bibliography, etc), in other words: everything that is important to present inter-connected ideas in a coherent manner.

We'll learn how to customise the style later (and if you've been given a template to follow, then you'll not even need to modify the style…)

7 Starting with a template

In TEXstudio, go to "File » New From Template…", select "Article" in the list of "Builtin" template, and finally click "OK".

What we get is the following file:

```
\documentclass[]{article}

%opening
\title{}
\author{}

\begin{document}

\maketitle

\begin{abstract}

\end{abstract}

\section{}

\end{document}
```

Now in TEXstudio, click on the "Build & View" button. This should generate the document without error…

> Congratulations! If the document displayed, then you have a working LATEX environment!
>
> With the installation done, we can now concentrate on learning LATEX itself. (note: you might still need to install additional packages when required)

Let's take a look at that file…

A LATEX file has two main parts:

- the *preamble* (everything before `\begin{document}`), which the template calls *"opening"*
- the document's *body* (`\begin{document}` and all of its content)

The *preamble* is where we invoke all the packages necessary to generate our documents, as well as configure those packages. Some LATEX commands can

only be called in the *preamble* (they will throw an error if called in the *body*); this is especially the case with configuration commands).

The *body* is where our document's content goes.

The `\documentclass` command specifies which document class (template) our document follows. We chose the class `article`, but there are others built-in in LaTeX (`letter`, `beamer`, `book`, `memoir`...).
The square brackets `[]` is where we specify options to our document class, for example `twoside` (section 28).

In the *preamble* of our template, you can notice the following commands: `\title` and `\author`. As the name implies, their goal is to declare the document's title and author (section 7.1).
Then in the *body*, the command `\maketitle` generate the document's title (including author, as well as date). It is possible to customise the date (section 7.2).

We'll examine the rest of the template in the following sections:

- `\begin{abstract}`: section 7.3
- `\section{}`: section 9

7.1 Set the title and author

The first thing we'll do to that template document, is specify the title and author, which is really straightforward:

```
\title{Article title}
\author{Your Name here}
```

Next we will modify the document's date...

7.2 Customise the date

By default, `\maketitle` will print out the date the document has been last generated (the result of calling `\today`), but we might want to change this for different reasons...

One reason could be that we want to specify a publication date, and not change afterwards.
Another reason would be to add a revision date, on top of the publication date that is set in stone. We are going to do the latter.

Here is how we do it, using the `\date` command:

```
\title{Getting started with \LaTeX}
\author{Pierre S. Caboche}

%% Custom format to indicate revision date
\date{
  July 15, 2022 \\                    % Date set manually
  \medskip
  \footnotesize \emph{Updated: \today}  % Will update automatically
}
```

7.3 Write an abstract

In our “Article” document template, you will notice the following block:

```
\begin{abstract}

\end{abstract}
```

As the name implies, this is where you need to write the abstract for your article.

An abstract is the summary of a research paper (whether published or not). The goals of an abstract is to:

- introduce the content of the paper
- present some of the ideas developed in the paper
- help remember the key points

An abstract is usually short (one paragraph, 6-7 sentences, 150-200 words).

If you don’t know what to put in an abstract, then a good tip I was given years ago was to write one or two sentences for each of the following subjects:

Background, **P**urpose, **M**ethodology, **R**esults, **C**onclusion

Usually: 1 sentence per subject, and an extra sentence each for Methodology and Conclusion.

So that’s what the “abstract” is… I know, this has nothing to do with LaTeX, but people might be wondering about the purpose of this block.

Note that I don’t always follow those rules for my articles. Specifically, my abstracts tend to be longer and describe the paper in more details.

8 Organising your files

8.1 File hierarchy

Unlike other text editors (e.g. LibreOffice), a LaTeX document can be made up of multiple files.

On top of that, LaTeX will create a number of other intermediate files (with extensions such as: `.aux`, `.bbl`, `.blg`, `.idx`, `.ilg`, `.ind`, `.lof`, `.log`, `.log`, `.out`, `.toc`, and even a `.gz` archive), as well as the final `.pdf` file.

All these files will be found in the same folder (in addition to *our* own files, which will define the content of our document). So we need to organise this folder.

Our goal is to have only one `.tex` file at the root of our folder hierarchy, and all other files grouped in different folders.

Here is the main hierarchy that I used for this document[3]:

[3] folders may contain sub-folders

`include/`
: folder contains our external dependencies, configurations, and custom macros. In particular:

  `packages.tex`
  : file containing all our dependencies and their configuration

  `macros.tex`
  : file containing our custom macros.

`content/`
: folder containing all the `.tex` files that form the content of our article, divided in parts / chapters / sections, etc.

  We also use this folder to store bibliography files (`.bib`), see Section 22

  I also strongly suggest that you organise your files in a way that makes it easy to see if they belong to a part / chapter / section or other. One way to achieve this is by adopting some naming conventions for your files.

  For example:

  ```
  part-(...).tex
  chapter-(...).tex
  section-(...).tex
  ```

`files/`
: folder containing other files necessary for generating the document (e.g. images, source code files…)

> You are free to use whichever file structure you want. However, in this article we will be using the structure described above, and will refer to each folder or file by their name in that hierarchy.

For your LaTeX documents, I suggest that you organise your files sooner rather than later:

Starting from a template, you may start adding the different parts, sections, sub-sections (section 9) to lay out the structure of your document. Then you may add a Table of Content (section 10) to better see that structure.
Finally, in the different sections you may add some notes in the form of comments (see “12 — Comments”) to know which ideas will go into which section.

At that point, you may still have most of your ideas in just one LaTeX file.

However, when you start writing the actual content of your document (not just adding sections names and comments), that’s when you need to make different files (in the `content/` folder).

8.2 Using `\input`

The `\input` command is used to load the content of a file, *verbatim*, into the current document, where the `\input` command has been called.

```
%% Importing our custom macros
\input{include/macros.tex}

  ...

%% Adding some content to our document
\input{content/part-style-indexes.tex}
```

This way, the content of our document can be broken down into smaller, more manageable files, but they will appear as one big file to the LaTeX compiler. As a result of this behaviour...

> The path specified as parameter to the `\input` command is always relative our main `.tex` document at the *base* of our folder hierarchy.

For example, if from a `.tex` document in our `content/` folder we include another `.tex` file, its path must be relative to the *base* of our folder hierarchy, where our main `.tex` document is located.

> This is also true for other LaTeX command that accept a file path as a parameter, like `\includegraphics` (page **??**), or `\lstinputlisting` (page 63).

The `\input` command is not to be confused with the `\usepackage` command... (see next section)

8.3 Calling `\usepackage`

The `\usepackage` command is used to load one or more LaTeX packages, sometimes with some parameters.

For example:

```
%%% In the preamble

%% Loading one package
\usepackage{xstring}

%% Loading several packages
\usepackage{titleps, xcolor}

%% Loading one package, with some option
\usepackage[super]{nth}
\usepackage[authoryear]{natbib}
```

Note that `\usepackage` can only be called in the preamble (not in the body of the document).

9 Parts, Sections, etc.

A document of class `article` (like this one), is divided into: parts, sections, subsections. The document class `book` also adds the “chapter” subdivision:

`\part`
`\chapter` (`book` only)
`\section`
`\subsection`

For example:

```
\part{Your first document}
   % ... introduce the part ...

\section{Starting with a template}
   % ... content of the section ...

\subsection{Set the title and author}
   % ... content of the subsection ...
```

The previous commands will define the different subdivisions that need to be *numbered*. Such subdivisions will not only be assigned a number, but they will also appear in the Table of Contents if you add one (see Section 10).

Also worth noting: the `\part` command does not reset the numbering of the chapters / sections it contains (i.e. you could have Sections 1, 2, 3 in part I; Sections 4, 5, 6, 7 in part II, etc.)

If you don’t want to number a subdivision, use the *starred* version of the previous commands, i.e.:

`\part*`
`\chapter*` (`book` only)
`\section*`
`\subsection*`

> Non-numbered subdivisions will not appear in the Table of Contents.

10 Adding a Table of Contents (TOC)

Adding a table of contents is very easy.

Just call `\tableofcontents` where you want your Table of Contents to be (only one per document) :

```
\tableofcontents
```

10.1 Customising the Table of Contents

It is also possible to customise the Table of Contents.

Here is what I used for this article:

```
%% Customise the title of the Table of Contents (default: Contents)
\renewcommand*\contentsname{Table of Contents}

%% Limit the depth of the TOC
\addtocontents{toc}{\setcounter{tocdepth}{3}}
\setcounter{tocdepth}{3}

%% Print the Table of Contents
\tableofcontents
```

> Note: if you modify those parameters, you might need to generate your document several times for the changes to take effect.

More customisation options can be found at (tom, 2021) :

https://texblog.org/2011/09/09/10-ways-to-customize-tocloflot/

11 Paragraphs, footnotes

In this section, we will learn how to make paragraphs, sub-paragraphs, etc.

The best way to explain how it works is to show the LaTeX code, and show you how it renders.

This is the LaTeX code:

```
This is the first line of our first paragraph (after all, we have to
    ↪ start somewhere\ldots)
This is on a new line (in the \LaTeX\ code), but it is still part of the
    ↪ same paragraph (in the generated document); so it is possible to
    ↪ add comments without disrupting the flow of the paragraph. For
    ↪ example\dots\
% This is a comment...
% Everything after the % until the end of the line is a comment, and will
    ↪  be ignored
this was added after the comment. As you can see, this is still part of
    ↪ the same paragraph; the comment didn't appear, and the flow of text
    ↪  was not disrupted.


By leaving (at least) one empty line between this and the previous block
    ↪ of text, we start a new sub-paragraph. Sub-paragraphs are very
    ↪ useful if you want to introduce a new topic in continuation with
    ↪ the previous point, but not a completely new idea that would
    ↪ warrant a new paragraph.

If you want to create a new paragraph, you need to terminate the current
    ↪ paragraph with a pair of backslashes ( \texttt{\textbackslash\
    ↪ textbackslash} ) and then leave at least one empty line between
    ↪ this and the next block of text.
\\


This is the start of a new paragraph. Now let's see what happens when we
    ↪ add a pair of backslashes ( \texttt{\textbackslash\textbackslash} )
    ↪  but do not leave an empty line with the next block of text\dots
\\
This is a new sub-paragraph; but unlike the others, this one is not
    ↪ indented (please, bear in mind: in some cases, that can make it
    ↪ difficult to see that this is a sub-paragraph at all).
\\

So far we've seen that a new paragraph can be used to introduce a new
    ↪ idea, while a new sub-paragraph can be used to introduce a new
    ↪ point pertaining to the same idea. The layout of the text (
    ↪ indentation, line breaks) will help us differentiate the different
    ↪ ideas/points.
\\

\medskip

And if you need a bigger break between paragraphs, you can use \texttt{\
    ↪ textbackslash medskip} or \texttt{\textbackslash bigskip} (medium
    ↪ or big skip, respectively). Use sparingly, though.

\begin{center}
  \textreferencemark\textreferencemark\
  Now, a word about footnotes\dots\
  \textreferencemark\textreferencemark
\end{center}

Sometimes when talking about a particular subject, you might feel the
    ↪ need to add some related information. Your first instinct would be
    ↪ to put such information in parentheses (but the problem with
    ↪ parentheses is that they appear in the middle of a text, and break
    ↪ the flow of the discussion. So you usually want to keep them both
    ↪ short and relevant. This is a counter-example, to illustrate what
    ↪ happens when the content in parentheses is too long: we tend to
    ↪ lose focus and forget what we were talking about). \\
```

```
One way to avoid being side-tracked by tangent ideas is to use \emph{
    ↪ footnotes}. As the name implies, footnotes are notes that appear at
    ↪  the foot of the page (or, in this case, at the end of a \texttt{
    ↪ mdframed} box) and referred to by number, like this\footnote{this
    ↪ is an example of a footnote. A footnote will not disrupt the flow
    ↪ of the discussion, so they can be used to add some comment or extra
    ↪  information that would be too long to put in parentheses
}.\\
```

...and this is the output. Please note that the code explains how paragraphs work. I needed some sample text, but instead of using a generic *Lorem Ipsum*, I figured that this section could be self-describing...

