LaTeX Typesetting

Björn Lindenberg et. al., delivered by Mattias Davidsson

November 8, 2023

Contents

Introduction

Preface

- This lecture will hopefully be as much of a workshop as a lecture
- Learning by doing
- Teacher and Learner
- Should be up and going with the final paper template at the end of lecture

Introduction

LaTeX is a programming language with the following properties:

- It is a document preparation system, and serves as a tool for typesetting journal articles, technical reports, books, and slide presentations.
- It is most suitable for scientific writing, with advanced typesetting of mathematics.
- It is free and available for most common operating systems.

Note that LaTeX is different from word processors, what you see is NOT what you get. Instead it encourages high-quality content in your document, since the focus is not on cosmetics.

Why use LaTeX ?

- LaTeX "enforces" proper and better typesetting, especially for inexperienced writers.
- Provides multi-lingual typesetting.
- Automatic generation of bibliographies and indexes.
- Makes it easy to handle large structured documents (*chapters, sections, subsections, index, appendix, table of contents, etc.*)

If you don't want to think about formatting and want the software to make things look good for you, use LaTeX.

Software

Software

A small selection of distributions:

- TeX Live (Linux, MacOSX, Unix and Windows)
- MiKTeX (Windows)
- MacTeX (MacOSX)
- TeXnicCenter
- TeXmaker or TeXstudio
- Overleaf (Online)

Now - let's get startd with Overleaf

- Let's spend (a maximum of) 5 minutes to get you an account
- In another five minutes you should have created your first LaTeXdocument
- As you can see this presentation was made using LaTeX
- And here is where we are heading...
- Links to presentation and to overleaf in chat

Literature

Literature

Books

- (Swedish) Per Jacobsson, *Introduktion till LaTeX*, Studentlitteratur, 2004.
- Leslie Lamport, *LaTeX. A Document Preparation System*, Addision-Wesley, 1994.
- Helmut Kopka and Patrick W. Daly, *Guide to LaTeX 2ε*, Pearson, 2003.
- Frank Mittlebach et al, *The LaTeX Companion*, Addison-Wesley, 2004.
- Michel Goossens et al, *The LaTeX Graphics Companion*, Addison-Wesley, 1997.
- George Gratzer, *Math Into LaTeX*, Birkhauser, 2000.

Internet

- TEX Users Group
- LATEX on wikibooks
- TEX Frequently Asked Questions
- TEX.Stack Exchange
- Ti*k*Z web site.

Manuals

- The Not So Short Introduction to LATEX 2_ε
- LATEX 2_ε for authors
- The Comprehensive LATEX Symbol List
- Using Imported Graphics in LATEX 2_ε.

Compiling a Classic Example

Compiling a Classic Example

We can save code

```
\documentclass{article}
\begin{document}
Hello world!
\end{document}
```

in a standard text file named `hello.tex` and compile it. Lets try it!

- The typeset version of `hello.tex` is written to `hello.dvi`.← dep on compiler
- Data for creating cross references is stored in `hello.aux` and in `hello.log` the compilation text is stored.

Compiling a Classic Example

Hello world!

Classes and Packages

Classes and Packages

Structure of a LaTeX file:

```
\documentclass[options]{class}
preamble
\begin{document}
body
\end{document}
```

- The *class* specifies the type of document. Standard are `article`, `book`, `report` or `letter`. But there also many good non-standard classes such as `amsart` (AMS) or `scrartcl` (Koma-Script).
- We can specify *options* to choose for instance paper format and font size. But [*options*] can also be left out, e.g., `\documentclass{article}`.

Some class options for the standard classes

`10pt`, `11pt`, `12pt`	font size of body text
`letterpaper`, `legalpaper`, `executivepaper`, `a4paper`, `a5paper`, `b5paper`	paper format
`landscape`	landscape paper orientation
`onecolumn`, `twocolumn`	number of columns
`oneside`, `twoside`	single or double paged

Example: A double-paged book in A5 format, with two columns with body text at 11 points font size:

```
\documentclass[twoside,a5paper,twocolumn,11pt]{book}
```

In the *preamble* you load *packages*, define your own commands and set global options to get extra functionality.

You use the command `\usepackage` to load packages.

Example:

```
\usepackage[utf8]{inputenc}
\usepackage{amsmath,amssymb}
\usepackage{enumerate}
\usepackage{graphicx}
```

Your text body is then placed within the `document` environment:

```
\begin{document}
   Main body of text...
\end{document}
```

Document Structure

Document Structure

Title and author:

```
\title{Elements}
\author{Euclid\thanks{Euclid@alexandria.antiquity}}
\date{300 BC}
...
\maketitle % Inside document environment
```

- The argument to \thanks is shown as a footnote.
- You may leave \date out, and today's date will be used.
- In case of multiple authors \and should be used, e.g., \author{Carl Michael Bellman \and Johann Sebastian Bach}.

Elements

Euclid*

300 BC

* Euclid@alexandria.antiquity

1

Music in LaTeX

Carl Michael Bellman
Sweden

Johann Sebastian Bach
Germany

November 8, 2023

Abstract:

```
\begin{abstract}
  A short text describing the contents of the document.
\end{abstract}
```

Headings:

```
\part{heading}
\chapter{heading}        \section{heading}
\subsection{heading}     \subsubsection{heading}
\paragraph{heading}      \subparagraph{heading}
```

With an asterisk the heading is not numbered, e.g., `\section*{heading}`.

Table of contents:

```
\tableofcontents
```

Other listings: `\listoffigures` and `\listoftables`.

Example: An article heading structure:

```
\section{Divisibility}
\subsection{The Division Algorithm}
\subsection{Prime numbers}
\section{Congruence}
\subsection{The Chinese Remainder Theorem}
```

Result:

1 Divisibility

1.1 The Division Algorithm

1.2 Prime numbers

2 Congruence

2.1 The Chinese Remainder Theorem

Syntax

So now it is time to start writing some text in the document environment.

Note that:

- A return character in the code yields a space character by LaTeX.
- In fact, only rarely use your own line breaks in written text. Let LaTeX do it for you! With this in mind, page breaks are also handled automatically.
- One or more empty lines in the code is interpreted as a new paragraph. You can also use `\par`.

Syntax

Special signs

You control LaTeX using commands beginning with the special sign backslash (\) followed by one or more letters.

Example: `\LaTeX` → LaTeX .

Note that LaTeX is case sensitive in command names, e.g., `\LaTeX` and `\latex` are two separate commands.

Several space characters following each other is interpreted as one and if it comes after a command, the command "eats" the space.

Example: `a  b   c` → a b c

Example: `\LaTeX is fun!` → LaTeXis fun!

You can use `\␣` to insert a space character after a command.

Example: `\LaTeX\ is fun!` → LaTeX is fun!

Curly brackets (`{}`) are used for grouping, e.g., around command arguments.

Example: `\textbf{bold text}` → **bold text**

Groups can be used to make local changes.

Example: `abc {\tiny abc} abc` → abc abc abc

White space after command arguments does not disappear.

