https://www.latex-tutorial.com/tutorials/:

LaTeX-Tutorial.com

A simple guide to LaTeX - Step by Step (latex-tutorial.com)

Learn about LaTeX in short lessons with full code examples. A comprehensive guide to basic and advanced features.

These tutorials provide a hands-on introduction to LaTeX. You will see, the usage is very simple. Even if you have only used word processors (e.g. Word) before, you can learn LaTeX in no time.

In the following lessons you will be introduced to all the basic features of LaTeX, one feature at a time. As a beginner, you should either start with the first lesson or, if you just want a very brief introduction, try the interactive quick start guide. While the beginners' course will enable you to typeset your research papers or thesis, the advanced lessons will introduce you to very powerful features, which are a bit harder to grasp, but can boost your productivity.

You can try out the mathematical typesetting engine directly in your browser. I've set up a sandbox for you to try it out.

Tutorials

I'm constantly trying to improve and extend these lectures. Please let me know if you think a crucial feature is not covered on this website.

Contents

00. LaTeX Installation Guide - Easy as pie.

Get LaTeX running and install an editor with only a few mouse clicks. The full power of typesetting with LaTeX, right in front of you.

There are many editors for LaTeX and I don't think there is an editor that fits everyone. It's a matter of personal taste. For this reason, I will show you how to get a basic LaTeX system running, so you can follow along with my tutorials. If you just want to follow along without having to decide for an editor yet, try the online editor from overleaf.com. I've chosen MiKTeX for Windows, because it contains everything you need to compile, but not more, so you will not be confused by an overwhelming user interface. There are also editors with more features, but for me MiKTeX always got the job done.

Linux

If you're running Linux, you can find the *texlive* package in most repositories. Afterwards you can use any text editor to follow along and compile the .tex files with the command line tool *pdflatex*. I don't use a graphical editor myself, but I heard Kile is a good one, but it requires the KDE libraries and Qt to be installed.

Windows

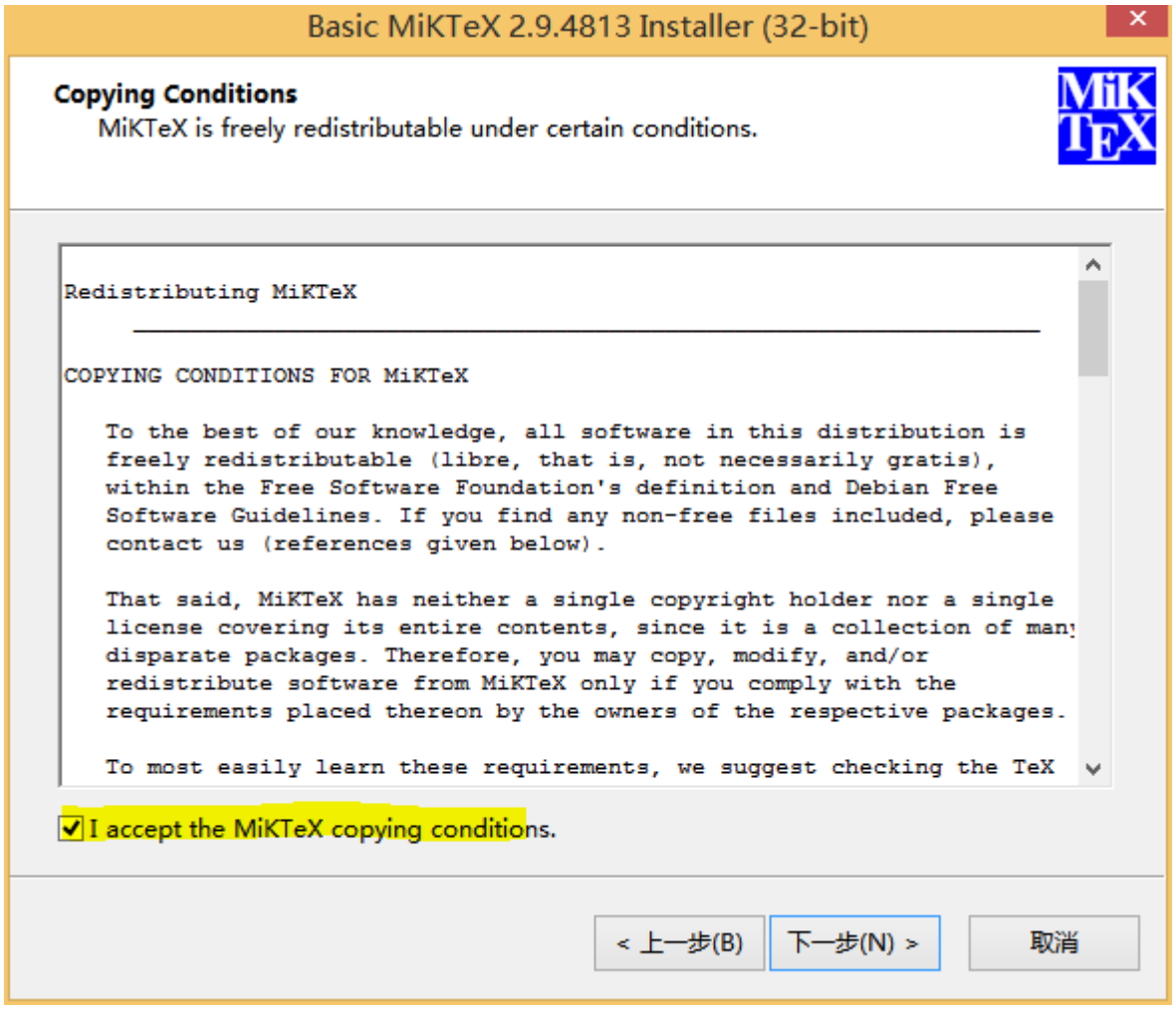
For Windows, a good point to start is definitely installing the MiKTeX bundle. It will manage all the packages for you and also comes with a lightweight and easy to use editor.