This is the first line of our first paragraph (after all, we have to start somewhere...) This is on a new line (in the LaTeX code), but it is still part of the same paragraph (in the generated document); so it is possible to add comments without disrupting the flow of the paragraph. For example... this was added after the comment. As you can see, this is still part of the same paragraph; the comment didn't appear, and the flow of text was not disrupted.

By leaving (at least) one empty line between this and the previous block of text, we start a new sub-paragraph. Sub-paragraphs are very useful if you want to introduce a new topic in continuation with the previous point, but not a completely new idea that would warrant a new paragraph.

If you want to create a new paragraph, you need to terminate the current paragraph with a pair of backslashes (\\ ) and then leave at least one empty line between this and the next block of text.

This is the start of a new paragraph. Now let's see what happens when we add a pair of backslashes (\\ ) but do not leave an empty line with the next block of text...
This is a new sub-paragraph; but unlike the others, this one is not indented (please, bear in mind: in some cases, that can make it difficult to see that this is a sub-paragraph at all).

So far we've seen that a new paragraph can be used to introduce a new idea, while a new sub-paragraph can be used to introduce a new point pertaining to the same idea. The layout of the text (indentation, line breaks) will help us differentiate the different ideas/points.

And if you need a bigger break between paragraphs, you can use `\medskip` or `\bigskip` (medium or big skip, respectively). Use sparingly, though.

※※ Now, a word about footnotes... ※※

Sometimes when talking about a particular subject, you might feel the need to add some related information. Your first instinct would be to put such information in parentheses (but the problem with parentheses is that they appear in the middle of a text, and break the flow of the discussion. So you usually want to keep them both short and relevant. This is a counter-example, to illustrate what happens when the content in parentheses is too long: we tend to lose focus and forget what we were talking about).

One way to avoid being side-tracked by tangent ideas is to use *footnotes*. As the name implies, footnotes are notes that appear at the foot of the page (or, in this case, at the end of a `mdframed` box) and referred to by number,

like this[a].

[a] this is an example of a footnote. A footnote will not disrupt the flow of the discussion, so they can be used to add some comment or extra information that would be too long to put in parentheses

11.1 Footnotes

In the previous code sample, we saw an example of a `footnote`.

In LaTeX, it's very easy to add a footnote with the `\footnote` command:

```
This is some text\footnote{and this is the footnote}.
```

Footnotes will appear at the foot of the page (see: example[4]), or at the bottom of components like a `mdframed` box.

As mentioned earlier, footnotes can be useful when you want to add some information but it would be too long to put in parentheses without disturbing the flow of ideas.

So the following can be used to add additional information:

- parentheses: used for short explanations, closely related to the subject
- footnotes: used for additional information that is either too long to be in parentheses, or not closely related to the subject
- margin notes (section 11.2): to bring the reader's attention on a particular subject

[4] this is an example of a footnote. As you can see, it is able to hold a lot more information than a margin note

11.2 Margin notes and paragraphs

LaTeX also allows to put notes in the margins, thanks to *margin paragraph* and *margin notes*.

This is a margin paragraph

To add a margin paragraph, use the `\marginpar` command:

```
\marginpar{This is a margin paragraph} To
add a margin paragraph, use the \lstinline[language=tex]|\marginpar|
command:
\lstinputlisting[language=tex]{content/sample-marginpar.tex}
```

To add a *margin note*, you will need `marginnote` package, which provides the `marginnote` command:

```
%%% In the preamble
\usepackage{marginnote}
```

This is a margin note

A margin note can be added by using the `\marginnote` command:

```
\marginnote{This is a margin note} A
margin note can be added by using the \lstinline[language=tex]|\
    ↪ marginnote|
command:
\lstinputlisting[language=tex]{content/sample-marginnote.tex}
```

Notice how in both cases, we put the `marginpar` or `marginnote` command immediately after the first word in the paragraph. This is to make sure the margin paragraph/note will be aligned vertically with the first word of the paragraph.

Also keep in mind that in a two-sided document (section 28), a margin paragraph will be on the *outer* margin (i.e. left margin for even-numbered pages, right margin for odd-numbered pages). If the document is one-sided, margin paragraphs will appear in the right margin.

marginpar vs. footnote

A margin paragraph (or note) stands out from the rest of the text. They can be used to bring the reader's attention on a particular paragraph. However, *it is better to keep them short*.

In contrast, a footnote (*section 11.1*) can be much longer, and is used to provide additional information without breaking the flow of the discussion.

11.3 Putting text in a frame

Previously, we've seen how margin notes and margin paragraphs could be used to bring attention to certain parts of your document.

Now we will learn how to make some ideas stand out, by putting them in a big box, thanks to the `mdframed` package:

```
%%% In the preamble
\usepackage{mdframed}
```

The `mdframed` package defines, amongst other things, the `mdframed` environment. Here is a very simple example :

```
\begin{mdframed}
Help! I've been framed!
\end{mdframed}
```

You can notice how an environment is delimited by `\begin{`*`envname`*`}` and `\end{`*`envname`*`}`.

And here is the result of the code above:

Help! I've been framed!

This is a very simple frame, but we can modify its appearance with the `\mdfsetup` command.

One thing to know about the `\mdfsetup` command, is that is modifies the properties of *every* `mdframed` that comes after it. This is a problem if you want to modify the attributes of only *one* frame.

To get around this issue, we will create an *environment*.
We will name it "`vegas`". Why? because *"what happens in* `vegas` *stays in* `vegas`*"* (and what happens in an environment stays in that environment").

So here is our "`vegas`" environment:

```
\newenvironment{vegas}[1][!!! Welcome to Vegas !!!]{
  \mdfsetup{
    frametitle={
      \tikz
      \node[rectangle,fill=yellow!60,draw=red, ultra thick]
      {#1};
    },
    frametitleaboveskip=-\ht\strutbox,
    frametitlealignment=\centering,
    backgroundcolor=yellow!30,
    linewidth=4pt,
    linecolor=red!70,
    roundcorner=8pt,
    shadow=true,
    shadowcolor=orange!40,
  }
  \begin{mdframed}}
  {\end{mdframed}}
```

For the purpose of this example, we declared this environment with one optional parameter: the title of the frame, which defaults to "!!! Welcome to Vegas !!!".

Now it's time to use this environment:

```
\begin{vegas}
What happens in this environment stays in this environment\ldots
\end{vegas}
```

And here is what it looks like:

!!! Welcome to Vegas !!!

What happens in this environment stays in this environment…

Wow, so many colours!

Now, let's create another frame:

```
\begin{mdframed}[
  frametitle={The title of the frame},
  frametitleaboveskip=-\ht\strutbox,
]
Help! I've been framed!
\end{mdframed}
```

And here is the result:

The title of the frame

Help! I've been framed!

As you can see, our very flamboyant "vegas" environment did not have any influence over to the new frame. The \mdfsetup command only affected the environment where it was used (i.e. "vegas"), and had no effect outside of it.

Again, we can define a new environment, call \mdfsetup inside of it, and the effect of \mdfsetup will not be seen outside of the environment:

```
\newenvironment{blueframe}[1]{
\mdfsetup{
  frametitle={
    \tikz
    \node[rectangle,fill=blue!20]
    {#1};
  },
  linecolor=blue!20,
  linewidth=2pt
}
\begin{mdframed}}
{\end{mdframed}}
```

```
\begin{blueframe}{Put title here}
Help! I've been framed!
\end{blueframe}
```

Result:

Put title here

Help! I've been framed!

In our two previous environments ("vegas" and "blueframe"), we passed the frame title as a parameter (whether optional or mandatory). However, if we

don't need to add a title, there is a simpler way to define frame environment, thanks to the `\newmdenv` command.

Below is the definition for "`note`", an environment which I used to add some important notes throughout this document, in the form of a frame with a slightly yellow background:

```
\newmdenv[
  middlelinewidth=2pt,
  backgroundcolor=yellow!10
]{note}
```

Here is how to call it:

```
\begin{note}
This is an example of note.
\end{note}
```

Here is the result:

> This is an example of note.

As you can see, all the options that we would normally pass to the `\mdfsetup` command are passed in the optional parameter (in square brackets `[]`) of `\newmdenv`. The name of the environment is passed as the mandatory argument (in curly brackets `{}`) of `\newmdenv`.

Another way to modify the appearance of a `mdframed` is to define a new style with the `\mdfdefinestyle` command, then use that style by specifying the `style` attribute in the `mdframed` options:

```
\mdfdefinestyle{pinky}{
  linecolor=pink,
  linewidth=2pt,
  backgroundcolor=pink!30
}

\begin{mdframed}[style=pinky]
They're Pinky and the Brain, \\
Yes, Pinky and the Brain, \\
One is a genius, the other's insane\ldots
\end{mdframed}
```

And the result:

> They're Pinky and the Brain,
> Yes, Pinky and the Brain,
> One is a genius, the other's insane…

This concludes our presentation of `mdframed`.

If you want to know more about this package (with many other examples), please refer to its official documentation:

`https://www.ctan.org/pkg/mdframed`

11.4 Other types of boxes

There exists other kinds of boxes, to draw boxes around text:

- `\parbox`
- `\mbox`
- `\makebox`
- `\fbox`
- `\framebox`
- `fancybox` (package)

For more information, please see:

`https://en.m.wikibooks.org/wiki/LaTeX/Boxes`

12 Comments

In LaTeX, a comment starts with: %

Everything after the "%" till the end of the line will be ignored. If you want to actually print a "%" in LaTeX, you'll need to escape it with a backslash: \%

Unlike in other computer languages, there are no multiline comments in LaTeX, only single-line comments.

Fortunately, an editor like TeXStudio can make the process of adding/removing comments a lot easier, thanks to the `Ctrl+T` shortcut (for *"Toggle Comment"*).

To toggle comments in TeXStudio:

- Select the lines you want to add comments to (or remove comments from)
- Press `Ctrl+T`

  This will either add or remove the comments from those lines

Comments are extremely useful in LaTeX! In fact…

When you think about it, comments are one of the most powerful features in LaTeX.

Indeed, comments allow you to keep multiple drafts of the same paragraphs, which you can then enable/disable with a single `Ctrl+T` in TeXstudio. This way, you never have to throw away any idea! (just comment them out, so they don't show up in the document).

Comments have a number of applications:

Describing
: The most obvious use of a comment is to describe what some LaTeX code is doing (e.g. describe the packages that you are using and what you are using them for)

Taking notes
: For example, if you already have your document's plan laid out (i.e. you've already defined the structure in terms of: parts, sections, sub-sections…), you can leave comments to list the main ideas to put in those sections later (as some kind of *TODO* list)

  Furthermore, if several people are working on a document you can leave notes to other contributors in the form of comments (and if you work alone, you can always leave notes to yourself)

Drafting ideas
: This is probably the most important use of comments (this represents about 99% of the cases when I use comments in LaTeX)

  Sometimes you're writing things but are not completely satisfied with the wording. In LaTeX, it's very easy to put a sentence in comments, rewrite it, and repeat the process until you're satisfied (while not having to discard what you wrote, and not having those earlier versions appear in your document).