Example: `\textit{italic} style` → *italic* style

Special characters and escape commands

	Syntax	Escape
\	space if followed by a blank	\textbackslash
{	start group	\{
}	close group	\}
%	comment	\%
~	space with word bonding	\textasciitilde
$	math mode	\$
_	subscript	_
^	supscript	\textasciicircum
&	alignment	\&
#	command parameter	\#

Environments

For more complicated objects, such as lists and tables, environments are used:

```
\begin{name}
  Text or commands.
\end{name}
```

Here `name` specifies the used environment, e.g., `center`.

Nested environments are possible, make sure that they are closed in the correct order. The following is **not** correct:

```
\begin{environment1}
 \begin{environment2}
  This won't work!
  \end{environment1}
\end{environment2}
```

Typefaces

Typefaces

Namn	Kommando	Example
Upright	`\textup{`*text*`}`	The quick brown fox
Italic	`\textit{`*text*`}`	*The quick brown fox*
Slanted	`\textsl{`*text*`}`	*The quick brown fox*
Bold	`\textbf{`*text*`}`	**The quick brown fox**
Sanserif	`\textsf{`*text*`}`	The quick brown fox
Small Caps	`\textsc{`*text*`}`	The quick brown fox
Typewriter	`\texttt{`*text*`}`	`The quick brown fox`

These commands can be combined:

Example: `\textit{\textsf{Abcdef}}` → *Abcdef*

The command `\emph{`*text*`}` typesets *text* in italic, if the surrounding environment is not already set to italic, then *text* is set upright.

Example: `\textit{aa \emph{bb} cc}` → *aa* bb *cc*

Sizes

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

These commands do not take arguments; they are used in groups.

Example: `{\Large 1 {\tiny 2} 3\small 4} 5` $\rightarrow$ 1 ² 3⁴ 5

Lists

Lists

We can make lists numbered or marked with bullet points.

Enumeration Example:

(i) First statement.

(ii) Second statement.

(iii) The third statement.

The list above was typeset using the code:

```
\begin{enumerate}[(i)]
    \item First statement.
    \item Second statement.
    \item The third statement.
\end{enumerate}
```

Bullet points are achieved with the `itemize` environment.

An item in a list may contain another list up to four levels.

Example:

```
\begin{itemize}
  \item First main point
    \begin{itemize}
      \item Sub point 1.1
      \item Sub point 1.2
    \end{itemize}
  \item Second main point
\end{itemize}
```

- First main point
  - Sub point 1.1
  - Sub point 1.2
- Second main point

We may also change the marker of bullet points in lists by `\item[`$text$`]`.

Example: Here is an example using the `description` environment.

```
\begin{description}
  \item[InDesign] Layout program.
  \item[\LaTeX] Typesetting program.
  \item[Word] Word processor.
\end{description}
```

Result:

InDesign Layout program.

LaTeX Typesetting program.

Word Word processor.

Bits and Pieces

Bits and Pieces

Footnotes are inserted using the command

`\footnote{`*text*`}`

where the reference sign should be. The *text* appears at the bottom of the page, above the foot margin.

Margin texts are inserted using the command

`\marginpar{`*text*`}`

where *text* is placed in the left or right margin depending on if we write a single page or double paged document.

Long documents may be split over several smaller files. To insert a *file* you use one of the following commands:

`\input{`*file*`}`
`\include{`*file*`}`

Page breaks: Normally we let LaTeX handle all page breaks. However, sometimes we want explicit page breaks, e.g., before the appendices. For this we may use one of the following commands:

```
\clearpage
\cleardoublepage
\newpage
```

The two first commands place all unplaced figures before the page break (more on this later).

Horizontal spaces are generated using the commands such as

```
\hspace{length}
\
\,
\quad
\qquad
```

Bits and Pieces

The possibilities to change the appearance of the document is almost unlimited. Often it involves changing the values of parameters or using specialized packages for some particular functionality.

If we want to make any change global for the whole document we need to place the setting in the preamble.

A few examples:

- To change the **indentation** of a new paragraph, we write

  ```
  \setlength{\parindent}{3em}
  ```

  Here each indent will be a space as wide as three em-lines in the current typeface and size (an em-line is retrieved from `---`).

- The **numbering** of sections is controlled by the counter `secnumdepth`. If we set

  ```
  \setcounter{secnumdepth}{1}
  ```

  a heading inserted using `\section` is numbered, but not the headings on lower levels, such as `\subsection`, etc.
- Using the counter `tocdepth` we may control which heading levels that should be included in the **table of contents**.

  ```
  \setcounter{tocdepth}{3}
  ```

If we want to preserve **parent list information** of markers in nested enumerated lists, then we can use settings such as these:

```
% 1st level: (I), (II), ...
\renewcommand{\theenumi}{(\Roman{enumi})}
% 2nd level: a, b, c, ...
\renewcommand{\theenumii}{\alph{enumii}}
% 3rd level: 1, 2, 3, ...
\renewcommand{\theenumiii}{\arabic{enumiii}}
```

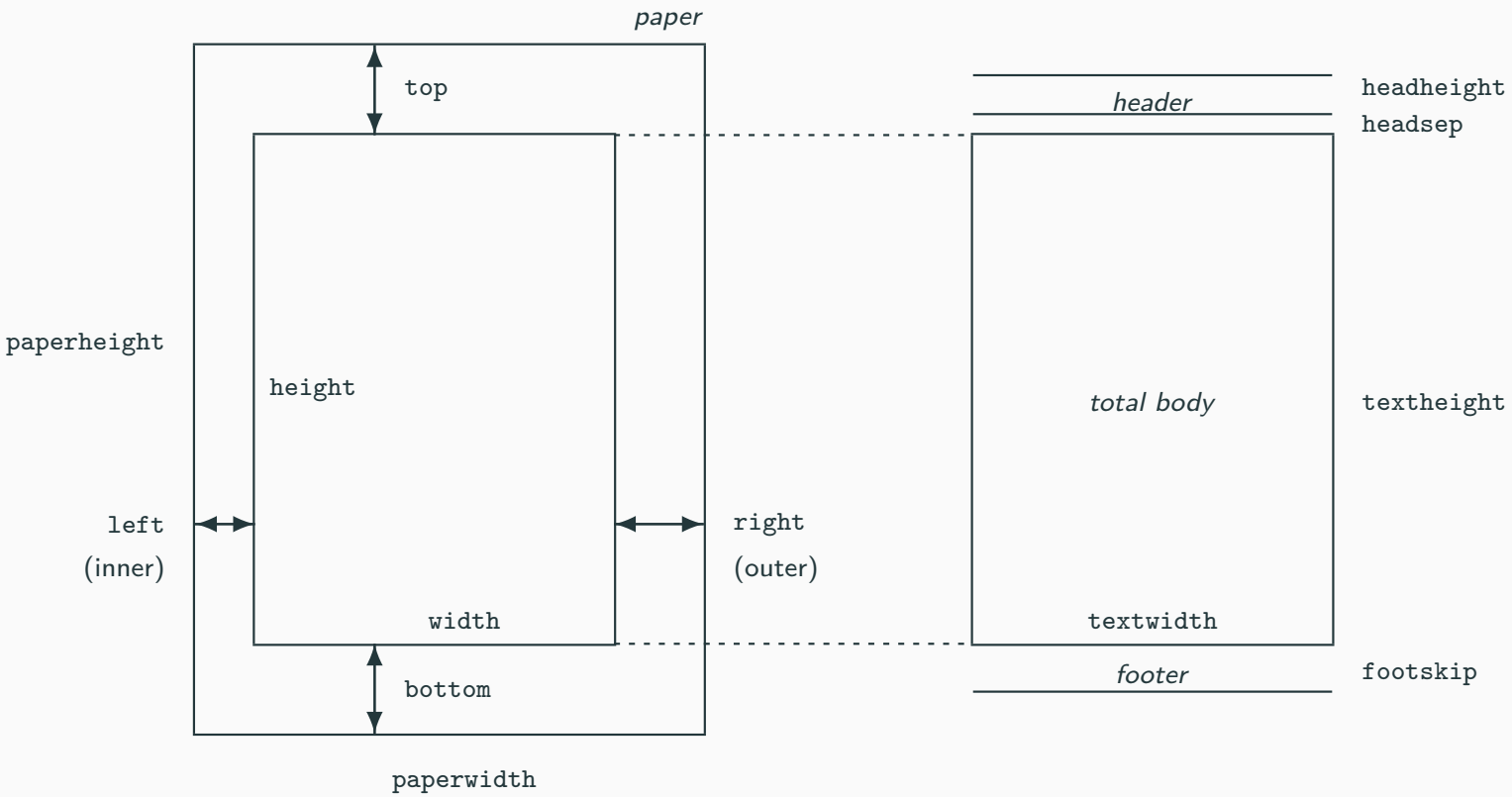
Output with nested `enumerate` environments:

(I). ← (Roman)
    (I).a ← (Roman).alph
        (I).a.1 ← (Roman).alph.arabic
        (I).a.2 ...
    (I).b ...
(II). ...

The margins of the document format is easiest defined using the `geometry` package.

Example:

```
\usepackage[a3paper,twoside]{geometry}
\geometry{left=5cm,right=4cm,top=5cm,bottom=6cm}
```



Mathematics

Mathematics

$\mathcal{AMS}$LaTeX packages is recommended when typesetting mathematics:

```
\usepackage{amsmath,amssymb,amsthm}
```

These and the full documentation can be found here: $\mathcal{AMS}$LaTeX.

- Mathematical statements may only be inserted in a certain state of the code called **math mode**.
- So most of the commands presented in this section will only work in math mode, and some will require $\mathcal{AMS}$LaTeX.

To enter math mode we use two techniques:

- **Inlined:** Directly between two $ characters among written text, e.g.,

  ```
  Einstein found the identity $e = mc^2$.
  ```

  yields the output

  Einstein found the identity $e = mc^2$.

- **Environments:** Or in a mathematics environment, such as `equation` or `aligned`, e.g.,

```
Therefore
\begin{equation*}
  e = mc^2
\begin{equation*}
as required.
```

Therefore

$$e = mc^2$$

as required.

The **underscore** (_) and **circumflex** (^) generates subscripts and superscripts respectively. **Remember to group the scripts if they consist of more than one character!**

Example: `$f_n$` $\rightarrow$ f_n

Example: `$x^{10}$` $\rightarrow$ x^{10}

Forgetting the group when writing e^{ni}:
Example: `$e^ni$` $\rightarrow$ e^ni

You can combine subscript with superscript:
Example: `$k_{n^2}^{a_j}$` $\rightarrow$ $k_{n^2}^{a_j}$

We can also use them with ordinary text:
Example: `list$_n$` $\rightarrow$ list$_n$

Stand-alone formulas are typeset using environments. We use an **asterisk** (*) if we want the equation/equations unnumbered.

`equation/equation*`	Without line break.
`gather/gather*`	Multiple rows are centered.
`align/align*`	Multiple rows, justified by &.

- Note that `\[ ... \]` is short for the environment `equation*`.
- There are more environments, but these are the most common.

- By default expressions of inlined math will differ compared to environments. LaTeX formats the symbols for better alignment to the surrounding text.
- If this is not to your liking on certain expressions, you can use `\displaystyle` to revert this. Although there will be disruptions of vertical alignments between rows.

Inlined math in standard format: $\sum_{j=0}^{\infty} \alpha_j e^{-\omega_j (x - x_0)}$

Inlined math with `\displaystyle`: $\displaystyle\sum_{j=0}^{\infty} \alpha_j e^{-\omega_j (x - x_0)}$

Example: The code

```
\begin{equation}
  \int_{-\infty}^\infty f(x + t) e^{\pi i t} \ dt
\end{equation}
```

generates

$$\int_{-\infty}^{\infty} f(x+t)e^{\pi it}\ dt \tag{1}$$

If we use `equation*` then the equation numbering will disappear. Good practice is to **only number expressions that you will reference** in written text.

For multiple statements we can use `gather`. Line breaks are done by `\\`.

Example: The code

```
\begin{gather}
  |x + y| \leq |x| + |y|, \\
  f_{n + 2} = f_{n + 1} + f_n, \\
  a a^{-1} = e.
\end{gather}
```

will output

$$|x + y| \leq |x| + |y|, \tag{2}$$

$$f_{n+2} = f_{n+1} + f_n, \tag{3}$$

$$aa^{-1} = e. \tag{4}$$

Again, if we do not wish to have any numbering, we use `gather*` instead. If you want to remove the numbering of a specific expression, then use `\nonumber` on the corresponding line.

With `align` we can justify each row along a vertical line.

Example: The code

```
\begin{align*}
  |x + y|   & \leq |x| + |y|,    \\
  f_{n + 2} & = f_{n + 1} + f_n, \\
  a a^{-1}  & = e.
\end{align*}
```

leads to

$$\begin{aligned} |x + y| &\leq |x| + |y|, \\ f_{n+2} &= f_{n+1} + f_n, \\ aa^{-1} &= e. \end{aligned}$$

This environment is suitable when you want to show multiple steps of a computation or present expressions that are too long for the margins.

Math Mode Commands

Rational expressions are typeset using the command

`\frac{`*numerator*`}{`*denominator*`}`

where *numerator* and *denominator* may be arbitrary expressions. To get a **binomial coefficient** you exchange `\frac` for `\binom`.

Example: In math mode the code

```
\binom{n}{k} = \frac{ n! }{ k! ( n - k )! }.
```

yields

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Note that space characters vanishes in math mode, and may thus be used however you want to enhance **code readability** of complicated mathematical expressions.

Root expressions are generated by `\sqrt`.

Example: `\sqrt{2x}` $\rightarrow$ $\sqrt{2x}$
Example: `\sqrt[3]{8}` $\rightarrow$ $\sqrt[3]{8}$

Alternative cases can be typeset using the environment `cases` (within math mode).