Step 1 – Go to miktex.org

Step 2 – Open download section

Step 3 – Download MiKTeX

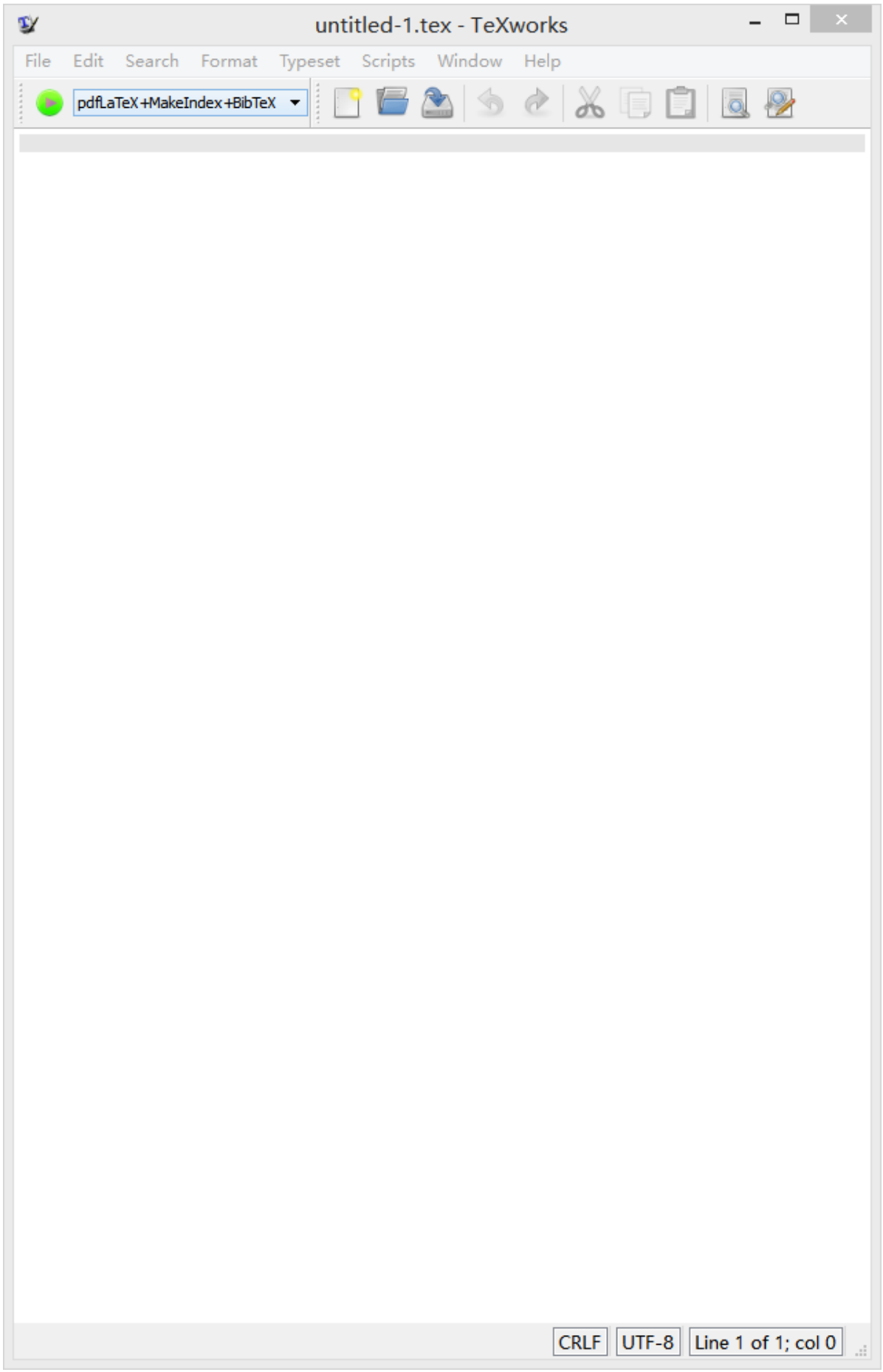
Step 4 – Run MiKTeX Installer



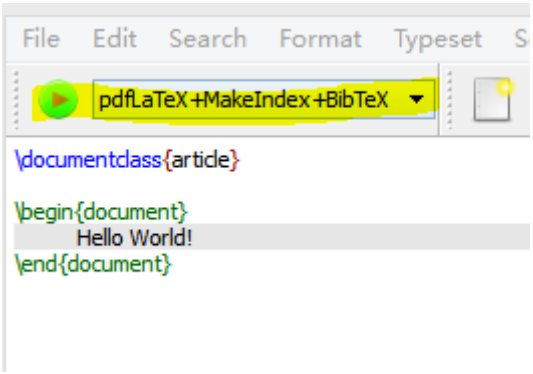
Step 5 – Choose to install missing packages automatically

Step 6 – Open TeXworks

The Installation is complete at this point. TeXworks is the name of your new LaTeX editor for now.



Step 7 – Write code and hit compile



Step 8 – Enjoy your very first document

Now you're able to compile all the code shown on this website and on the blog. All packages will be downloaded automatically. Instructions for Linux and Mac will follow soon.