In many of those cases, you don't want your comments to be removed by accident (i.e. any case other than *Drafting ideas*). To avoid that, simple use more than one ' `%` ' for your comment.

This way, one `Ctrl+T` will not be enough to remove the comment…

```
%%% Packages and configuration
```

13 Collaboration, version management

The main focus of LaTeX is document generation. As such, it does not provide dedicated collaboration and version management tools.

However, LaTeX is a markup language. This means that, collaboration and management of LaTeX files can be handled with the same tools as any other source files; so you can use your favourite version management tools with LaTeX (e.g. GIT, SVN…)

In section "12 — Comments", we've seen how LaTeX comments can be used for note-taking, drafting entire paragraphs, etc. A version-control management system can help you keep a history of such items.

Part VI
Managing breaks

In part V, we talked about how to how to write paragraphs, how to organise your documents in sections, etc.

Sometimes, however, a paragraph or section break does not land in a place that is convenient for us, as it disrupts the flow of the article, For example, a section may start at the bottom of a page, a figure may end up on a different page from the text that references it (I always find this confusing to read when that happens), etc.

To avoid that, you might want to make some manual adjustments to the document layout. This is the subject of this part…

14 Non-breaking spaces

In LaTeX, you can create a non-breaking space by using a tilda: ~

Non-breaking spaces are used to keep certain terms together, without automatic line breaks being inserted in the middle.

This is useful, for example, to keep numbers and their units together:

```
The maximum theoretical speed of SATA-3 is 6~Gb/s, or 750~MB/s.
```

Whenever possible, LaTeX will keep those terms together.

In the previous example, the “6” should not be separated from its “Gb/s” unit, and the “750” should not be separated from the “MB/s”.

15 Phantom text

Since we are on the subject of adding breaks and spaces, here are the commands to add some empty space based on the width/height of a given text:

`\phantom{text}`
: Creates some space, of the same width and height as the `text` passed as parameter (a combination of `\hphantom` and `\vphantom`)

`\hphantom{text}`
: Creates some horizontal space, of the same width as the `text` passed as parameter

`\vphantom{text}`
: Creates some vertical space, of the same height as the `text` passed as parameter

Below is an example of how to put an \underline under some \phantom text:

```
I really think that \underline{\phantom{\LaTeX}} is great!
```

And here is the result:

I really think that ____ is great!

16 Skips and breaks

16.1 Adding vertical spaces

Sometimes you might need to add some extra vertical space between paragraphs (or other elements). These are called “skip”.

Skips come in different sizes:

\smallskip	a small skip
\medskip	a medium skip
\bigskip	a big skip

16.2 Page breaks

You can start a new page with the \newpage command.

You can also start a new page with \pagebreak, in which case the content of the old page will be spread out vertically.

Having several \newpage (or \pagebreak) will not create empty pages. To insert empty page(s), see section 16.3…

16.3 Inserting a blank page

While writing about \newpage and \pagebreak, I wondered *“how to insert a blank page in a LaTeX document?”*

After a minute of searching, I found the following solution, which I then wrapped in a custom macro with an optional parameter (section 26.4):

```
\newcommand{\insertBlankPage}[1][This page intentionally left blank.]{
\newpage
\thispagestyle{empty} % remove header and footer for current page
\vspace*{\fill}
\begin{center}
#1
\end{center}
\vspace*{\fill}
\newpage
}
```

The macro will insert a blank page, with a custom message (passed as an optional parameter) written in the center of the page. If the message is not specified, it will default to “This page intentionally left blank.”

This macro uses the `\thispagestyle` command, which sets the style for the current page only. By setting it to `empty`, we remove the header and footer temporarily. Therefore only the message will appear on the center of the page.

Here are some examples of how to call our custom macro:

```
% To display a blank page with a custom message
\insertBlankPage[Some text]

% To display a blank page
% with the default "This page intentionally blank."
\insertBlankPage

% In this case, the message will actually be blank
\insertBlankPage[]
```

We will be calling the following:

```
\insertBlankPage
```

See next page for the result...

This page intentionally left blank.

Part VII
Cross-references, bibliography

17 Cross-references

Adding a cross-reference in LaTeX is easy!

First, you need to put a label next to the object you want to reference (e.g. section, table, figure…) This is done with the `\label` command:

```
\section{Cross-references} \label{cross-references}
```

Of course, labels must be *unique*!

Don't worry though, TeXstudio will highlight any label that has been defined more than once in a document.

Labels are *case-sensitive*, but they can be *any* string of characters (including numbers, white spaces, punctuation, multibyte unicode characters…) This is different from identifiers used in bibliography (section 22.1)
For the sake of consistency, it can be a good idea to follow the same naming convention for all types of `id`, including labels (i.e. only alphanumeric characters, dashes, underscores).

Now that we've defined a label, it's easy to reference it.

The following shows some examples of how to use commands like `\ref`, `\nameref`, `\pageref` :

```
Section \ref{cross-references}
is called \emph{``\nameref{cross-references}"}
and can be found on page \pageref{cross-references}.
```

…and here is the result:

> Section 17 is called *"Cross-references"* and can be found on page 40.

17.1 Things to avoid with references

In your document, avoid expressions such as *"in the next section'*, *"in the previous section'*… Instead, refer the that section (or other object) by reference.

As explained in "8 — Organising your files", we put the content of our document in separate files, which we then include using the `\input` command.

By doing so, it becomes incredibly easy in LaTeX to reorganise your document by changing the order in which we call `\input` on our different `.tex` files. This means that the "next" or "previous" section is now completely different.

You will not have this problem by using a `\label` and associated reference commands.

17.2 Tip: Finding references with TEXstudio

TEXstudio offers some useful features to work with references:

- if you right-click on the reference of a `\ref`, `\nameref`, `\pageref`...you have the option to "Go to Definition"
- likewise, if you right-click on the reference of a `\label`, you have the option to "Find usages"
- when tying "`\ref{`" (or "`\nameref{`", "`\pageref{`"...), TEXstudio will list the different `labels` available
- TEXstudio will let you know if a `\label` has been declared more than once, or if a reference does not exist (by underlying the label id with a red or green squiggly line).

18 Tip: Counting pages

It is possible (with some caveats, as discussed below), to determine the total number of pages in LATEX: put a `\label` on the last page, and reference that label with `\pageref`.

Here are the details:

1. just before the `\end{document}` at the end of our document, put an empty element (e.g. a `\hphantom`[5], or an empty `\mbox`), followed by a `\label` [6]:

```
\mbox{}\label{LastPage}
\end{document}
```

   Note that if you insert a blank page (section 16.3) at the end, and you want to include blank pages in the total page number (probably a bad idea, but it's possible...), you would need to make the label as part of the last page:

```
\insertBlankPage[This page intentionally left blank.\label{LastPage}]
\end{document}
```

2. now refer to this label with `\pageref` :

```
This document contains \pageref{LastPage} pages.
```

And here is the result: "This document contains 81 pages."

Note: this can only work if your page number starts at 1, and you don't change it with something like a `\setcounter`:

```
\setcounter{page}{42}
```

[5] see: "15 — Phantom text"

[6] the reason why we put an empty element is, sometimes the index at the end of the document is more than one page long. We want an *invisible* object to serve as an anchor on the *last* page for our label.

19 Custom counters

LaTeX uses counters to keep a reference on all sorts of objects: parts, sections, subsections (section 9), tables (section 37), figures (section 41), etc.

In LaTeX it is also possible to create your own custom counter.

For example, if we were to write a book to teach a foreign language, we would probably incorporate some conversation examples. Then we might want to assign a number to this conversation, so that we can easily refer to it later (e.g. *"In conversation 2, Mr Smith used the* subjunctive mood *to indicate it is important that Mr Novak* be *present at the meeting on Monday."*)

For that purpose, we can define a new counter named "`conversation`", using the `\newcounter` command:

```
\newcounter{conversation}
```

If we want to reset the counter for every section (i.e. every time the "`section`" counter is increased), then we will declare the "`conversation`" counter like this:

```
\newcounter{conversation}[section]
```

To increase the counter, we use the `\refstepcounter` command:

```
\refstepcounter{conversation}
```

Finally we can output the current counter value with `\the`*`countername`*. In our example, *`countername`* will be equal to `conversation`:

```
Conversation: \theconversation \label{conv:shopping}
```

Note that in this code, we used the `\label` command to defined a label (`conv:shopping`). This label will take the value of the latest counter that was used (i.e. `\theconversation`, the current value of counter `conversation`).

Now we can use this as a reference, using the `\ref` command:

```
In conversation \ref{conv:shopping}, Mr Smith used the subjunctive mood
to indicate ...
```

For more information about counters, see:
`https://www.overleaf.com/learn/latex/Counters`

20 Quotation marks

While we are on the subject of quoting and referencing, here is how to make quotation marks in LaTeX:

Table 1: Quotation marks

`` `Single quotes' ``	→	'Single quotes'
`` ``Double quotes" ``	→	"Double quotes"
`` ``Double quotes'' ``	→	"Double quotes"

I often use double quotes when referring to sections by name.
For example: *"Cross-references"*.

21 Quotes

The following document shows how to add quotes using packages like `epigraph`, `fancychapters`, `quotchap`, `dirtytalk`, or `csquotes`: (Overleaf, 2022c)

`https://www.overleaf.com/learn/latex/Typesetting_quotations`

22 Adding a bibliography

In *"Cross-references"*, we've seen how to add references that point to different sections *within* our document. Now we must also learn how to quote source *outside* our document (e.g. books, articles, journal…). This is the role of the bibliography.

A few things to know about bibliography styles in LaTeX:

1. there exists different styles of bibliography
2. there also exists several bibliography packages in LaTeX (`bibtex`, `biblatex`, `natbib`)
3. each library provides different bibliography styles readily available
   - therefore, your choice of library may be dictated by the types of bibliography styles they provide
4. those different libraries are not compatible with each other
   - if you were to switch libraries, you might have to delete a few generated files, like the `.bbl` and `.aux` one, and then recompile the document

22.1 Bibliography file

Whichever bibliography library you choose, you will need to specify at least one bibliography file (`.bib`) to store all your external references.

Here is a short example:

As you can see, a reference has a type (`@book`), an `id`, and a list of "fields" (`key = value,`).

The `id` is important, as it allows to identify the reference to cite. Unlike with cross-references (section 17), a bibliography `id` cannot contain things like spaces, punctuation, etc. (only alphanumeric characters, dashes -)

In this example, the fields `language`, `date`, and `isbn` do note seem to be used by `natbib`. However, `natbib` allows to use `note` to add some information to the reference, so you can use that as a workaround.

`biblatex` doesn't use `note`, but makes use of the other fields.

> A bibliography file may contain many references, but only the references that have been cited at least once will appear in the bibliography section of the document.

This means it is possible to share the same bibliography file with several LaTeX documents (think of it as a centralised "database" for all your external references).

Figure 6: bibliography.bib

```
@book{c,
  author = {Kernighan B., Ritchie D.},
  title = {C Programming Language},
  year = {1988},
  publisher = {Prentice Hall},
  edition = {2nd},
  note = {December 1988, \emph{ISBN-13: 978-0131103627}},
  language = {English},
  date = {December 1988},
  isbn = {978-0131103627}
}

@book{c-prog,
  author = {Kernighan \& Ritchie},
  title = {C Programming Language},
  year = {1988},
  publisher = {Prentice Hall},
  edition = {2nd},
  note = {December 1988, \emph{ISBN-13: 978-0131103627}},
  language = {English},
  date = {December 1988},
  isbn = {978-0131103627}
}
```

22.2 Bibliography, APA-like style, with `natbib`

There exists different types of bibliography, but I have a preference for the Harvard APA style.