Example:

```
|x| = \begin{cases}
  -x & \text{if $x < 0$}, \\
  x  & \text{otherwise}.
\end{cases}
```

where `\text` is used to exit math mode on the following group.
Result:

$$|x| = \begin{cases} -x & \text{if } x < 0, \\ x & \text{otherwise.} \end{cases}$$

We can define **matrices** with `\\` and `&` as row and column separators respectively.

Example: The matrices

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}, \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{Bmatrix} 1 & 2 \\ 3 & 4 \end{Bmatrix}, \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}, \begin{Vmatrix} 1 & 2 \\ 3 & 4 \end{Vmatrix}$$

can be typeset by

```
\begin{<matrixtype>}
  1 & 2 \\
  3 & 4
\end{<matrixtype>}
```

where `<matrixtype>` in order of appearance is `matrix`, `pmatrix`, `bmatrix`, `Bmatrix`, `vmatrix` and `Vmatrix`.

Brackets, i.e., (), {}, [] and so forth, can be enlarged to enclose an expression by using `\left` followed by `\right` at the end of the expression.

Example: Ugly

```
(\frac{1}{n + 1} - k^2)^{1/2}
```

Result:

$$(\frac{1}{n+1} - k^2)^{1/2}$$

Example: Better

```
\left(\frac{1}{n + 1} - k^2\right)^{1/2}
```

Result:

$$\left(\frac{1}{n+1} - k^2\right)^{1/2}$$

Observe that `\left` and `\right` must come in pairs of two. With a period character you insert an **open bracket**, e.g.,

`\left ( <stuff> \right.`

Example: From

```
\left\lceil\frac{1}{x + 1} - \frac{x}{x^2 - 1}\right.
\left\{1 + \frac{1}{p}\right\|
```

we get:

$$\left\lceil \frac{1}{x+1} - \frac{x}{x^2-1} \right. \quad \text{and} \quad \left\{ 1 + \frac{1}{p} \right\|, \quad \text{respectively.}$$

Spaces

a b	`\thinspace`	`\,`	a b	`\negthinspace`	`\!`
a b	`\medspace`	`\:`	a b	`\negmedspace`	
a b	`\thickspace`	`\;`	a b	`\negthickspace`	
a b	`\quad`				
a b	`\qquad`				

Ellipses

$\ldots$	`\ldots`	$\cdots$	`\cdots`	$\ddots$	`\ddots`	$\vdots$	`\vdots`

Typefaces

Abc	`\mathrm{`*text*`}`	Abc	`\mathit{`*text*`}`
$\mathbf{Abc}$	`\mathbf{`*text*`}`	Abc	`\mathsf{`*text*`}`
$\mathtt{Abc}$	`\mathtt{`*text*`}`	$\mathcal{ABC}$	`\mathcal{`*text*`}`
$\mathfrak{Abc}$	`\mathfrak{`*text*`}`	$\mathbb{ABC}$	`\mathbb{`*text*`}`
$\boldsymbol{\alpha\beta\gamma}$	`\boldsymbol{`*text*`}`		

Operators and functions

- The **ams-packages** has commands for most of the basic symbols in mathematics, e.g., `\omega`, `\sin` and `\max`. For a quick look-up list see this wiki.
- Note that operators in mathematics are never in italic, e.g.,

$$\sin(x), \quad \det(\mathbf{A}), \quad \max_{x \in U}\{|f(x)|\}.$$

- Sometimes your particular operator is not standard. Of course you can always force characters to be upright by for example `\text`. But if it is a common operation you can define your own operator. You do this by placing

  `\DeclareMathOperator{`*function name*`}{`*text*`}`

  in the **preamble**.

Example:

```
% In preamble
\DeclareMathOperator{\alg}{ALG}
...
% In body
\begin{equation*}
    \alg_n(\mathbb{F}_p) = \mathbb{Q}(\sqrt[3]{2}).
\end{equation*}
```

Result:

$$\mathrm{ALG}_n(\mathbb{F}_p) = \mathbb{Q}(\sqrt[3]{2}).$$

A few more examples

- **Functions** are typeset using `\colon`.

  Example: `f \colon A \to B` $\rightarrow$ $f \colon A \to B$

- To **negate** a relation put the command `\not` before.

  Example: `A \not\subset B` $\rightarrow$ $A \not\subset B$

- The symbol of **divisibility** is given by `\mid`.

  Example: `a \mid b` $\rightarrow$ $a \mid b$

Theorem Environments

Theorems and such are typeset using environments, they are created by one of the following commands:

\newtheorem{*name*}[*counter*]{*heading*}
\newtheorem{*name*}{*heading*}[*counter*]

Example: With the code:

```
\newtheorem{theorem}{Theorem}[section]
\newtheorem{lemma}[theorem]{Lemma}
```

two environments are created, `theorem` and `lemma`. They are set using the headings "Theorem" and "Lemma", respectively. Both headings are numbered, the first regarding section and the second uses the same counter, e.g., in section 2 we may have the following headings: Theorem 2.1, Lemma 2.2, Lemma 2.3, Theorem 2.4 etc.

- By using

  `\theoremstyle{`*style*`}`

  before `\newtheorem`, you may control the **style** of the theorem you create. There are three predefined styles: `plain`, `definition` and `remark`. You may create new styles using `\newtheoremstyle`.
- We use theorems as **environments**:

```
\begin{<theorem name>}[info]
  text
\end{<theorem name>}
```

  where [*info*] is optional.

Example: The code

```
\begin{theorem}[Fermat's last theorem]
  Let $n$ be a natural number. If $n > 2$, then there are no
  natural number solutions to $x^n + y^n = z^n$.
\end{theorem}
```

results in:

> **Theorem 1.2 (Fermat's last theorem).** *Let n be a natural number. If $n > 2$, then there are no natural number solutions to $x^n + y^n = z^n$.*

Proofs are typeset using the environment `proof`:

```
\begin{proof}[optional heading]
  text
\end{proof}
```

If [*optional heading*] is left out, the predefined heading "Proof" is used. After *text* a Halmos symbol is inserted, i.e., □.

Example: The code

```
\begin{proof}[Complicated Proof]
  Suppose that $x^n + y^n = z^n$ for some ...
\end{proof}
```

give the result:

Complicated Proof.
Suppose that $x^n + y^n = z^n$ for some ... □

Tables

Tables

In-place **tables** are typeset using the environment:

\begin{tabular}{*column types*}
  rows
\end{tabular}

where \\ and & are used as row and column separators similar to matrices or arrays.

The number of columns and their types are defined by *column types*:

`l`	Aligned to the left.
`c`	Centered.
`r`	Aligned to the right.
`p{`*width*`}`	Column with fixed *width*, e.g., `p{3cm}`. Rows exceeding this width are broken automatically, without affecting the other cells on that row.

We may also have:

`|`	Will insert vertical lines between respective columns, e.g., `c|cc|l`.
`||`	Same but with double vertical lines.
`*{`n`}{`col`}`	Quick syntax for column expressions n times, e.g., `*{4}{c|}` is the same as `c|c|c|c|`.
`@{`$text$`}`	Inserts *text* on each row, between the two columns.
`\hline`	Horizontal line above a row. Must start the row.
`\cline{m-n}`	Horizontal line on a row from the mth column to the nth. Must start the row.
`\vline`	Inserts a vertical line on the row and column where it is used.