Mac OS

I personally don't use a Mac, but I got some feedback that the MacTeX distribution is quite good. Simple browse to https://www.tug.org/mactex/ and follow the instructions. The installation should be pretty simple.

Now you're ready to start with the first lesson: 01 First Document

01. Your First LaTeX document

Demonstration of how to create a basic LaTeX document and title page using LaTeX. The basic file layout explained.

The basic layout of a LaTeX file

Creating documents with LaTeX is simple and fun. In contrast to Word, you start off with a plain text file (.tex file) which contains LaTeX code and the actual content (i.e. text). LaTeX uses control statements, which define how your content should be formatted. Before you can see what the final result looks like, the LaTeX compiler will take your .tex file and compile it into a .pdf file. A basic example document can be created with the following code:

```
\documentclass{article}

\begin{document}

  Hello World!

\end{document}
```

Once you translated this code into a PDF document, you will find the text:

Hello World!

along with the page number at the bottom, which is added automatically when using the article class.

Let us now take a closer look at how the magic happens. As you can see, you will find a few statements beginning with a backslash \ in the code example above. This tells LaTeX that this is not actual text, that you want to see printed in your document, but instead is an instruction or command for the LaTeX compiler. All commands share the following structure: *\commandname{option}*. The first part indicates the name of the command and the second part in braces sets an option for this command. The options vary from command to command and you will learn some of them later on in this tutorial.

Most of the time, the commands are pretty self-explanatory: *\documentclass{article}* and what's even greater, you don't have to remember all of them, because you can later just copy and paste them from previous documents. Now let's take a little closer look at the command *\documentclass{article}* The command is obviously named documentclass and it does exactly that, it sets the document class (to article).

LaTeX uses document classes, to influence the overall layout of your document. For instance, there's one class to layout articles, one class to layout books (called book) and many more, which we probably don't need. In my tutorials, I will always use the class *article*. Feel free to play around and try different document classes anyway and see what happens!

This second example differs slightly from the first one, since this command involves a \begin and \end statement. In fact this is not a command but defines an environment. An environment is simply an area of your document where certain typesetting rules apply. It is possible (and usually necessary) to have multiple environments in a document, but it is imperative the *document environment* is the topmost environment. The following code shows how environments can be used:

```
% Valid:

\begin{document}
  \begin{environment1}
    \begin{environment2}
    \end{environment2}
  \end{environment1}
\end{document}

%Invalid:

\begin{document}
  \begin{environment1}
    \begin{environment2}
  \end{environment1}
    \end{environment2}
\end{document}

% Invalid:

\begin{document}
  \begin{environment1}
\end{document}
  \end{environment1}

% Also invalid:

\begin{environment}
  \begin{document}
  \end{document}
\end{environment}
```

Adding a title page

There are numerous choices for environments, and you will most likely need them as soon as you introduce large parts of mathematics or figures to your document. While it is possible to define your own environments, it is very likely that the environment you desire already exists. LaTeX already comes with a few predefined environments and even more come in so called packages, which are subject to another lesson later-on.

Let's try out a few more commands to make our document more interesting:

```
\documentclass{article}

\title{My first document}
\date{2013-09-01}
\author{John Doe}

\begin{document}
  \maketitle
  \newpage

  Hello World!
\end{document}
```

Obviously, the statements \title, \date and \author are not within the *document* environment, so they will not directly show up in our document. The area before our main document is called *preamble*. In this specific example we use it to set up the values for the \maketitle command for later use in our document. This command will automagically create a titlepage for us. The \newpage command speaks for itself.

If we now compile again, we will see a nicely formatted title page, but we can spot a page number at the bottom of our title page. What if we decide, that actually, we don't want to have that page number showing up there. We can remove it, by telling LaTeX to hide the page number for our first page. This can be done by adding the *\pagenumbering{gobble}* command and then changing it back to *\pagenumbering{arabic}* on the next page numbers like so:

```
\documentclass{article}


\title{My first document}
\date{2013-09-01}
\author{John Doe}


\begin{document}
  \pagenumbering{gobble}
  \maketitle
  \newpage
  \pagenumbering{arabic}


  Hello World!
\end{document}
```

That's it. You've successfully created your first LaTeX document. The following lessons will cover how to structure your document and we will then proceed to make use of many features of LaTeX.

Summary

- A document has a *preamble* and *document* part
- The document environment *must* be defined
- Commands beginning with a *backslash* \, environments have a *begin* and *end* tag
- Useful settings for *pagenumbering*:
    - *gobble* - no numbers
    - *arabic* - arabic numbers
    - *roman* - roman numbers

Next Lesson: 02 Sections

02. Using LaTeX paragraphs and sections

Learn how to structure your document using sections and paragraphs in LaTeX. A brief introduction to heading and the basic file layout.

Sectioning elements (sections, subsections, paragraphs etc.)

We have created a very basic document in the previous lesson, but when writing a paper, it's necessary to structure the content into logic units. To achieve this, LaTeX offers us commands to generate section headings and number them automatically. The commands to create section headings are straightforward:

```
\section{}

\subsection{}

\subsubsection{}


\paragraph{}

\subparagraph{}
```

Example output of sections and subsections

The *section* commands are numbered and will appear in the table of contents of your document. *Paragraphs* aren't numbered and won't show in the table of contents. Here an example output using sections:

1 Section

Hello World!