I find this style clearer. Besides, this style seems to be used by a number of learning institutes (e.g. the University of Portsmouth[7] , the University of Adelaide[8] ...)

In this article, we will try to get as close as possible to the Harvard APA style of bibliography, using LaTeX.

For this, we will use the `natbib` package.

> To use `natbib`, make sure the bibliography tool is set to `BibTeX`.
>
> In TeXstudio, go to “Options” » “Configure TeXstudio”; then in the “Build” menu, change “Default Bibliography Tool” to “BibTeX”.

Call this in the preamble:

```
\usepackage{natbib}
```

Then, in the body of the document, call the following where you want the bibliography to be printed:

```
\bibliographystyle{apalike}
\bibliography{content/bibliography}
```

Note:

> When calling `\bibliography`, *DO NOT* specify the `.bib` extension in the path.
>
> Also note that, similarly to the `\input` command, the path is relative to the *root* document, not the current source file.

[7] see: `https://www.city.ac.uk/__data/assets/pdf_file/0017/77030/portsmouth_harvard_guide.pdf`
[8] see: `http://maths.adelaide.edu.au/anthony.roberts/LaTeX/ltxxref.php`

Only the references that have been cited at least once in the document will appear in the bibliography. So the next step will be to add some citation.

There are many ways to cite a reference:

Table 2: Citing

	id:	c	c-prog
	\citeauthor{id}	Kernighan B.	Ritchie
	\citeyear{id}	1988	1988
	\cite{id}	Kernighan B. (1988)	Ritchie (1988)
(natbib)	\citefullauthor{id}	Kernighan B.	Ritchie
(natbib)	\citeyearpar{id}	(1988)	(1988)
(natbib)	\citep{id}	(Kernighan B., 1988)	(Ritchie, 1988)
(natbib)	\citealt{id}	Kernighan B. 1988	Ritchie 1988
(natbib)	\citealp{id}	Kernighan B., 1988	Ritchie, 1988
(natbib)	\citet{id}	Kernighan B. (1988)	Ritchie (1988)

As shown in table 2, some commands are only defined in the `natbib` package. Trying to use them with `biblatex` will throw an error.

22.3 Customising the Bibliography title (`natbib`)

The `natbib` package allows you to customise the bibliography title.

This is done by redefining the `\bibsection` command:

```
% Bibliography title can be customised
% Default:  \section*{\currentPart}
\renewcommand{\bibsection}{\section*{External References}}
```

```
The Bibliography title can be customised
\renewcommand{\bibsection}{\section*{External references}}
```

This is a change from its default implementation, which would look like this:

```
\renewcommand{\bibsection}{\section*{\currentPart}}
```

22.4 Bibliography, using `biblatex`

The `biblatex` library provides the following styles out of the box: `alphabetic`, `authortitle`, `authoryear`, `draft`, `numeric`, `reading`, and `verbose`.

Here is an important note if you want to use `biblatex` with TeXstudio:

> To use `biblatex` in TeXstudio, you will need to configure the bibliography tool to `biber`.
>
> Go to “Options” » “Configure TeXstudio”; then in the “Build” menu, change “Default Bibliography Tool” to “Biber”.

In the preamble:

```
\usepackage[utf8]{inputenc}
\usepackage[english]{babel}
\usepackage[style=authoryear]{biblatex}

%% Specify the path to the bibliography file
\addbibresource{content/bibliography.bib}
```

Here, we specified the `style` to be `authoryear` but other styles exist, as shown in Overleaf (2022a) :

`https://www.overleaf.com/learn/latex/Biblatex_bibliography_styles`

Then in the document, print the bibliography with `\printbibliography`:

```
\printbibliography
```

Part VIII
Text styles, indexes

In this part, we'll talk about custom text styles and adding indexes. This will be useful when we learn how to write our first macros, in section "26 — Your first custom macros!".

23 *Emphasis*, italics, bold…

A recurring question from people new to LaTeX is: *"how do I put text in **bold face?"*** when the underlying question really is *"how can I* emphasise *some important idea?"*

In LaTeX, there is a command specifically used for *emphasis*: `\emph`
The difference is, by default in LaTeX, *emphasis* is not made using **bold face**, but using *italics* instead.
Also, *if you were to* emphasise *some text inside of another* emphasis*, then the inner* emphasis *will appear* upright *(and not in italics), as if the two* emphases *had cancelled each other…*

So we advise you to `\emph` for emphasis in LaTeX.

However, if you really *must* specifically put your text in bold, italics, underline, etc., then here are some commands for that:

`\emph{text}`	*text*	*emphasis*
`\textit{text}`	*text*	*italics*
`\textbf{text}`	**text**	**bold face**
`\texttt{text}`	`text`	`monospaced font` (as configured in section 30)
`\underline{text}`	<u>text</u>	<u>underlined</u>

24 Adding an index

First, we need to use the `imakeidx` package and call the `\makeindex` command:

```
%%% In the preamble
\usepackage{imakeidx}
\makeindex
```

Then we need to use the `\index` command, to add an index on the relevant word:

```
Some text here, mentioning a keyword to be indexed. \index{keyword}
```

Finally, we need to print the index with `\printindex` :

```
\printindex
```

The difficult part is to specify an index whenever certain words are mentioned. For this purpose, we can use macros to make our task easier…

To see what an index looks like, go to page 81 (for this is where you will find the index for this document).

Part IX
Introduction to Macros

Some quick notes, before we start:

I usually make the following distinction:

- I call *“command”* the commands that are provided by TEX or a LATEX package
- I call *“macro”* the commands that are defined by the user, and are not part of a package

Also, I use the word “parameter” instead of “argument”. Indeed, talking about *“long vs. short arguments”* or *“optional argument”* sounded a bit strange. Unlike my ex, I don’t like arguments…

25 Some commands we used

So far, we’ve used a few LATEX commands without explaining what they are used for. Some of those commands take no parameters, while others take at least one parameter (and maybe an optional parameter).

Let’s take a look at those commands that are called without parameters, starting with `\dots`.

25.1 Calling a command with no parameters

The `\dots` command prints 3 little dots…
In *text* mode, `\dots` outputs the same as `\ldots` or `\textellipsis` (while in *math* mode, `\dots` tries to determine whether to use `\ldots` or `\cdots` based on context).
Now let’s try to use it…

Table 3: Command `\dots`

	LATEX code	Output
1	`Three little dots\dots then some text.`	Three little dots…then some text.
2	`Three little dots\dots\ then some text.`	Three little dots… then some text.

In example 1 we see that, in the output, there is no space between “dots…” and “then”. In the code, there needs to be a space; otherwise LATEX will try to find a command called “`dotsthen`” which doesn’t exist.
In example 2, we escaped the space (“`\ `”) and the space appeared in the output.

Here are some more examples, this time with `\textbackslash` (which, as the name implies, prints a backslash)…

Table 4: Command `\textbackslash`

	LaTeX code	Output
3	`\textbackslash dots`	\dots
4	`\textbackslashdots`	
5	`\textbackslash\textbackslash`	\\
6	`\textbackslash \textbackslash`	\\
7	`\textbackslash\ \textbackslash`	\ \

Example 3 wrote "\dots".
Example 4 threw an error: "`Undefined control sequence. \textbackslashdots`"
Example 5 wrote a pair of backslashes (\\ ).
So did example 6.
Example 7 wrote "\ \", with a space between the backslashes.

In summary, if there needs to be a space after the output of a command *with no parameter*, then that space needs to be escaped in the code ("`\ `").

Commands that are called with parameters do not have this issue, as illustrated here:

`\textbackslash\textbf{re}newcommand` **re**newcommand

Table 5: Other commands with no parameters

`\LaTeX`	Prints the word "LaTeX", using "confused casing"
`\medskip`	Leaves a medium vertical space.
`\bigskip`	Leaves a big vertical space.

25.2 Calling commands with parameters

We've seen several examples of commands accepting parameters throughout this article.

We will go into more details as we learn how to define (and call!) our own macros in "26 — Your first custom macros!"…

26 Your first custom macros!

26.1 Macros without parameters

`\newcommand{\cmd}{definition}`
: Defines a new command.
Will throw an error if the command already exists.

`\renewcommand{\cmd}{definition}`
: Redefines a command.
Will throw an error if the command does not exist.

Historically, `\def\cmd{definition}` was used to define or redefine commands.

> *DO NOT* use `\def` nowadays, as it does not check if the command already exists or not. Use `\newcommand` and `\renewcommand` instead.

If the command already exists, you *WANT* `\newcommand` to throw an error (as you are likely to replace a command used by some package, resulting in errors). Conversely, if the command does not exist, you *WANT* `\renewcommand` to throw an error (you might have mistyped the name, so the command is not replaced at all).

When a word appears often in a document, it is usually a good idea to create a macro specifically for that word, especially if we are in either of these cases:

- the word requires a specific formatting (e.g. emphasis, monospaced fonts, bold…), which must be consistent throughout the document.

  Formatting includes adding things such as:

  - `\texttrademark` (™) for Trademarks
  - `\textregistered` (®) for Registered Trademarks
- the word needs to be indexed, i.e. every page where the word appears needs to be listed in the index (see: “24 — Adding an index”)

In doing so, it is easy to change every occurrence of a word simply by modifying the corresponding macro.

Below are some examples of macros, which we can call whenever we need to refer to the terms they represent:

- as a first example, we will write a macro to print and add an index on recurring terms like: “TEXstudio” or “WYSIWYG”
- our next example will display a specific with a consistent format, in that case some “brand name” with a trademark or registered trademark (note: we will also include the trademarks when indexing the names).

  Earlier in this article I talked about the (now defunct) company Iomega®, makers of the Zip™ drive (now an obsolete product). So let’s use that as an example.

Here is how defined some macros to format and index those recurring terms:

```
\newcommand{\TeXstudio}{\TeX studio\index{TeXstudio}}
\newcommand{\WYSIWYG}{WYSIWYG\index{WYSIWYG}}
\newcommand{\WYSIWYM}{WYSIWYM\index{WYSIWYM}}

\newcommand{\Iomega}{Iomega\textregistered\index{Iomega\textregistered}}
\newcommand{\Zip}{Zip\texttrademark\index{Zip\texttrademark}}
```

> As explained in "8 — Organising your files", out of convenience we put many of our user-defined macros in the same source file, located at:
> `include/macros.tex`
>
> This makes it easy to find the list of all our macros, recurring terms, and how they are displayed throughout the document.

Of course, as our list of macros gets bigger, it is possible to create more source files and group our macros by category[9]

26.2 Macros with parameters

In section 26.1, we saw how to define macros without parameters, why we would want to create such macros, and how to call them (section 25.1).

In this section, we will see how to create macros that take parameters. These are useful when you repeat the same tasks, but with different parameters.

For that, we will go step by step…

`\newcommand{\`*`cmd`*`}[`*`nbArgs`*`]{`*`definition`*`}`
: Defines a new command.
Will throw an error if the command already exists.
`nbArgs` is the total number of parameters (maximum: 9)
In *`definition`*, the parameters' values are represented by the variables `#1` to `#9`.