Example: The table

AAA	BBB	CCC
ddd-ddd	012	x
eee-eee-eee	34567	yyy

was typeset using the code:

```
\begin{tabular}{|lr|c|}
  \hline
  AAA          & BBB   & CCC \\
  \hline
  ddd-ddd      & 012   & x   \\
  eee-eee-eee & 34567 & yyy \\
  \hline
\end{tabular}
```

To let a text span multiple columns we use the command:

`\multicolumn{`n`}{`col`}{`$text$`}`

where n give the number of columns that *text* should stretch over and using *col* you control the types of new columns. Allowed values in *col* is exactly one of `l`, `c` and `r`, together with `|` and `@`.

Example: Suppose we want to write the text "Options" centered in a column that stretches over four columns. Thus we write:

```
\multicolumn{4}{c}{Options}
```

A possible vertical line to the right of the four original columns disappears. If you wish that the line remains, you replace `{c}` with `{c|}`.

Example:

Movies

Title	*Year*	*Country*	Grade				
			1	2	3	4	5
Persona	1966	Sweden					✔
Play Time	1967	France					✔
Solaris	1972	USSR					✔
Sleeper	1973	USA				✔	
Pretty Woman	1990	USA	✔				
Matrix	1999	USA		✔			
The Road Home	1999	China				✔	
Lagaan	2001	India				✔	

At most we have eight columns: `|lcl|ccccc|`.

```
\begin{tabular}{|lcl|ccccc|}
  \hline
  \multicolumn{8}{c}{\textbf{Movies}}\\
  \hline
  \hline
  & & & \multicolumn{5}{c|}{\textsf{Grade}}\\
  \cline{4-8}
  \emph{Title} & \emph{Year} & \emph{Country} & 1 & 2 & 3 & 4 & 5 \\
  \hline
  Persona       & \oldstylenums{1966} & Sweden  & & & & & \ding{52} \\
  Play Time     & \oldstylenums{1967} & France  & & & & & \ding{52} \\
  Solaris       & \oldstylenums{1972} & USSR    & & & & & \ding{52} \\
  Sleeper       & \oldstylenums{1973} & USA     & & & & \ding{52} & \\
  Pretty Woman  & \oldstylenums{1990} & USA     & \ding{52} & & & & \\
  Matrix        & \oldstylenums{1999} & USA     & & \ding{52} & & & \\
  The Road Home & \oldstylenums{1999} & China   & & & & \ding{52} & \\
  Lagaan        & \oldstylenums{2001} & India   & & & & \ding{52} & \\
  \hline
\end{tabular}
```

Regarding the example:

- The command `\ding` requires the `pifont` package.
- To get lower case numbers the command `\oldstylenums` is used.

  Example: `\oldstylenums{1476}` $\rightarrow$ 1476

A few packages and environments suitable for tables:

`array`	More column types.
`supertabular`	Table spanning multiple pages.
`longtable`	Table spanning multiple pages.
`colortbl`	Columns and row with different background colors.
`booktabs`	More professional tables.

Graphics

Graphics

- To be able to insert **images** you have to include the following package:

  `\usepackage{graphicx}`

- Images are then inserted in-place using the command:

  `\includegraphics[`*parameters*`]{`*image file*`}`

- Note that if the *image file* is in a subdirectory, then you need to add `\graphicspath{{`*image directory*`/}}` to the preamble.

With [*parameters*] you can **manipulate** the inserted image. You write *parameters* like a list of elements on the form: *keyword*=⟨*value*⟩.

Some common keywords:

`height`	Forces the height of the image to be ⟨*value*⟩.
`width`	Forces the width of the image to be ⟨*value*⟩.
`scale`	Scales the image using the factor ⟨*value*⟩.
`angle`	Rotates the image ⟨*value*⟩ degrees counter clockwise.
`origin`	The center for the rotation, e.g., ⟨*value*⟩ may be `c` or `rt`.
`viewport`	Defines a window in the image. Here ⟨*value*⟩ must be on the form $x_1\ y_1\ x_2\ y_2$, where (x_1, y_1) and (x_2, y_2) is the lower left and right corner, respectively.
`clip`	Clips everything outside the window.

For more information see *Using Imported Graphics in LaTeX 2ε*.

Example: With

```
\includegraphics[width=3.5cm]{typo.eps}
\includegraphics[width=2cm,height=2.75cm]{typo.eps}
```

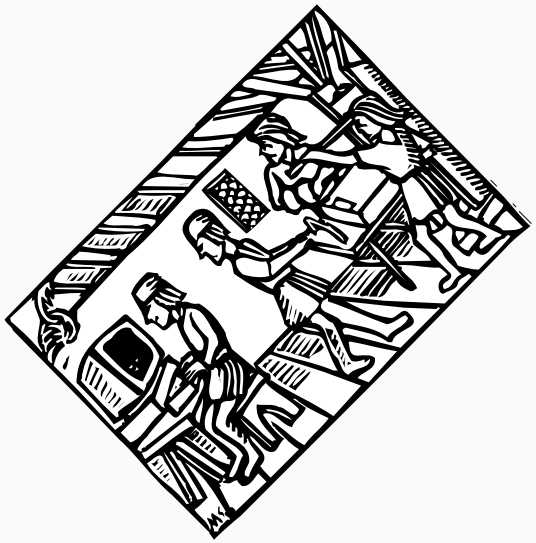
we get graphics such as

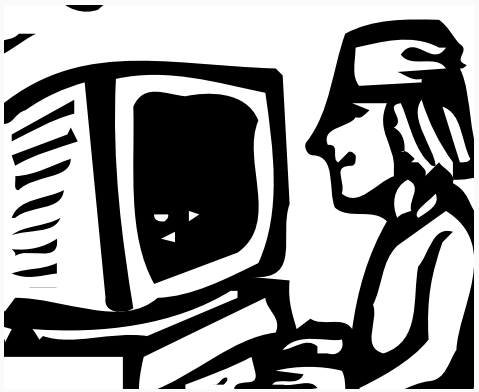
Assuming that we have an image file named `typo.eps`.

Example: From

```
\includegraphics[width=3cm,angle=45]{typo}
\includegraphics[viewport=30 145 165 255,
                 clip,width=3cm]{typo}
```

we have the following examples





Colors, TikZ & PGF

Colors

- For colors in LaTeX we can use the `xcolor` package.
- Predefined colors are: red, green, blue, cyan, magenta, yellow, black, gray, darkgray, lightgray, brown, lime, olive, orange, pink, purple, teal, violet, and white.

Example: `{\color{blue} blue text}` $\rightarrow$ blue text

You define your own colors using the commands

```
\definecolor{name}{gray}{n}
\definecolor{name}{rgb}{r, g, b}
```

where $0 \leq n, r, g, b \leq 1$.

Example: Let us define a golden red color:

```
\definecolor{goldred}{rgb}{0.8, 0.61, 0.11}
```

and use it:

```
{\color{goldred} golden red text}
```

which gives the result: gold red text.