1.1 Subsection

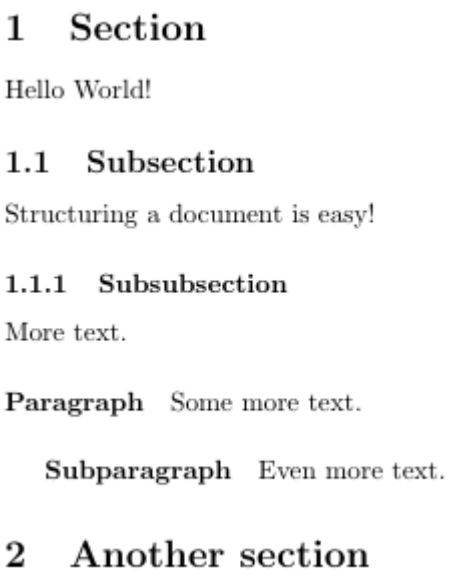
Structuring a document is easy!

In order to get this output, we just have to add a few lines to our program from lesson 1:

```
\documentclass{article}

\title{Title of my document}
\date{2013-09-01}
\author{John Doe}

\begin{document}

\maketitle
\pagenumbering{gobble}
\newpage
\pagenumbering{arabic}

\section{Section}

Hello World!

\subsection{Subsection}

Structuring a document is easy!

\end{document}
```

Hierarchy of sectioning elements

The following picture shows the hierarchical structure of all elements:



1 Section

Hello World!

1.1 Subsection

Structuring a document is easy!

1.1.1 Subsubsection

More text.

Paragraph Some more text.

Subparagraph Even more text.

2 Another section

I have used the following code to get this output:

```
\documentclass{article}

\begin{document}

\section{Section}

Hello World!

\subsection{Subsection}

Structuring a document is easy!

\subsubsection{Subsubsection}

More text.

\paragraph{Paragraph}

Some more text.

\subparagraph{Subparagraph}

Even more text.

\section{Another section}

\end{document}
```

It's very easy to structure documents into sections using LaTeX. This feature also exists in Word, but most people don't use it properly. In LaTeX it is very effortless to have consistent formatting throughout your paper. In the next lesson I will give a short introduction to *packages* and show some basic math typesetting. This is where LaTeX really excels.

Summary

- LaTeX uses the commands *section*, *subsection* and *subsubsection* to define sections in your document
- The *sections* will have successive numbers and appear in the table of contents
- *Paragraphs* are not numbered and thus don't appear in the table of contents

Next Lesson: 03 Packages

03. Using LaTeX packages

Use packages in LaTeX to add more functions. Demonstration of amsmath package and basic math typesetting.

LaTeX offers a lot of functions by default but in some situations, it can become in handy to use so called *packages*. To import a package in LaTeX, you simply add the *\usepackage* directive to the *preamble* of your document:

```
\documentclass{article}


\usepackage{PACKAGENAME}


\begin{document}
...
```

Install a package

When using Linux or Mac, most packages will already be installed by default and it is usually not necessary to install them. In case of Ubuntu installing *texlive-full* from the package manager would provide all packages available. The MiKTeX bundle in Windows, will download the package if you include it to your document.

Purpose of packages

There are countless packages, all for different purposes in my tutorials I will explain some of the most useful. To typeset math, LaTeX offers (among others) an *environment* called *equation*. Everything inside this environment will be printed in *math mode*, a special typesetting environment for math. LaTeX also takes care of equation numbers for us:

```
\documentclass{article}


\begin{document}


\begin{equation}
  f(x) = x^2
\end{equation}


\end{document}
```

This will result in the following output: $f(x) = x^2$ (1)

Using / Including a package

The automatic numbering is a useful feature, but sometimes it's necessary to remove them for auxiliary calculations. LaTeX doesn't allow this by default, now we want to include a package that does:

```
\documentclass{article}


\usepackage{amsmath}


\begin{document}


\begin{equation*}
  f(x) = x^2
\end{equation*}
```

Now we get the same output as before, only the equation number is removed: $f(x) = x^2$

Summary

- *Packages* add new functions to LaTeX
- All *packages* must be included in the *preamble*
- Packages add features such as support for pictures, links and bibliography

Next Lesson: 04 Math

04. LaTeX math and equations

Learn to typeset and align equations, matrices and fractions in LaTeX. Overview of basic math features, with live-rendering and sandbox in your browser.

---

1. Inline math
2. Equations
3. Fractions
4. Matrices
5. Scaling of Parentheses, Brackets etc.

There are two major modes of typesetting math in LaTeX one is embedding the math directly into your text by *encapsulating* your formula in *dollar signs* and the other is using a predefined *math environment*. You can follow along and try the code in the sandbox below. I also prepared a quick reference of math symbols.

Using inline math - embed formulas in your text

To make use of the inline math feature, simply write your text and if you need to typeset a single math symbol or formula, surround it with dollar signs:

```
...
This formula $f(x) = x^2$ is an example.
...
```

Output equation: This formula $f(x) = x^2$ is an example.

The equation and align environment

The most useful *math environments* are the *equation environment* for typesetting single equations and the *align* environment for multiple equations and automatic alignment:

```
\documentclass{article}

\usepackage{amsmath}

\begin{document}

\begin{equation*}
  1 + 2 = 3
\end{equation*}

\begin{equation*}
  1 = 3 - 2
\end{equation*}

\begin{align*}
  1 + 2 &= 3\\
  1 &= 3 - 2
\end{align*}

\end{document}
```

Output Equation:

$$1 + 2 = 3$$

$$1 = 3 - 2$$

Output Align:

$$1 + 2 = 3$$

$$1 = 3 - 2$$

The *align* environment will align the equations at the *ampersand &*. Single equations have to be *separated* by a *linebreak* \\. There is no alignment when using the simple *equation* environment. Furthermore it is not even possible to enter two equations in that environment, it will result in a *compilation error*. The asterisk (e.g. equation*) only indicates, that I don't want the equations to be numbered.