As an example, we will create a macro that will take 1 parameter: a label name (as seen in "17 — Cross-references"), and then display both the `\ref` and `\nameref` of that label.
We will call this macro `\longref` :

```
%% Note the use of non-breaking spaces (~) and long dash (---)
\newcommand*{\longref}[1]{\ref{#1}~---~\nameref{#1}}
```

With this macro, a simple call like ``` ``\longref{cross-references}" ``` will display: "17 — Cross-references"

As you might have noticed, we used `\newcommand*` (with a star) instead of d `\newcommand` (without a star) to define our macro. The difference is explained in the next section…

26.3 Long vs short parameters

In LaTeX, most of the macros we write do not need to accept paragraphs, only short amount of text or some small values.
This is the difference between "short" parameters (i.e. do not accept paragraphs), and "long" parameters (i.e. can accept paragraphs).

When using `\newcommand*` (with a star) a check is added, to make sure the macro can only accept short parameters .

[9] for this article, each code sample that needed to be featured got its own separate source file. This way it's easy to load the source file with `\input`, and show its code with `\lstinputlisting`.

`\newcommand{\cmd}[nbArgs]{definition}`
: Defines a new command.
Will throw an error if the command already exists.
Parameters can contain paragraphs.

`\newcommand*{\cmd}[nbArgs]{definition}`
: Defines a new command.
Will throw an error if the command already exists.
Will throw an error if any *of the parameters contains a paragraph.*

26.4 Macros with one optional parameter

It is possible to define macros that accept multiple optional parameters by *"chaining"* several macros that each take one optional parameter (basically, *"currying"*).

However, this technique is complicated, and the order of the optional parameters is fixed (so if you specify an optional parameter, you'll also need to specify the optional parameters that came before it, which defies the purpose of optional parameters).

Another approach is to have only one optional parameter containing *key-values*. This approach is used by a number of LaTeX packages. However, it is rather complex to explain, and is out of the scope of this article.

Just be aware that these techniques exist, and they rely on declaring macros with *one* optional parameter.

To create a macro with a single optional parameter, we need the following constructs:

```
\newcommand{\cmd}[nbArgs][optionDefault]{definition}
\newcommand*{\cmd}[nbArgs][optionDefault]{definition}
```

Specifying *optionDefault* means that:

- the command has an optional parameter
- its value is represented by `#1` (hence `#2` to `#9` are the values of mandatory parameters)
- *optionDefault* is the default value for `#1`.
  `#1` will take the value *optionDefault* if the optional parameter is *not set* (i.e. not specified at all). If you call the command with an empty parameter (`[]`), then `#1` will be empty (i.e. not set to *optionDefault*)

As a result:

- LaTeX allows to declare commands with *one* optional parameter (but you can chain commands together)
- specifying *optionDefault* doesn't mean that `#1` will never be empty.
  You can call the macro with `[]` as the optional parameter, in which case `#1` will be empty. If this is not what you expect, you might need to check whether `#1` is empty (or consider making the parameter mandatory).
- if you need more than one optional parameters, you will need to use more advanced techniques, like using *key-values* for the optional parameter

> Explaining how to test a value, or handle *key-values* parameters in LaTeX is out of the scope of this article.
>
> However, it is important to know about that the option parameter exists, how it's called, and know its limitations.

Many LaTeX commands make use of the optional parameter (usually as a set of *key-values*).

Examples found in this article include: `\documentclass`, `\usepackage`, `\lstinline`, `\lstinputlisting`, `\newcommand`, `\renewcommand`...

When calling a command with an optional parameter, the optional parameter is specified in square brackets `[ ]` .

For example:

```
\usepackage[super]{nth}
\usepackage[authoryear]{natbib}

...
\lstinline[language=tex]|some code here|
\lstinputlisting[language=tex]{path/to/file}
```

In section "16.3 — Inserting a blank page", we've seen an example of a macro that takes *one* optional parameter (not a *key-value*):

```
\newcommand{\insertBlankPage}[1][This page intentionally left blank.]{
\newpage
\thispagestyle{empty} % remove header and footer for current page
\vspace*{\fill}
\begin{center}
#1
\end{center}
\vspace*{\fill}
\newpage
}
```

Here are some examples of calls, and their effect:

```
% To display a blank page with a custom message
\insertBlankPage[Some text]

% To display a blank page
% with the default "This page intentionally blank."
\insertBlankPage

% In this case, the message will actually be blank
\insertBlankPage[]
```

This example illustrates the effect of the optional parameter:

- no square bracket → `#1` is set to the value of `optionDefault`
- square bracket present → `#1` is set to the value inside the square bracket, which can be an empty string.

26.5 Redefine macros

Redefining commands is done through `\renewcommand`:

`\renewcommand{\cmd}{definition}`
`\renewcommand*{\cmd}{definition}`
: Redefines a command.
Will throw an error if the command does not exist.

These commands do the same as their `\renewcommand` counterparts, except that they will *throw an error if the command does already not exist*.

We have seen a few examples of `\renewcommand` being used.

Indeed, several packages allow for *some* degree of customisation by letting you redefine some of the commands that they use.

This is notably the case in:

- “10.1 — Customising the Table of Contents”
- “22.3 — Customising the Bibliography title (`natbib`)”

Another possible use for `\renewcommand` would be to show how knowledge evolves as you progress through your document.

Let’s say for example that we are writing a book to teach Japanese from a beginner’s level…

At first, the beginner has no knowledge of *hiragana* and *katakana*, so we will add some *furigana*: little characters that can be used to indicate the pronunciation. For this, we will use the `ruby` package.

As the reader progresses through the book, they will get acquainted with more and more characters, and we will progressively remove those pronunciation guides.
So at the beginning, we will define commands that represent our current state of knowledge (where we need some pronunciation guides for *every* character).

To print the word for “hello” (*konnichiwa*), we will need the following commands:

```
\newcommand{\KO}{\ruby{こ}{ko}}
\newcommand{\N}{\ruby{ん}{n}}
\newcommand{\NI}{\ruby{に}{ni}}
\newcommand{\CHI}{\ruby{ち}{chi}}
\newcommand{\WApart}{\ruby{は}{wa}}
```

Then we use those commands:

```
\KO\N\NI\CHI\WApart。
```

…and it will print: こ(ko) ん(n) に(ni) ち(chi) は(wa)。

As you can notice, every character has a *furigana* (pronunciation guide) on top…

Later in the book, we will teach some new characters, and expect the student to remember them; so we will remove the pronunciation guides.

For example, after we've introduced the lessons about に[ni] and ち[chi], we'll remove the pronunciation guides for these two characters, by redefining the corresponding commands:

```
\renewcommand{\NI}{に}
\renewcommand{\CHI}{ち}
```

From then on, "`\KO\N\NI\CHI\WApart。`" will render as: こ[ko]ん[n]にちは[wa]。

As expected, the *furigana* have been removed from に and ち.

We can do the same (but in reverse), to show how many characters have been covered as we progress: we would make a character table that highlights which characters have been studied yet, and show this table several times throughout the book (we'll inport the same file at different places in the book, thanks to the `\input` command).

As we go through the book, more and more characters will become highlighted, therefore showing the overall progression.

This is just an example, and there are better implementations for our commands showing Japanese characters (which fall out of the scope of this article).

> The key takeaway is that the `\renewcommand` command (combined with `\input`) can be used to illustrate our progression through the document (e.g. list of topics that have been covered).

26.6 Summary

```
\newcommand{\cmd}{definition}
\newcommand*{\cmd}{definition}
\newcommand{\cmd}[nbArgs]{definition}
\newcommand*{\cmd}[nbArgs]{definition}
\newcommand{\cmd}[nbArgs][optionDefault]{definition}
\newcommand*{\cmd}[nbArgs][optionDefault]{definition}
\renewcommand{\cmd}{definition}
\renewcommand*{\cmd}{definition}
\renewcommand{\cmd}[nbArgs]{definition}
\renewcommand*{\cmd}[nbArgs]{definition}
\renewcommand{\cmd}[nbArgs][optionDefault]{definition}
\renewcommand*{\cmd}[nbArgs][optionDefault]{definition}
```

`newcommand`
: Defines a new command.
Will throw an error if the command already exists.

`renewcommand`
: Redefines a new command.
Will throw an error if the command does NOT already exist.

`*`
: Makes command accept only *short* parameter. The command will throw an error if any of the parameters contains a paragraph.

`\cmd`
: The command name (\ is required)

`[nbArgs]`
: *Optional* (default: 0)
Specifies the number of parameters to the command (including optional parameter)

`[optionDefault]`
: *Optional*
If specified, the command accepts an optional parameter, with *optionDefault* as its default value.

definition
: The command definition

27 Environment

In “11.3 — Putting text in a frame”, we created an environment and showed that what happens within that environment was *isolated* from the rest of the document.

This is in contrast to commands. A command may change some values, and those changes may affect the rest of the document.

That is the main difference between a command and an environment.
Other differences have to do with the way an environment is declared and called.

We have seen a couple environments already (`abstract`, `mdframed`), and will see a few more by the end of this document (`itemize`, `enumerate`, `description`, `table`, `longtable`, `tabular`...)

An environment starts with `\begin{`*envname*`}` and ends with `\end{`*envname*`}`. Everything in-between those two tags constitutes the content of the environment:

```
\begin{mdframed}
Help! I've been framed!
\end{mdframed}
```

When declaring an environment (with the `\newenvironment` command), you need to specify what to do at the *beginning* and at the *end* of the environment (with the content being processed in-between):

```
\newenvironment{blueframe}[1]{
\mdfsetup{
  frametitle={
    \tikz
    \node[rectangle,fill=blue!20]
    {#1};
  },
  linecolor=blue!20,
  linewidth=2pt
}
\begin{mdframed}}
{\end{mdframed}}
```

The rest (parameters, optional parameter, default value...) is very similar to macros (section 26). Below are the different commands to define (and redefine) an environment:

```
\newenvironment{envname}{begdef}{enddef}
\newenvironment*{envname}{begdef}{enddef}
\newenvironment{envname}[nbArgs]{begdef}{enddef}
\newenvironment*{envname}[nbArgs]{begdef}{enddef}
\newenvironment{envname}[nbArgs][optionDefault]{begdef}{enddef}
\newenvironment*{envname}[nbArgs][optionDefault]{begdef}{enddef}
\renewenvironment{envname}{begdef}{enddef}
\renewenvironment*{envname}{begdef}{enddef}
\renewenvironment{envname}[nbArgs]{begdef}{enddef}
\renewenvironment*{envname}[nbArgs]{begdef}{enddef}
\renewenvironment{envname}[nbArgs][optionDefault]{begdef}{enddef}
\renewenvironment*{envname}[nbArgs][optionDefault]{begdef}{enddef}
```

27.1 Numbered environment

To create a numbered environment, we need to define and use counters, as explained in section “19 — Custom counters”.

See also:
`https://www.overleaf.com/learn/latex/Environments#Defining_a_new_environment#Numbered_environments`

Part X
Customising the style

So far we've focused mainly the features to help you write the *content* of the document (e.g. organising the LaTeX files, writing paragraphs, adding notes, a Table of Contents, managing an index, cross-references, a bibliography…) and writing macros to help you with repetitive tasks.

All these are very important to write ideas that flow naturally while being inter-connected with each other. So we've focalised on writing those ideas, and trusted LaTeX to produce good-looking documents for us, using its default style.

In this part, we'll finally learn how to add all the bells and whistles, and tweak the appearance of our document to our liking…

28 Making the document two-sided

Making a document two-sided is useful if that document is destined to be printed, and its pages are meant bound together.

This means that the style needs to change between even and odd pages: the inner margin (closer to the binding) needs to be wider, margin notes (section 11.2) need to be switched between left and right (to remain in the outer margin), etc.