TikZ & PGF

Using the `tikz` package you may create images directly in LaTeX. Quick look-up and examples in this wiki link. A comprehensive documentation through this link: pgfmanual.pdf.

One way to use tikz is by the following environment:

```
begin{tikzpicture}[scale=2]
   commands for drawing
end{tikzpicture}
```

This creates a drawing area at the current position in the text and scales the content by 2. The scale argument is optional. You can also scale in one dimension or different scaling for two dimensions:

```
\begin{tikzpicture}[xscale=2, yscale=5]
   commands for drawing fancy stuff
\end{tikzpicture}
```

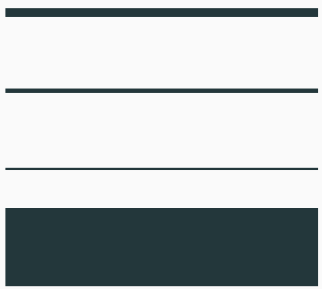
The default drawing unit is 1 cm.

Line thickness, line type, color etc. are controlled using different parameters. Most drawing commands take multiple arguments in square brackets divided by a comma.

For instance, changing the thickness of lines:

```
\begin{tikzpicture}
  \draw [ultra thick] (0,1.5) -- (2,1.5);
  \draw [thick] (0,1) -- (2,1);
  \draw [thin] (0,0.5) -- (2,0.5);
  \draw [line width=0.5cm] (0,0) -- (2,0);
\end{tikzpicture}
```

Result:



You can also change the line style:

```
\begin{tikzpicture}
 \draw [dashed, ultra thick] (0,1) -- (2,1);
 \draw [dashed, red] (0, 0.5) -- (2,0.5);
 \draw [dotted] (0,0) -- (2,0);
\end{tikzpicture}
```

Result:

You can put decorations on lines. For instance you can draw arrows or bars at the start and end of a line:

```
\begin{tikzpicture}
  \draw [->] (0,0) -- (2,0);
  \draw [<-] (0, -0.5) -- (2,-0.5);
  \draw [|->>] (0,-1) -- (2,-1);
\end{tikzpicture}
```

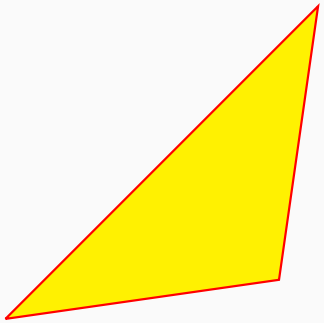
Result:

If you do not specify the arrow type - is used. You may combine the different arrow types (see the third line).

To draw a polygon with corners in the points (x_1, y_1), ..., (x_n, y_n):

Example:

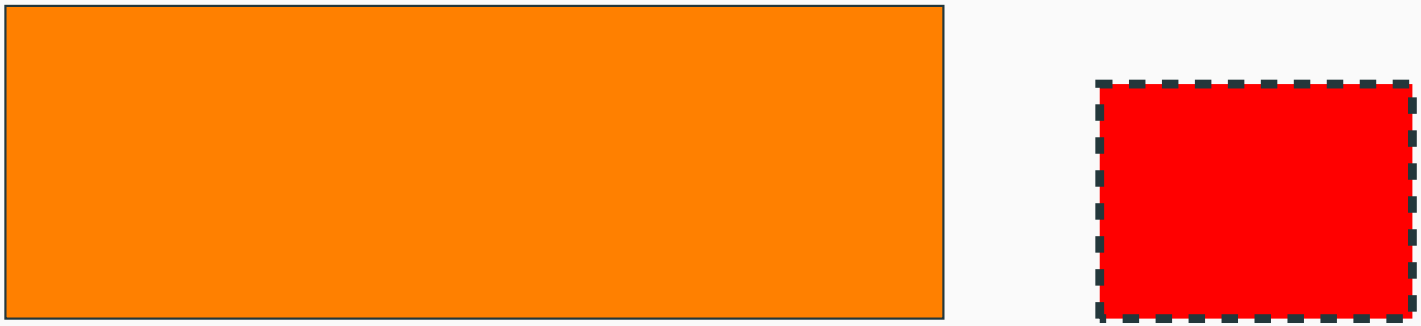
```
\draw [red, fill=yellow] (0,0)--(2,2)--(1.75,0.25)--(0,0);
```



The argument `red` draws a red outline, while `fill=yellow` fills the polygon with specified color.

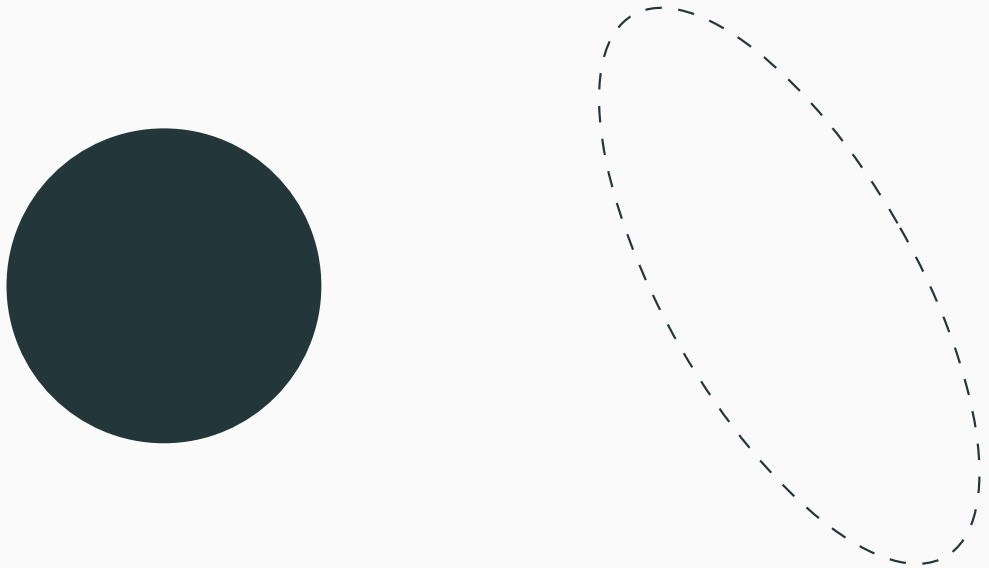
Drawing a rectangle is done with (x_1, y_1) and (x_2, y_2) as opposite corners:

```
\draw [fill=orange] (0,0) rectangle (2,6);
\draw [fill=red, ultra thick, dashed] (7,0) rectangle (9,1.5);
```



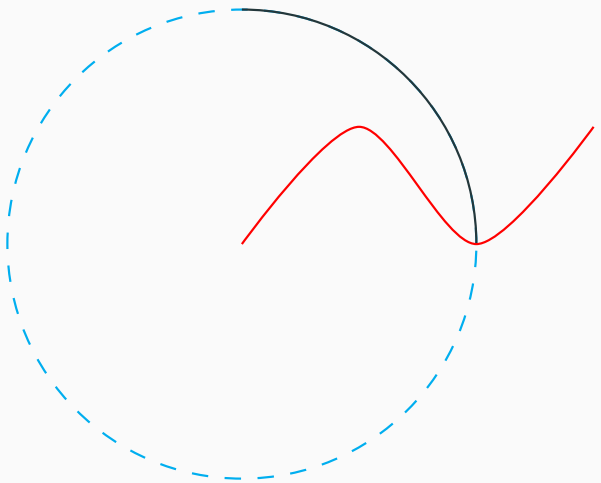
Drawing a circle is done by specifying the center (x, y) and the radius r. For an ellipse we specify the x and y radius:

```
\draw [fill] (0,0) circle [radius=1];
\draw [dashed] (4, 0) circle
        [x radius=0.8, y radius=2, rotate=30];
```



Other drawing commands could be arc and plot.