Fractions and more

LaTeX is capable of displaying any mathematical notation. It's possible to typeset integrals, fractions and more. Every command has a specific syntax to use. I will demonstrate some of the most common LaTeX math features:

```
\documentclass{article}


\usepackage{amsmath}


\begin{document}


\begin{align*}
  f(x) &= x^2\\
  g(x) &= \frac{1}{x}\\
  F(x) &= \int^a_b \frac{1}{3}x^3
\end{align*}


\end{document}
```

Output:

$$f(x) = x^2$$

$$g(x) = \frac{1}{x}$$

$$F(x) = \int_b^a \frac{1}{3}x^3$$

It is also possible to combine various commands to create more sophisticated expressions such as:

```
\frac{1}{\sqrt{x}}
```

Output: $\frac{1}{\sqrt{x}}$

The more complex the expression, the more error prone this is, it's important to take care of opening and closing the braces *{}*. It can take a long time to debug such errors. The *Lyx* program offers a great formula editor, which can ease this work a bit. Personally, I write all code by hand though, since it's faster than messing around with the formula editor.

Matrices

Furthermore it's possible to display matrices in LaTeX. There is a special matrix environment for this purpose, please keep in mind that the matrices only work within math environments as described above:

```
\begin{matrix}
1 & 0\\
0 & 1
\end{matrix}
```

Output: $\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$

Brackets in math mode - Scaling

To surround the matrix by brackets, it's necessary to use special statements, because the plain [ ] symbols do not scale as the matrix grows. The following code will result in wrong brackets:

```
[
\begin{matrix}
1 & 0\\
0 & 1
\end{matrix}
]
```

Output: $[\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}]$

To scale them up, we must use the following code:

```
\left[
\begin{matrix}
1 & 0\\
0 & 1
\end{matrix}
\right]
```

Output: $\left[\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}\right]$

This does also work for parentheses and braces and is not limited to matrices. It can be used to scale for fractions and other expressions as well:

```
\left(\frac{1}{\sqrt{x}}\right)
```

Output: $\left(\frac{1}{\sqrt{x}}\right)$

Summary

- LaTeX is a *powerful* tool to typeset math
- *Embed formulas* in your text by *surrounding* them with *dollar signs $*
- The *equation environment* is used to typeset *one* formula
- The *align environment* will align formulas at the *ampersand & symbol*
- Single formulas *must* be separated with *two backslashes* \\
- Use the *matrix environment* to typeset matrices

- Scale parentheses with *left(* *right)* automatically
- All mathematical expressions have a unique command with unique syntax
- Notable examples are:
    - \int^a_b for integral symbol
    - \frac{u}{v} for fractions
    - \sqrt{x} for square roots
- Characters for the *greek alphabet* and other *mathematical symbols* such as *lambda*

Next Lesson: 05 Figures

05. Insert an image in LaTeX - Adding a figure or picture

Learn how to insert images and caption them. Examples for a single figure, and multiple figures next to each other, using the subfigure environment.

Captioned images / figures in LaTeX

From time to time, it's necessary to add pictures to your documents. Using LaTeX all pictures will be indexed automatically and tagged with successive numbers when using the *figure environment* and the *graphicx package*.

```
\documentclass{article}


\usepackage{graphicx}


\begin{document}


\begin{figure}
  \includegraphics[width=\linewidth]{boat.jpg}
  \caption{A boat.}
  \label{fig:boat1}
\end{figure}


Figure \ref{fig:boat1} shows a boat.


\end{document}
```

The code above will create the following pdf:

Figure 1: A boat.

The *figure* environment takes care of the numbering and positioning of the image within the document. In order to include a figure, you must use the *\includegraphics* command. It takes the image width as an option in brackets and the path to your image file. As you can see, I put *\linewidth* into the brackets, which means the picture will be scaled to fit the width of the document. As a result smaller pictures are upscaled and larger pictures downscaled respectively. As I mentioned before the brackets contain the path to the image. In this case the image is stored in the same directory as my .tex file, so I simply put *boat.jpg* here to include it. For large documents, you probably want to store image files in a different folder, say we created a folder *images*, then we would simply write *images/boat.jpg* into the braces. In the next command we set a *\caption*, which is the text shown below the image and a *\label* which is invisible, but useful if we want to refer to our figure in our document. You can use the *\ref* command to refer to the figure (marked by label) in your text and it will then be replaced by the correct number. LaTeX is smart enough to retrieve the correct numbers for all your images automatically. Note that you will need to include the *graphicx* package in order to use this code.

Image positioning / setting the float

At some point, you will notice that the figure doesn't necessarily show up in the exact place as you put your code in the .tex file. If your document contains a lot of text, it's possible that LaTeX will put the picture on the next page, or any other page where it finds sufficient space. To prevent this behavior, it's necessary to set the *float* value for the figure environment.

```
%...
\begin{figure}[h!]
%...
```

Setting the float by adding *[h!]* behind the figure environment \begin tag will force the figure to be shown at the location in the document. Possible values are:

- h (here) - same location
- t (top) - top of page
- b (bottom) - bottom of page
- p (page) - on an extra page
- ! (override) - will force the specified location

However, I have only used the *[h!]* option so far. The *float package* (\usepackage{float}) allows to set the option to *[H]*, which is even stricter than [h!].