To make a document two-sided, you need to specify the option `twoside` in `\documentclass`:

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

29 Changing the margins

Changing the margins is done with the `geometry` package:

```
%%% In the preamble
\usepackage{geometry}
\geometry{
  a4paper,
  total={170mm,257mm},
  top=30mm,
  bottom=30mm,
  inner=35mm,
  outer=35mm,
  marginparwidth=20mm,
}
```

With the `\geometry` command, we specify things like the paper size (`a4paper`), as well as the `top` and `bottom` margins (which are self-explanatory).
Similarly, we could also specify the `left` and `right` margins, but instead we set the `inner` and `outer` margins.

In the case of a two-sided document (section 28) used for printing and binding, `inner` refers to the margin closer to the binding, while `outer` refers to the margin that is away from it. If the document is meant to be printed and bound, then the `inner` margin should be slightly bigger than the `outer` one (to leave

enough room for the binding).

In the case of a one-sided document (i.e. not meant for printing and binding), `inner` and `outer` simply refer to `left` and `right` margins respectively.

As we have modified the margin sizes, then we must also adjust the `marginparwidth` size, otherwise the "Margin notes and paragraphs" (section 11.2) might not fit in the margin anymore.

If you're not using margin notes or margin paragraphs, there is no need to specify `marginparwidth`. Otherwise, `marginparwidth` should be smaller than the value for the `outer` margin.

30 Changing the fonts

Changing the fonts for our document is done with the `fontspec` package:

```
%%% In the preamble

\usepackage{fontspec}

\setromanfont{DejaVu Serif}
\setsansfont{DejaVu Sans}
% \setmonofont{DejaVu Sans Mono}

\setmainfont{DejaVu Sans}
```

Here, we select one font per typeface:

Roman
: *a.k.a Serif*
A font that contains *serif*, the small lines or strokes at the end of larger strokes or characters. You can set the *Serif* font using `\setromanfont`

Sans
: *a.k.a Sans-Serif, Gothic*
A font that contains *no* serif. You can set the *Sans-Serif* font using `\setsansfont`

`Monospaced`
: *a.k.a Typewriter*
A font in which all characters are of equal horizontal space. This is important for showing computer code, input files, CSV files, etc. (any application where we need to easily count characters, and see how each character would align in a grid). You can set the *Monospaced* font using `\setmonofont`

Finally, we set the main font for the document with the `\setmainfont` command.

Fonts with *serif* look aesthetically pleasing on printed documents, while *sans serif* fonts tend to be easier to read on screens[10].

This is why I prefer to use *sans serif* fonts for documents like this one, meant to be read on-line rather than being printed.

[10] most modern printers have a print resolution in excess of 600 Dots-Per-Inch; for comparison, an iPhone 13 has a screen density of 460 Pixel-Per-Inch

In most LaTeX document classes, the default style usually uses a *serif* font as the main font, specifically because *serif* fonts look better in print.

For this reason, calling the `\setromanfont` command will also reset the main font for the document.
That's why it is advisable to call `\setmainfont` only *after* `\setromanfont`.

In our example, we set both the *serif* and *sans-serif* fonts, but keep the default *monospaced* provided by LaTeX.

31 Customising headers and footers

To define our headers and footers, we will use the `titleps` package, which I find much easier to use than `fancyhdr`.

For this section, we will take a look at how headers and footers were defined for this article, and analyse the example:

```
%%% In the preamble

\newcommand{\copyrightNotice}{\color{gray}{\emph{\textcopyright\ 2022
    ↪ Pierre S. Caboche. All rights reserved.}}}

\usepackage{titleps}
\usepackage[table]{xcolor}

\newpagestyle{main}{
  \sethead
  % even
  [\thesection\ --- \sectiontitle]
  []
  []
  % odd
  {}
  {}
  {\thesection\ --- \sectiontitle}

  \setfoot
  % even
  [\textbf{\thepage\ \color{lightgray}{|}}]
  []
  [\copyrightNotice]
  % odd
  {\copyrightNotice}
  {}
  {\textbf{\color{lightgray}{|} \color{black}{\thepage} }}

  \headrule
  %\footrule
}
\pagestyle{main}

\widenhead[25pt][25pt]{25pt}{25pt}
```

The configuration of headers and footers is done in the preamble of the document. The first thing we do is use the `titleps` package.

Then we use the `xcolor` package because we want to use the colour `lightgray` in our footer (note: we will need the `[table]` option in section *34.1 — Tabular features*) .

Then we define the *page style* "main" with the `\newpagestyle` command.

In \newpagestyle, we define our headers and setters, using \sethead and \setfoot. These commands have the following signature:

```
\sethead[evenLeft][evenCenter][evenRight]{left}{center}{right}
\setfoot[evenLeft][evenCenter][evenRight]{left}{center}{right}
```

These 2 commands each have:

- 3 mandatory parameters: *left*, *center*, *right*
- 3 optional parameters: *evenLeft*, *evenCenter*, *evenRight*

Headers and footers are both divided in 3 parts: *left*, *center*, and *right*. The mandatory parameters apply to both *one-sided* and *two-sided* documents (section 28):

- if the document is *one-sided*, then *left*, *center*, and *right* define the text to be displayed on the *left*/*center*/*right* part of the header/footer *for all pages*
- if the document is *two-sided*, then:
  - *left*, *center*, and *right* only apply to *odd* pages
  - *evenLeft*, *evenCenter*, and *evenRight* only apply to *even* pages

In our headers/footers, we use the following commands:

\thesection	prints the section number
\sectiontitle	prints the section title
\thepage	prints the page number

The commands \headrule and \footrule add a rule, respectively: below the headline (\headrule), and above the footline (\footrule).

Now that the "main" *page style* defined, we set the *page style* to "main" using the \pagestyle command.

Finally, we use the \widenhead to make the headlines/footlines wider. Here is the syntax:

```
\widenhead[evenLeft][evenRight]{left}{right}
```

We could have simplified our header/footer definitions by using \sethead*, \setfoot*, and \widenhead* instead of \sethead, \setfoot, and \widenhead.

The *starred* version of these commands do not take the optional parameters; instead, for *even* pages, they use the same values as *left*, *center*, and *right*, but flipped:

```
\sethead*{left/evenRight}{center}{right/evenLeft}
\setfoot*{left/evenRight}{center}{right/evenLeft}
\widenhead*{left/evenRight}{right/evenLeft}
```

I used the *non-starred* version of these commands because they better illustrate how to define headers and footers (and the distinction better *even* and *odd* pages in the case of *two-sided* documents).

Part XI
Lists

32 List overview

This part will give you an overview (with examples) of the different types of list in LaTeX.

32.1 `itemize` and `enumerate`

The `itemize` environment creates bullet points.
The `enumerate` environment creates enumerated lists.
Below are some examples where we mix `itemize` and `enumerate`:

Code

```
\begin{itemize}
  \item item 1
  \begin{itemize}
    \item item 1.1
    \item item 1.2
  \end{itemize}
  \item item 2
  \begin{enumerate}
    \item item 2.1
    \item item 2.2
  \end{enumerate}
\end{itemize}
```

Result

- item 1
  - item 1.1
  - item 1.2
- item 2
  1. item 2.1
  2. item 2.2

```
\begin{enumerate}
  \item item 1
  \begin{enumerate}
    \item item 1.1
    \item item 1.2
  \end{enumerate}
  \item item 2
  \begin{itemize}
    \item item 2.1
    \item item 2.2
  \end{itemize}
\end{enumerate}
```

1. item 1
   (a) item 1.1
   (b) item 1.2
2. item 2
   - item 2.1
   - item 2.2

32.2 `description`

The `description` environment allows to describe terms:

Code

```
  \begin{description}
    \item[label 1] description
        ↪ 1
    \item[label 2] description
        ↪ 2
    \item[label 3] \mbox{} \\
    description 3
  \end{description}
```

Result

label 1 description 1

label 2 description 2

label 3
description 3

Part XII
Code samples

33 Adding code samples

If you are writing an article or a thesis about computer science, chances are you will need to include some code samples in your document.

In LaTeX this is very easy, thanks to the `listings` package...

```
\usepackage{listings}
```

There are different ways to add code samples to a document:

Inline, using the `\lstinline` command:

```
\lstinline[language=tex]| your code here...|
```

For example:

```
Inline, using the \lstinline[language=tex]|\lstinline| command:
```

As a block, using the `lstlisting` environment

```
\begin{lstlisting}[language=tex]
  % some LaTeX code here...
\end{lstlisting}
```

From a source file, using `\lstinputlisting{}`

```
\lstinputlisting[language=tex]{path/to/source_file.ext}
```

As you might have noticed, those commands accept a list of optional parameters. We mainly use it the `language` parameter to specify the programming language, as in: `language=tex` .

It is possible to configure how the code shall be displayed. For example, here is what I used for this document[11] :

```
%%% In the preamble
\usepackage{listings}
\lstset{
   basicstyle=\fontsize{9}{9}\selectfont\ttfamily
  ,frame=lines
  ,tabsize=2
  ,keywordstyle=\bfseries\itshape,
  ,commentstyle=\itshape\color{teal}
  ,stringstyle=\color{magenta}
  ,breaklines=true,
  ,postbreak=\mbox{\textcolor{red}{$\hookrightarrow$}\space}
}
```

[11] I mean, this is the *actual* configuration that I used!
In fact, most of the LaTeX code featured in this document was also used to generate this document. To achieve this, I've put the code in a separate source files, which I then loaded with `\input` (for the commands to take effect). Then, whenever needed, I used the `\lstinputlisting` command to display the content of the source file in question.
In other words, I used LaTeX both to generate this document, as well as to show how this document was generated using LaTeX. This is quite powerful!

Note that, with `breaklines=true`, the `\lstinline` command may break the text if at the end of a line (and when `breaklines=false`, it might just write past the margin, which is not better).

To avoid that, you might occasionally want to replace `\lstinline` with `\texttt` (although that requires more work, e.g. replacing every `\` with `\textbackslash` and escaping other characters with a `\`).

Part XIII
Tables

In this part, we are going to give a quick introduction to "tables" in LaTeX.

There are many packages which allow to makes tables, each with different sets of options and features. This is a vast subject.

Please refer to the packages' respective documentation for more information.

34 tabular

The `tabular` environment is used to create tables and other kinds of grids.

Here is an example:

Table 6: Tic, Tac,Toe

X	O	X
O	X	X
X	O	O

Now let's take a look at the code for that grid:

```
\begin{tabular}{c | c | c}
X & O & X \\
\hline
O & X & X \\
\hline
X & O & O \\
\end{tabular}
```

In the `tabular` environment, we specify the alignment and separators for the cells (`{c | c | c}`).
Here are *some* values for the alignment (list non-exhaustive):

`l`	align left
`c`	center
`r`	align right
`p{`*width*`}`	makes the cells a `\parbox` of a certain *width*
`|`	a vertical line, to separate columns
`||`	a double vertical line, to separate columns

An horizontal line is drawn with the `\hline` command.