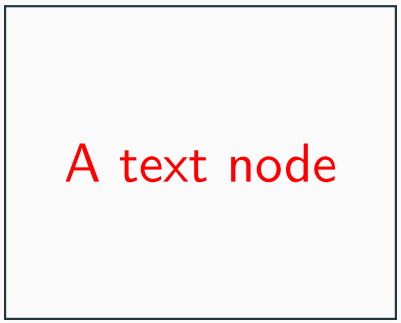
```
\draw [dashed,cyan] (0,0) circle (2);
% Follow arc from (2,0)
% counter-clock 90 degrees, radius 2
\draw (2,0) arc (0:90:2);
% Interpolate some points using a fixed tension
\draw [red] plot [smooth, tension=0.5]
        coordinates { (0,0) (1,1) (2,0) (3,1)};
```



Text labels are done with the **node** object

Example:

```
\draw (0,0) rectangle (2.5,2);
\node [red] at (1.25,1) {A text node};
```

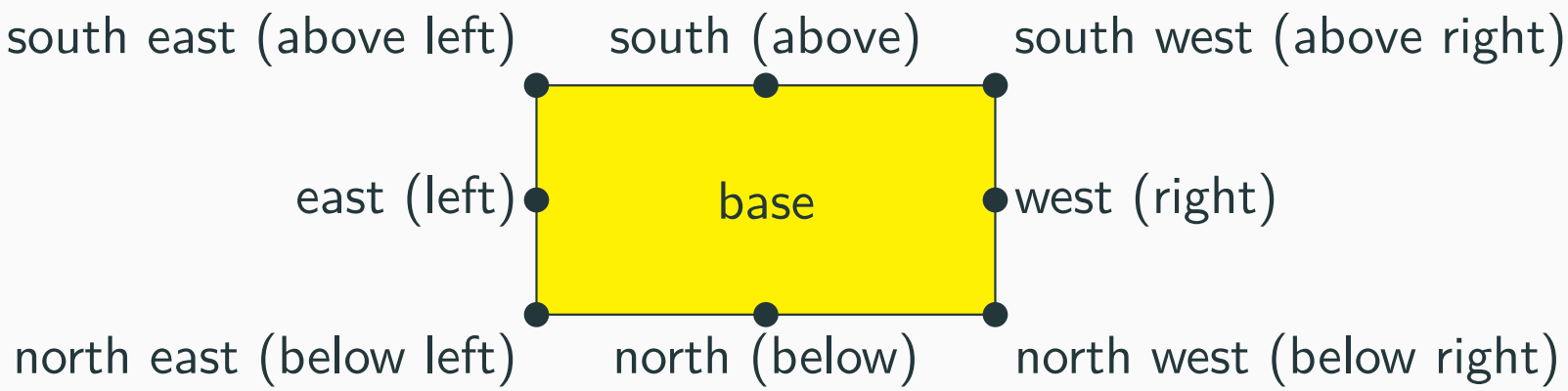


You can also specify the **anchor** for nodes. We can for example use `anchor = west` to have the text anchored to the **right** of the coordinate. To avoid confusion we can also simply state `right` as the option.

Example:

```
\draw (0,0) rectangle[dashed] (1,1);
\node at(0.5,0.5) {Base};
% Anchors to the right
\node [anchor = west] at (1, 0.5) {West};
% Same as anchor = north
\node [below] at (0.5,0) {North};
```

Text nodes anchored by `[anchor = <label>]` with the equivalent option in parenthesis:



Floating Objects

Floating Objects

Most of the time we want our figures or tables to be **floating**. That is, we want them independent and separated from the main text, and placed at the top or bottom of a page along with a caption explaining any figure specific notion or result.

Two important environments are then `figure` and `table`, where the general pattern is

```
\begin{<environment>}[<options>]
  <image or table definitions>
\end{<environment>}
```

Here `<options>` give hints to the compiler of where we want our figure to be. Common values are: `t` (**top**), `b` (**bottom**), `p` (**on its own page**) and `h` (**here**). Note that `<options>` can be left out and that these are only hints for LaTeX. For even more control see the `float` package.

We get captions by having the command

`\caption{`*<text explaining the figure>*`}`

inside the environment.

To center the object in the float, the command `\centering` can be used.
For multiple images in a figure research the `subfig` package.

Example pattern for a floating table with the caption above.

```
\begin{table}
  \centering
  \caption{Rejection of the normality
  hypothesis at significance level $\alpha$}
  \begin{tabular}{columns}
    rows
  \end{tabular}
\end{table}
```

To make an image or `tikzpicture` float we can use `figure`.

Example:

```
\begin{figure}
  \centering
  \includegraphics[width=3cm]{typo.eps}
  \caption{Gutenberg at work.}
\end{figure}
```

Figure 1: Gutenberg at work.

Cross References

Cross References

Multiple parts of the document are automatically numbered in LaTeX, e.g., heading, sections, list items, floating objects, and theorems.

To be able to refer to other parts in the text, we have to tag them with a unique label, using the command `\label{` *label* `}`.

Here it is customary to use prefixes in the naming to mark which type of object that we refer to in code (especially when you have co-authors working on the same document). Examples would be `eq:<`*equation tag*`>` for equations, `sec:<`*section tag*`>` for sections, `fig:<`*figure tag*`>` for figures and so forth.

Thus, if you have a complicated mathematical expression involving Lagrange polynomials in some equation environment and you need to refer to it later in the text. Then `\label{eq:lagrange}` would probably be a good tag.

Where in the code should the `\label` be placed?

Heading	After `\section` etc.
List	After the corresponding `\item`.
Formula	Inside the environment. Corresponding row in a multi-row formula.
Theorem	Inside the environment.
Floating object	After (or inside) `\caption`.

Example:

```
\section{Abelian Groups}
\label{sec:abelian_groups}
\begin{equation}
  \label{eq:abelian}
  a + b = b + a
\end{equation}
```

To refer to a label, we have the following commands:

```
\ref{label}
\pageref{label}
\eqref{label}
```

where the last refers to equations and requires $\mathcal{AMS}$-LaTeX. For even better functionality, see the `hyperref` package.