Multiple images / subfigures in LaTeX

Sometimes when writing a document, adding single images is not optimal, especially when the reader is supposed to compare several results or graphs. In such situations, it might be necessary to use a different environment, called *subfigure*. The subfigure environment allows you to place multiple images at a certain location next to each other and the usage is pretty straightforward.

First you need to add the subcaption package to your preamble:

```
\documentclass{article}


\usepackage{graphicx}
\usepackage{subcaption}


\begin{document}


%...


\end{document}
```

Next, you need to add multiple subfigure environments within a figure environment.

```
%...


\begin{figure}[h!]
  \centering
  \begin{subfigure}[b]{0.4\linewidth}
    \includegraphics[width=\linewidth]{coffee.jpg}
    \caption{Coffee.}
  \end{subfigure}
  \begin{subfigure}[b]{0.4\linewidth}
    \includegraphics[width=\linewidth]{coffee.jpg}
    \caption{More coffee.}
  \end{subfigure}
  \caption{The same cup of coffee. Two times.}
  \label{fig:coffee}
\end{figure}


%...
```

This will show two pictures next to each other in your document, like this:

(a) Coffee. (b) More coffee.

Figure 1: The same cup of coffee. Two times.

If you look closely, you will see, that I've set the width of the image manually:

```
%...
\begin{subfigure}[b]{0.4\linewidth}
%...
```

and even though there are two images aligned next to each other, their widths are both set to 0.4, yet they fill up the whole space. You should always set this value to .1 less than you expect. If you want to align three images next to each other, you should consecutively add three subfigures, each with a 0.2\linewidth. I suggest, if you need some other arrangement, you simply play around with the width factor until you are satisfied with the result. A more elaborate example with multiple rows and columns could look like this:

```
%...
\begin{figure}[h!]
  \centering
  \begin{subfigure}[b]{0.2\linewidth}
    \includegraphics[width=\linewidth]{coffee.jpg}
     \caption{Coffee.}
  \end{subfigure}
  \begin{subfigure}[b]{0.2\linewidth}
    \includegraphics[width=\linewidth]{coffee.jpg}
    \caption{More coffee.}
  \end{subfigure}
  \begin{subfigure}[b]{0.2\linewidth}
    \includegraphics[width=\linewidth]{coffee.jpg}
    \caption{Tasty coffee.}
  \end{subfigure}
  \begin{subfigure}[b]{0.5\linewidth}
    \includegraphics[width=\linewidth]{coffee.jpg}
    \caption{Too much coffee.}
  \end{subfigure}
  \caption{The same cup of coffee. Multiple times.}
  \label{fig:coffee3}
\end{figure}
%...
```

This will print out the following figure in your document:

(a) Coffee. (b) More coffee. (c) Tasty coffee.

(d) Too much coffee.

Figure 3: The same cup of coffee. Multiple times.

Summary

- Use the *graphicx package* and *figure environment* to embed pictures
- Pictures will be numbered automatically
- Change the width of your image by using *\includegraphics[width=\linewidth]{}*
- Refer to pictures in your document by setting a *\label* and using the *\ref* tag
- Set the position of your image by adding a float option such as *[h!]*
- If you want to show multiple figures next to each other, use the *subcaption* package and the *subfigure* environment

Next Lesson: 06 Table of Contents

06. Generate a table of contents in LaTeX

LaTeX offers features to automatically generate a table of contents, a list of figures and a list of tables. Learn here how to use them.

Table of contents

Generating a *table of contents* can be done with a few simple commands. LaTeX will use the section headings to create the *table of contents* and there are commands to create a *list of figures* and a *list of tables* as well. I will give a small example code to create a *table of contents* first:

```
\documentclass{article}


\begin{document}


\tableofcontents
\newpage


\section{Section}


Dummy text


\subsection{Subsection}


Dummy text


\end{document}
```

After compiling the .tex file *two* times, you will get the following table of contents:

List of figures / tables

The generation of a list of figures and tables works the same way. I added a dummy figure and table and put the lists in the appendix of my document:

```
\begin{document}
...
\begin{figure}
  \caption{Dummy figure}
\end{figure}

\begin{table}
  \caption{Dummy table}
\end{table}
...
\begin{appendix}
  \listoffigures
  \listoftables
\end{appendix}

\end{document}
```

After compiling *two* times again, the lists will be generated like this:

Depth

Sometimes it makes sense to only show a subset of the headings for all sections or for a particular section. For this reason you can set a the *tocdepth* by using the command \setcounter{tocdepth}{X}, where X is the desired depth. A value of 0 means that your table of contents will show nothing at all and 5 means, that even subparagraphs will be shown. The value has to be set in the preamble of your document and automatically applies to the whole document:

```
% ...


\setcounter{tocdepth}{1} % Show sections
%\setcounter{tocdepth}{2} % + subsections
%\setcounter{tocdepth}{3} % + subsubsections
%\setcounter{tocdepth}{4} % + paragraphs
%\setcounter{tocdepth}{5} % + subparagraphs


\begin{document}
%...
\tableofcontents
%...
\end{document}
```

Using the example from above, the setting tocdepth = 1 will lead to the following output:

You can easily increase the verbosity of your table of contents, by setting tocdepth to something higher like 3, which would lead to the following output:

If you don't want to change the depth for all sections, you can also adjust the tocdepth for each section individually. In this case you don't have to set the tocdepth before the section which should have more or less depth.

```
%...
\begin{document}
%...
\addtocontents{toc}{\setcounter{tocdepth}{1}} % Set depth to 1
\section{Another section}
\subsection{Subsection}
\subsubsection{Subsubsection}
%...
\addtocontents{toc}{\setcounter{tocdepth}{3}} % Reset to default (3)
\end{document}
```