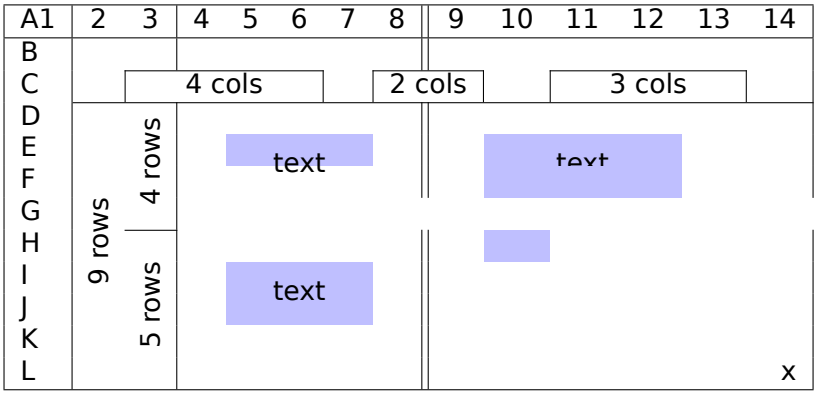
Finally, we need to specify the content of the grid:
Each cell is separated by an `&`, and each row is terminated by a `\\`.

These are the basics for using a `tabular`.

34.1 Tabular features

Now we'll look at a more complex example that shows some of the features of the `tabular` environment... Besides `tabular`, this example also requires the `xcolor` package.

Table 7: A more complex tabular



In table 7, we assigned a letter to each row, and we number each column. This makes it easier to understand certain commands (i.e. `\cline`, which takes column numbers as a parameter).

Now let's take a look at the code:

```
\begin{tabular}{| l | c  c | l  c  c  c  c || c  c  c  c  c  r |}
\hline
A1 & 2 & 3 & 4 & 5 & 6 & 7  & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\
\hline
B & & & & & & & & & & & & & \\
\cline{3-6} \cline{8-9} \cline{11-13}
C &
& \multicolumn{4}{|c|}{4 cols}
&
& \multicolumn{2}{|c|}{2 cols}
&
& \multicolumn{3}{|c|}{3 cols} &
\\
\cline{2-14}
D
& \multirow{9}{*}{\rotatebox{90}{9 rows}}
& \multirow{4}{*}{\rotatebox{90}{4 rows}}
& & & & & & & & &
\\
E & & & &
\multicolumn{3}{c}{\multirow{2}{*}{\cellcolor{blue!25}text}}
& & &
\multicolumn{3}{c}{\multirow{2}{*}{\cellcolor{blue!25}text}}
& &
\\
F & & & & & & & & & \multicolumn{3}{c}{\cellcolor{blue!25}} & &
\\
G & &
\\
\cline{3-3}
H & & \multirow{5}{*}{\rotatebox{90}{5 rows}}
& & & & & & \cellcolor{blue!25} & & & &
\\
I  & & & & \multicolumn{3}{c}{\cellcolor{blue!25}}
& & & & & &
\\
J  & & & & \multicolumn{3}{c}{\multirow{-2}{*}{\cellcolor{blue!25}text}}
& & & & & &
```

```
\\
K & & & & & & & & & & & &
\\
L & & & & & & & & & & & & x
\\
\hline
\end{tabular}
```

In the `tabular` environment, we define our column list and their alignment.

Column separators are specified with a `|` in the list of columns. Notice how on row G the vertical lines are broken (including the right border). This is because we didn't specify enough `&`. If the cell is not specified (even as an empty cell), then the column separators won't appear.

The `\hline` command draws a line that spreads the whole width of the table.

For finer control, we use the `\cline` command. For example, `\cline{3-6}` draws a line from column 3 to column 6 (both included).

We notice a number of cells that occupy more than one column. This is done using the `\multicolumn` command.

For example in C3-6, we do:

```
\multicolumn{4}{|c|}{4 cols}
```

This creates a cell that spans 4 columns, with its content centered and cell separators on both sides (`|c|`).

> Note that a `\multicolumn` may also "span" only 1 column:
>
> `\multicolumn{1}{c}{text}`
>
> This is mainly used to change the alignment or the borders for that particular cell (e.g. make the cell content *centered* in an otherwise *left*-aligned column), in table headers for instance.

Now that we spanned a cell over multiple columns, we're going to do the same over multiple rows. For that, we will need the `multirow` package:

```
%%% In the preamble
\usepackage{multirow}
```

This package provides us with the `\multirow` command

We will also need to rotate the content of the cell to make it vertical. This is done using the `\rotatebox`, which is part of the `graphics` package (note: this might already be loaded if you are using a package like `pgfplots`, see section 39).

```
%%% In the preamble
\usepackage{graphics}
```

In D3, we define a cell that spans 4 rows, and rotate the text 90° counterclockwise:

```
\multirow{4}{*}{\rotatebox{90}{text}}
```

Next in E5 (and other places), we define a cell that spans multiple columns *and* multiple rows:

```
\multicolumn{3}{c}{\multirow{2}{*}{text}}
```

As we'll see, \multirow is harder to work with than \multicolumn...

Next, we change the colour of a cell by using the \cellcolor command. To use \cellcolor, we will need to call the xcolor package with the table option:

```
%%% In the preamble
\usepackage[table]{xcolor}
```

We paint cell H10 with the colour blue!25 (i.e. "blue, 25% saturation") by doing this:

```
\cellcolor{blue!25}
```

Then we try to colour cells that span multiple columns/rows, with various degree of success...

\cellcolor works very well with \multicolumn, but not \multirow:

- as you can see in cell E5-F7, only the first row (E5-7) gets painted
- in E10-F12, we try to paint the second row (F10-12), but then the text gets "eaten" away
- in I5-J7, we painted the first row (I5-7), then made a \multirow on the second line (J5-7) with a *negative* number of rows (so the text gets printed *over* the previous row). It's a really ugly workaround, though...

So as you can see, working with \multirow (and some other advanced features) is not always easy.

> As with many things in LaTeX, it's often best to keep your tables simple, and focus on what's important: deliver clear and impactful ideas without relying on complicated features.

34.2 Paragraphs a inside table

In the next sample, we will create a table with cells that can accept paragraphs as their content:

Table 8: Paragraphs inside a tabular

A1	2	3	4
B	Multiple rows...		This cell can contain a whole paragraph! If there is a need to add a newline, use the \newline command.
C		As you can see, we can colour the cell very easily!	

Here is the code:

```
\begin{tabular}[t]{| l | c | p{3cm} || p{5cm} |}
\hline
A1
& \multicolumn{1}{|c|}{2}
& \multicolumn{1}{c||}{3}
& \multicolumn{1}{c|}{4}
\\

\hline
B
& \multirow{12}{*}{\rotatebox{90}{Multiple rows\dots}}
&
&
This cell can contain a whole paragraph! \newline
\newline
If there is a need to add a newline, use
the \texttt{\textbackslash newline} command.
\\

C
&
&
\cellcolor{blue!25}
As you can see, we can colour the cell very easily!
&
\\
\hline
\end{tabular}
```

As you can see in table 8:

- in the column definition, we use `p{`*width*`}` (e.g. `p{3cm}`).
  This forces you to specify the column's width, but allows you to put long texts in your table.
- to add a new line, use the `\newline` command (`\\` will not work)
- we didn't need a `\multirow` for cell C3, so colouring it was easy (unlike in table 7, where we had to use some workarounds).
- for cells B2-C2, where we tried to display some text rotated at 90 degrees, We used a `\multirow`.
  Noticed however that the number of rows it spans had to be tweaked manually (it's not 2, it's 12). Colouring it is also complicated (as always with `\multirow`)

35 `longtable`

A `longtable` is used for tables which might stretch over several pages.

To use a `longtable`, you first need to load the `longtable` environment:

```
%%% In the preamble
\usepackage{longtable}
```

In a `longtable` environment you may define the following:

- the *preamble*, which comprises:
  - *caption* and *label*
    These are used for cross-references (section 17), and to show the long table in the list of tables (section 38)

  - the *first head*
    This is the header which appears at the beginning of the table.
    Ends with: `\endfirsthead`
  - the *head*
    This is the header which appears at the top of the table for every page after the first. If the *first head* is not defined, then the *head* will also appear at the beginning of the table.
    Ends with: `\endhead`
  - the *foot*
    This is the footer that appears at the bottom of the table for every page after the first. If the *last foot* is not defined, then the *foot* will also appear at the end of the table.
    Ends with: `\endfoot`
  - the *last foot*
    This is the footer that appears at the end of the table.
    Ends with: `\endlastfoot`

- the *content*
  The table's *content* of the table is found after the *preamble*

In terms of usage...

- *first head*
  Can be used to show the column names
- *head*
  Can be used to indicate *"...continued from previous page."*
  Can also be used to show the column names (on top of every page)
- *foot*
  Can be used to indicate *"Continued on next page..."*
- *last foot*
  Closes the table (e.g. print a `\hline` if the table has borders)

The code below shows how to define a `longtable`:

```
\begin{longtable}{|l|l|l|}
  \caption{A long table.}
  \label{table:longtable-example} \\

  %%% first head (at the beginning of the table)
  \hline
  \multicolumn{1}{|c|}{Column 1} &
  \multicolumn{1}{c|}{Column 2} &
  \multicolumn{1}{c|}{Column 3}
  \\
  \hline
  \endfirsthead

  %%% head (repeated on every page)
  \hline
  \multicolumn{3}{|l|}{\dots continued from previous page.}
  \\
  \multicolumn{1}{|c}{Column 1} &
  \multicolumn{1}{c}{Column 2} &
  \multicolumn{1}{c|}{Column 3}
  \\
  \hline
  \endhead

  %%% foot (repeated on every page)
```

```
  \hline
  \multicolumn{3}{|r|}{Continued on next page\dots} \\
  \hline
  \endfoot

  %%% last foot (at the end of the table)
  \hline
  \endlastfoot

  %%% content...

  Data 1 & Data 2 & Data 3 \\
  Data 4 & Data 5 & Data 6 \\
  ...

\end{longtable}
```

36 Other table packages

There are other packages allowing to create tables, each with different options:

nicematrix	https://www.ctan.org/pkg/nicematrix
array	https://www.ctan.org/pkg/array
tabularx	https://www.ctan.org/pkg/tabularx
...	...

37 `table`

The `table` environment is used to reference a table (section 17), add a caption, and add the table to the list of tables (section 38).

Here is an example:

```
\begin{table}[h]           % we want to anchor the table "here" ([h])
  \caption{Tic-Tac-Toe again} % the table caption
  \label{table:some-table}    % used for cross-references
  \centering                  % used to center the table on the page

  \begin{tabular}{c | c | c}
    X & O & X \\
    \hline
    O & X & X \\
    \hline
    X & O & O \\
  \end{tabular}
\end{table}
```

And here is the result:

Table 9: Tic-Tac-Toe again

X	O	X
O	X	X
X	O	O

38 List of tables

We can easily add a list of tables (preferably at the end of the document) with the `\listoftables` command:

```
\listoftables
```

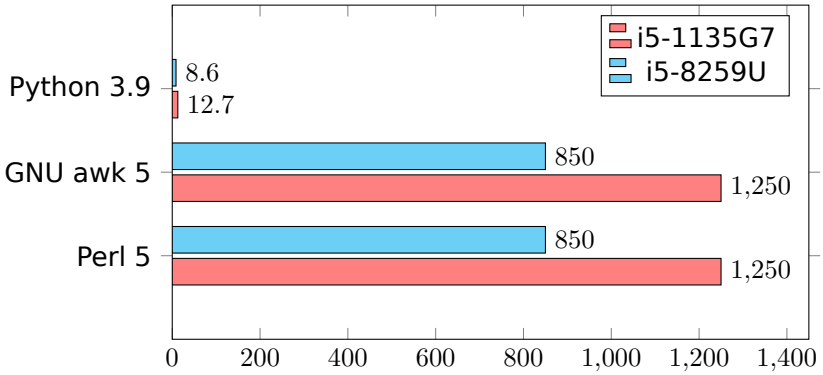
Part XIV
Plots, Images, and Figures

39 Plots

Plots are a very important component in LaTeX, as they allow to draw all sorts of graphs.