Example: Suppose that the heading in the previous example is the second section and that it begins on page 14. Also suppose that the formula is the fourth in this section. We now have:

Code	Result
`in section~\ref{sec:abelian_groups}`	in section 2
`on page~\pageref{sec:abelian_groups}`	on page 14
`the equation~\eqref{eq:abelian}`	the equation (2.4)

Bibliography Management

Bibliography Management

One of the easiest ways to maintain and typeset a bibliography in LaTeX is through an external database file. This file must have an extension `.bib` and will contain a list of **records** of different types, each with a **label** and some set of **fields**.

By any choice of **bibliography management system** and **citation style**, we can cite these sources in text using the command `\cite{`$label$`}`. In the end of the document we typeset our list of references (our cited sources) by calling for a print of the bibliography.

E.g., with the system BibTeX we can start this reference section by

```
\bibliographystyle{IEEEtran}
\bibliography{ref.bib}
```

which will create a *Reference*-heading and typeset the list of all cited sources found in `ref.bib` using IEEE-style.

Here the external file *ref.bib* consists of records following the pattern @$type\{label, <fields>\}$.

Example from Google Scholar:

```
@book{aschbacher2000,
  title={Finite group theory},
  author={Aschbacher, Michael},
  volume={10},
  year={2000},
  publisher={Cambridge University Press}
}
```

For more on types and fields within `.bib`-files, see this link for a quick rundown.

We can then cite this book in the text:

```
By Lagrange's theorem, $|H|$ is a divisor of $|G|$
\cite{Aschbacher2000}.
```

Result:

By Lagrange's theorem, $|H|$ is a divisor of $|G|$ [**?**].

Moreover, the generated reference section will update to something like this:

References

When using BibTEX some additional steps may have to be taken when compiling the document (if your editor of choice does not do this automatically)

```
pdflatex myDocument.tex
bibtex myDocument
pdflatex myDocument.tex
```

The first run of `pdflatex` creates the *.aux* file needed for BibTEX to run. `bibtex myDocument` parses the *.aux* file and creates the file *myDocument.bbl*. The second run of *pdflatex* parses this *.bbl* file and adds the references to the document. You might have to run `pdflatex` a third time for all references to appear correctly.

Source Code Listings

Source Code Listings

A good package for typesetting **source code** is `listings`. It introduces an environment `lstlisting` which parses source code within the environment, where we can use it by

```
\begin{lstlisting}[<options>]
   <source code>
\end{lstlisting}
```

The parameters given in `options` can contain a bunch of settings such as '`language = Python`' or '`frame = single`'. If we have many preferences that we need to set, then it is possible to define a code style in the preamble with `\lstdefinestyle`. Quick run down here. Comprehensive view in this document: listings.pdf. For another good source code package (in particular for Python) see `minted`.

```
% Preamble
\usepackage{listings}
\lstdefinestyle{pycode}{
    language = Python,
    basicstyle = \ttfamily,
    keywordstyle=\bfseries\color{blue},
    frame = single,
    backgroundcolor = \color[rgb]{0.9,0.9,0.9}
}
% Body
\begin{lstlisting}[style = pycode,
    caption = Some Python Code]
    def running_mean(x, N):
        csum = np.cumsum(np.insert(x, 0, 0))
        return (csum[N:] - csum[:-N]) / float(N)
\end{lstlisting}
```

Listing 1: Some Python Code

```
def running_mean(x, N):
    csum = np.cumsum(np.insert(x, 0, 0))
    return (csum[N:] - csum[:-N]) / float(N)
```

We may also typeset code directly from source code by `\lstinputlisting{<source code>}`. E.g.,

```
\lstinputlisting[style = pycode,
                 firstline=76,
                 lastline=77]{priomem.py}
```

could give us this:

```
    def updatebeta(self, beta):
        self.beta = beta
```

Presentations

Presentations

A good package for presentations is the `beamer` class, which is responsible for the slides you are looking at now. In `beamer` everything is written inside `frame` environments, one for each slide.

```
% class Beamer
\documentclass{beamer}
% Preamble
\usetheme{Copenhagen}
% In body
\begin{frame}
    <frame stuff>
\end{frame}
```

Introduction to Graph Theory

Björn Lindenberg

October 7, 2019

We can put some content in a frame:

```
\begin{frame}{Graphs}
    \begin{definition}
        A \textbf{graph} $\mathcal{G}$ is an ordered pair
        $\mathcal{G} = (V,E)$ consisting of a \textbf{vertex}
        set
        \begin{equation*}
           V = \{v_1, v_2, v_3, \dots\},
        \end{equation*}
        and an \textbf{edge} set $E$. Each edge in $E$ has one
        or two vertices associated with it, called
        \textbf{endpoints}, and an edge is said
        to \textbf{connect} its endpoints.
    \end{definition}
\end{frame}
```

Graphs

Definition

A **graph** $\mathcal{G}$ is an ordered pair $\mathcal{G} = (V, E)$ consisting of a **vertex** set

$$V = \{v_1, v_2, v_3, \dots\},$$

and an **edge** set E. Each edge in E has one or two vertices associated with it, called **endpoints**, and an edge is said to **connect** its endpoints.

Custom Commands

Custom Commands

To define your own commands you use:

`\newcommand{`*new-command*`}[`n`]{`*code*`}`

where n gives the number of arguments to the command *new-command*, with a maximum of nine. In *code* you refer to the first argument using `#1`, and to the second using `#2` etc.

Note that you may not define an existing command using `\newcommand`. Instead, use `\renewcommand`. But be careful with which commands you redefine.

There are also commands that define new environments or redefine existing environments: `\newenvironment` and `\renewenvironment`, respectively.

- Here is a command that accepts **one** argument:

```
\newcommand{\good}[1]{Good #1!}
```

Example: `\good{heavens}` $\rightarrow$ Good heavens!
Example: `\good{morning}` $\rightarrow$ Good morning!

- The following is a **math mode** command that accepts **two** arguments:

```
\newcommand{\tuple}[2]{#1_1, #1_2, \ldots, #1_{#2}}
```

Example: `$\tuple{x}{n}$` $\rightarrow$ $x_1, x_2, \ldots, x_n$
Example: `$\tuple{\alpha}{15}$` $\rightarrow$ $\alpha_1, \alpha_2, \ldots, \alpha_{15}$

Common Errors & Advice

Common Errors & Advice

Some common errors:

- Misspelling a command.
- Forgetting to close braces in complicated expressions.
- Using a command out of order, e.g., `\alpha` in text mode.
- Ending an environment or ending nested environments in the wrong order.
- Placing `\label` before `\caption`.

Common Errors & Advice

Advice:

- Start writing as much as possible in LaTeX.
- Create and refine templates with often used custom commands and structure.
- A good work flow: Write a few paragraphs, save, compile, correct possible errors, recompile if necessary, review the result and repeat.
- Focus on content. When following IMRaD, write your parts in this order: Results -> Methods -> Discussion -> Introduction.
- Only do document cosmetics as a last step, when your body of text is proofread and overall set in stone.
- Use the wealth of knowledge found online. *The Comprehensive TEX Archive Network* has a very large list of different packages for LaTeX.