This will generate the following table of contents, using the default tocdepth for the first section, but tocdepth = 1 for this section:

Spacing

If you're not happy with the spacing of the headings in your table of content, the easiest way of changing the spacing of your table of contents (and document in general) is by using the setspace package. First add *\usepackage{setspace}* to your preamble:

```
%...
\usepackage{setspace}
%...
\begin{document}
%...
```

You can then proceed to set the spacing for individual parts of your document, including the table of contents like so:

```
%...
\begin{document}
%...
\doublespacing
\tableofcontents
\singleplacing
%...
```

Which will lead to the following output:

Summary

- Autogenerate a table of content using *\tableofcontents*
- Create lists of your figures and tables with *\listoffigures* and *\listoftables*
- Always compile *twice* to see the changes
- Globally change the depth with *\setcounter{tocdepth}{X}*; X = {1,2,3,4,5}
- For single sections use *\addtocontents{toc}{\setcounter{tocdepth}{X}}* instead.

Next Lesson: 07 BiBTeX

07. Bibliography in LaTeX with Bibtex/Biblatex

Learn how to create a bibliography with Bibtex and Biblatex in a few simple steps. Create references / citations and autogenerate footnotes.

We have looked at many features of LaTeX so far and learned that many things are automated by LaTeX. There are functions to add a table of contents, lists of tables and figures and also several packages that allow us to generate a bibliography. I will describe how to use bibtex and biblatex (both external programs) to create the bibliography. At first, we have to create a .bib file, which contains our bibliographic information.

Creating a .bib file

A .bib file will contain the bibliographic information of our document. I will only give a simple example, since there are many tools to generate the entries automatically. I will not explain the structure of the file itself at this point, since I suggest using a *bibtex generator* (choose one from google). Our example will contain a single book and look like this:

```
@BOOK{DUMMY:1,

AUTHOR="John Doe",

TITLE="The Book without Title",

PUBLISHER="Dummy Publisher",

YEAR="2100",

}
```

If you don't want to use a BibTeX generator or a reference management tool like Citavi (which generates BibTeX files automatically for you), you can find more examples of BibTeX formats here.

Using BibTeX

After creating the bibtex file, we have to tell LaTeX where to find our bibliographic database. For BibTeX this is not much different from printing the table of contents. We just need the commands *\bibliography* which tells LaTeX the location of our .bib file and *\bibliographystyle* which selects one of various bibliographic styles.

```
\documentclass{article}


\begin{document}


Random citation \cite{DUMMY:1} embeddeed in text.


\newpage


\bibliography{lesson7a1}
\bibliographystyle{ieeetr}


\end{document}
```

By using this code, we will obtain something like this:

Random citation [1] embeddeed in text.

References

[1] J. Doe, *The Book without Title*. Dummy Publisher, 2100.

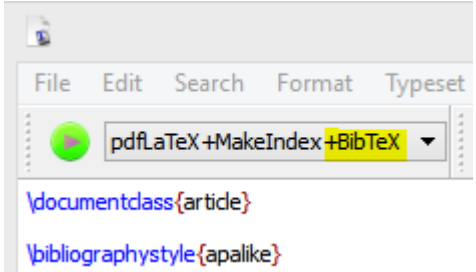
I named my .bib file lesson7a1.bib, note that I did not enter the .bib extension. For the style, I've chosen the *ieeetr* style, which is very common for my subject, but there are many more styles available. Which will change the way our references look like. The *ieeetr* style will mark citations with successive numbers such as [1] in this example. If I choose the style to *apalike* instead, I will get the following result:

Random citation [Doe, 2100] embeddeed in text.

References

[Doe, 2100] Doe, J. (2100). *The Book without Title*. Dummy Publisher.

Most editors will let you select, to run bibtex automatically on compilation. In TeXworks (MiKTeX) for example, this should be selected by default.



If you use a different editor, it can be necessary to execute the bibtex command manually. In a command prompt/shell simply run:

```
pdflatex lesson7a1.tex
bibtex lesson7a1
pdflatex lesson7a1.tex
pdflatex lesson7a1.tex
```

It is necessary to execute the pdflatex command, before the bibtex command, to tell bibtex what literature we cited in our paper. Afterwards the .bib file will be translated into the proper output for out references section. The next two steps merge the reference section with our LaTeX document and then assign successive numbers in the last step.

Autogenerate footnotes in LaTeX using BibLaTeX

The abilities of BibTeX are limited to basic styles as depicted in the examples shown above. Sometimes it is necessary to cite all literature in footnotes and maintaining all of them by hand can be a frustrating task. At this point BibLaTeX kicks in and does the work for us. The syntax varies a bit from the first document. We now have to include the *biblatex package* and use the *\autocite* and *\printbibliography* command. It is crucial to move the *\bibliography{lesson7a1}* statement to the *preamble* of our document:

```
\documentclass{article}


\usepackage[backend=bibtex,style=verbose-trad2]{biblatex}
\bibliography{lesson7a1}


\begin{document}


Random citation \autocite[1]{DUMMY:1} embeddeed in text.


\newpage


\printbibliography


\end{document}
```

The *\autocite* command generates the footnotes and we can enter a page number in the brackets *\autocite[1]{DUMMY:1}* will generate a footnote like this:

[1] John Doe. *The Book without Title*. Dummy Publisher, 2100, p. 1.