Here is an example of chart that I used in another article:

Figure 7: An example of a bar chart



To draw plots, you will need the `pgfplots` package:

```
%%% In the preamble
\usepackage{pgfplots}
```

We will not teach you how to draw plots, because the subject is extremely vast.

40 Add an image

We can add images to our LaTeX files thanks to the `\includegraphics` command from the `graphics` package:

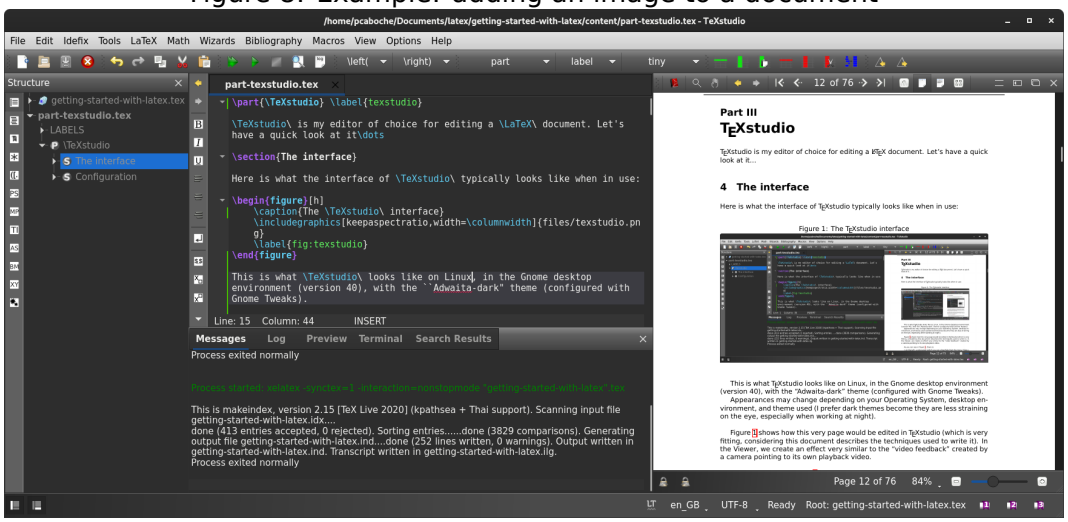
```
%%% In the preamble
\usepackage{graphics}
```

Example:

```
\includegraphics{path/to/image-file}
```

To learn more about `\includegraphics`:
`https://latexref.xyz/_005cincludegraphics.html`

Figure 8: Example: adding an image to a document



40.1 Resize an image to fit the page

To resize an image to fit the page width, in `\includegraphics` we set the parameters `keepaspectratio` and `width`:

`keepaspectratio` Will make the graphic as large as possible, without distortion, within the constraints of `width`, `height`. `totalheight`.

`width` We set the `width` to either `\linewidth` or `\columnwidth`, and let `keepaspectratio` resize the image for us.

`\linewidth`: the width of the current line, decreased for each nested list (but because we are not in a list, then it is equal to `\textwidth`, the horizontal width of the page body).

`\columnwidth`: in a two-column document, the width of a column (in a two-column document, it is equal to `\textwidth`).

Example:

```
\includegraphics[keepaspectratio,width=\columnwidth]{path/to/image-file}
```

41 Figures

Figures are used for easy reference of plots, images, and other visual elements. Figures can be listed in a list of figures, usually at the end of a document.

We will study the code associated with figure 7.

Here is the code in question:

```
\begin{figure}[h]
  \caption{An example of a bar chart}
  \centering
  \medskip

  \pgfplotsset{compat=newest}
  \begin{tikzpicture}
    \begin{axis}
      [
      xbar,
      xmin=0,
      width=10cm,
      height=6cm,
      enlarge y limits=0.5,
      xlabel={Throughput (in MB/s). Higher is better.},
      xmax=1450,
      %ylabel={Language},
      symbolic y coords={perl,gawk,python},
      yticklabels={Python 3.9,GNU awk 5,Perl 5},
      ytick=data,
      nodes near coords,
      nodes near coords align={horizontal},
      legend pos=north east
      ]
      \addplot[fill=red!50] coordinates {(12.7,python) (1250,gawk) (1250,
          ↪ perl)};
      \addplot[fill=cyan!50] coordinates {(8.6,python) (850,gawk) (850,
          ↪ perl)};
      \legend{i5-1135G7,i5-8259U};
    \end{axis}
  \end{tikzpicture}
  \label{figure:throughput-comparison}

\end{figure}
```

The content of a figure is contained in a `figure` environment. The `[h]` indicates that we want to anchor the figure “here” (at the location where it has been declared in the document).

Then we define the `\caption` of the figure (which will also appear in the list of figures, if one is defined in the document).
Then the command `\centering` indicates that the figure is to be centered in the page.

Then we insert the plot itself (which I am not going to describe, that would take too long).
Finally, after the plot, we insert a label. Notice that the ID for the figure starts with “`figure:`”. This is a convention which allows to easily identify the figures.

42 List of figures

We can easily add a list of figures (preferably at the end of the document) with the `\listoffigures` command:

```
\listoffigures
```

Part XV
Extras

In this part, we list some techniques, as well as some very useful packages (which I actually used in this document) that didn't get to have their own chapter...

43 Other useful packages

43.1 Adding web links with `hyperref`

The `hyperref` package defines the `\url` command, which allows to add links to web pages.

Here is how to import the package:

```
%%% In the preamble
\usepackage{hyperref}
```

The `\url` command is very straightforward:

```
\url{www.example.com}
```

Result:
www.example.com

43.2 Ordinal numbers with `nth`

The `nth` package is very useful to print ordinal numbers in English (1^{st}, 2^{nd}, 3^{rd}, 4^{th}...)

Here is how to import the package, with the ordinal mark set to be *superscripted*:

```
%%% In the preamble
\usepackage[super]{nth}
```

Here are some examples:

```
\nth{1}, \nth{2}, \nth{3}, \nth{4}, \nth{21}, \nth{22}, \nth{23},
\nth{101}, \nth{1001}, \nth{1002}, \nth{1000001}
```

And here is the result:
1^{st}, 2^{nd}, 3^{rd}, 4^{th}, 21^{st}, 22^{nd}, 23^{rd}, 101^{st}, 1001^{st}, 1002^{nd}, 1000001^{st}

Without `\nth`, you would have to use the `\textsuperscript` command.

With languages other than English, you would have to use the `\textsuperscript` command (when applicable). For example in French:
1er – premier (masculine), 1ère – première (feminine), 2ème – deuxième, 3ème – troisième…

```
1\textsuperscript{er} -- premier (masculine),
1\textsuperscript{ère} -- première (feminine),
2\textsuperscript{ème} -- deuxième,
3\textsuperscript{ème} -- troisième\dots
```

As you can see, this is more complicated in French.

Not only do you have to cater for masculine/feminine/plural, but there are other rules. For example, “2nd” normally translates to « deuxième » if there are more than two elements, but translates to « second(e) » if there are more than two elements.

This goes to show that the `nth` package may not have an equivalent in other languages.

44 Conclusion

This article summarises what I have learned about LaTeX over the past few months, while working on some articles.

It all started with a few notes, some tips and tricks about LaTeX that I wanted to remember, and then grew into something much bigger: a guide for people who have never used LaTeX before to be able to write documents or work on their thesis.

Interestingly, the writing of this document closely followed the steps described in it… It starts with opening a template (either a relatively blank template provided by a TeX editor like TeXstudio, or a standard template that is adopted by the members of the same school, university, organisation…) and saving this template in a dedicated folder, where you will store all the files needed for generating the document.

Then you start writing down the subjects you want to talk about, and organising them in parts, sections, sub-sections… So you add a Table-of-Contents to have an overview of the overall structure of your documents.

Now that you have some basic plan in place, you start filling out the different parts with ideas. Sometimes the idea is clear enough and a new sentence appears in your document, other times the idea is a bit fuzzy, so you write it down in the form of a comment.

Comments are an extremely valuable tool in LaTeX. They really help in the writing process!

Need to write down an idea that you might want to develop later? Write down as a comment! *Found an interesting article that you need to explore later?* Write down the reference as a comment! *Not completely satisfied with the phrasing of a certain paragraph?* Comment it out, keep several versions in comments, compare those versions, and eventually you will find the right way to express your idea. *Need to write a note to the people you're collaborating with on the document?* Write a comment!

Comments are fantastic in LaTeX, and you can put them anywhere. Other text editors only allow you to add comments in a small box in the margin; this is very limiting. It doesn't come close to the power of LaTeX comments and Ctrl+T! (toggle comments on or off, on the selected lines in TeXstudio)

Now that ideas start flowing, and the document starts growing, you organise the content in multiple files of manageable size, which you store in separate folders depending on the file's purpose.

And now you start to spot recurring terms, or words that need to appear in the same, consistent format. So you start writing custom *macros* for that, and you also consider adding an index to your document.

Then it's not only recurring terms, but every repeating patterns that you turn into macros, which you can call at will. And it makes your life easier.

Now that you have a satisfying amount of content, well organised, with lots of cross-references, you want to experiment with the style of the document (unless someone provided you with a template that you need to follow, in that case your document already meets a certain standard of beauty). So you start modifying things like fonts, margin sizes, and probably add a header and footer to the document.

In any case, it is better to focus on the content of the document first, and modify the style much later. If you want to change the style too early, and in the absence of content, you will have to use some placeholder text (e.g. *Lorem Ipsum*), images, etc. which do not reflect the true content of your document.

So you modify your template much later, when you have at least a dozen pages ready.

And this is how this document came to be!

I told you in the abstract that this document was describing how it was written. Although I was initially talking about the different techniques used to generate a document in LaTeX, in return this also applies to the writing process itself.

In any case, I hope you found this article useful, that it helped you on your journey with LaTeX, or maybe convinced you to try out LaTeX.

References

CTAN (2022). The Comprehensive TeX Archive Network. https://www.ctan.org.

Kernighan B., R. D. (1988). *C Programming Language*. Prentice Hall, 2nd edition. December 1988, *ISBN-13: 978-0131103627*.

MiKTeX (2022). The MiKTEX project. https://miktex.org.

Overleaf (2022a). Biblatex bibliography styles. https://www.overleaf.com/learn/latex/Biblatex_bibliography_styles.

Overleaf (2022b). The TEX family tree: LATEX, pdfTEX, XƎTEX, LuaTEX and ConTEXt. https://www.overleaf.com/learn/latex/Articles/The_TeX_family_tree:_LaTeX,_pdfTeX,_XeTeX,_LuaTeX_and_ConTeXt.

Overleaf (2022c). Typesetting quotations. https://www.overleaf.com/learn/latex/Typesetting_quotations.

Ritchie, K. . (1988). *C Programming Language*. Prentice Hall, 2nd edition. December 1988, *ISBN-13: 978-0131103627*.

TeXstudio (2022). TEXstudio. https://www.texstudio.org.

TeXworks (2022). TeXworks. *TeX Users Group*. http://www.tug.org/texworks/.

tom (2021). 10 ways to customize toc/lof/lot. https://texblog.org/2011/09/09/10-ways-to-customize-tocloflot/.

TUG (2021). Just what is TEX? *TeX Users Group*. http://www.tug.org/whatis.html.

Wikipedia (2022a). LaTeX. https://en.wikipedia.org/wiki/LaTeX.

Wikipedia (2022b). TeX. https://en.wikipedia.org/wiki/TeX.

List of Tables

List of Figures

Index