For BibLaTeX we have to choose the citation style on package inclusion with:

```
\usepackage[backend=bibtex,style=verbose-trad2]{biblatex}
```

The *backend=bibtex* part makes sure to use BibTeX instead of Biber as our backend, since Biber *fails* to work in some editors like TeXworks. It took me a while to figure out how to generate footnotes automatically, because the sources I found on the internet, didn't mention this at all.

BibTeX Formats

This is not meant to be a comprehensive list of BibTeX formats, but rather give you an idea of how to cite various sources properly. If you're interested in an extensive overview of all BibTeX formats, I suggest you to check out the resources on Wikibooks.

Article

```
@ARTICLE{ARTICLE:1,
    AUTHOR="John Doe",
    TITLE="Title",
    JOURNAL="Journal",
    YEAR="2017",
}
```

[3] J. Doe, "Title," *Journal*, 2017.

Book

```
@BOOK{BOOK:1,
    AUTHOR="John Doe",
    TITLE="The Book without Title",
    PUBLISHER="Dummy Publisher",
    YEAR="2100",
}
```

[1] J. Doe, *The Book without Title*. Dummy Publisher, 2100.

Book

Inbook (specific pages)

```
@INBOOK{BOOK:2,
    AUTHOR="John Doe",
    TITLE="The Book without Title",
    PUBLISHER="Dummy Publisher",
    YEAR="2100",
    PAGES="100-200",
}
```

[4] J. Doe, *The Book without Title*, pp. 100–200. Dummy Publisher, 2100.

Website

```
@MISC{WEBSITE:1,
        HOWPUBLISHED = "\url{http://example.com}",
        AUTHOR = "Intel",
        TITLE = "Example Website",
        MONTH = "Dec",
        YEAR = "1988",
        NOTE = "Accessed on 2012-11-11"
}
```

[2] Company, "Example." http://example.com, Dec 1988. Accessed on 2012-11-11.

This is a list of the formats that I have most commonly used. If you think some important format is missing here, please let me know.

BibTeX Styles

Here's a quick overview of some popular styles to use with BibTeX.

Abbrv

References

[1] J. Doe. Title. *Journal*, 2017.

[2] J. Doe. *The Book without Title*. Dummy Publisher, 2100.

[3] J. Doe. *The Book without Title*, pages 100–200. Dummy Publisher, 2100.

[4] Intel. 8259a - programmable interrupt controller. http://bochs.sourceforge.net/techspec/intel-8259a-pic.pdf.gz, Dec 1988. Accessed on 2012-11-11.

Alpha

References

[Doe, 2017] Doe, J. (2017). Title. *Journal*.

[Doe, 2100a] Doe, J. (2100a). *The Book without Title*. Dummy Publisher.

[Doe, 2100b] Doe, J. (2100b). *The Book without Title*, pages 100–200. Dummy Publisher.

[Intel, 1988] Intel (1988). 8259a - programmable interrupt controller. http://bochs.sourceforge.net/techspec/intel-8259a-pic.pdf.gz. Accessed on 2012-11-11.

Apalike

References

[Doe17] John Doe. Title. *Journal*, 2017.

[Doe00a] John Doe. *The Book without Title*. Dummy Publisher, 2100.

[Doe00b] John Doe. *The Book without Title*, pages 100–200. Dummy Publisher, 2100.

[Int88] Intel. 8259a - programmable interrupt controller. http://bochs.sourceforge.net/techspec/intel-8259a-pic.pdf.gz, Dec 1988. Accessed on 2012-11-11.

IEEEtr

References

[1] J. Doe, *The Book without Title*. Dummy Publisher, 2100.

[2] Intel, "8259a - programmable interrupt controller." http://bochs.sourceforge.net/techspec/intel-8259a-pic.pdf.gz, Dec 1988. Accessed on 2012-11-11.

[3] J. Doe, "Title," *Journal*, 2017.

[4] J. Doe, *The Book without Title*, pp. 100–200. Dummy Publisher, 2100.

Plain

References

[1] John Doe. Title. *Journal*, 2017.

[2] John Doe. *The Book without Title*. Dummy Publisher, 2100.

[3] John Doe. *The Book without Title*, pages 100–200. Dummy Publisher, 2100.

[4] Intel. 8259a - programmable interrupt controller. http://bochs.sourceforge.net/techspec/intel-8259a-pic.pdf.gz, Dec 1988. Accessed on 2012-11-11.

I'm trying to keep this list updated with other commonly used styles. If you're missing something here, please let me know.

Summary

- Generate a bibliography with BibTeX and BibLaTeX
- First define a .bib file using: *\bibliography{BIB_FILE_NAME}* (do *not* add .bib)
- For *BibTeX* put the \bibliography statement in your *document*, for*BibLaTeX* in the *preamble*
- *BibTeX* uses the *\bibliographystyle* command to set the citation style
- *BibLaTeX* chooses the style as an option like: *\usepackage[backend=bibtex, style=verbose-trad2]{biblatex}*
- *BibTeX* uses the *\cite* command, while *BibLaTeX* uses the *\autocite* command
- The *\autocite* command takes the *page number* as an option: *\autocite[NUM]{}*

Next Lesson: 08 Footnotes

08. LaTeX Footnotes – Quick Explanation

Learn how to create footnotes in LaTeX and how to refer to them, using the built-in commands footnote, label and ref.

One of the benefits of using LaTeX is how easy footnotes can be added and referred to. So how to use footnotes? LaTeX offers the *\footnote* command and referencing works using the *\label* and *\ref* commands.

The following code shows some example text and how to add a footnote with a label:

```
...

This is some example text\footnote{\label{myfootnote}Hello footnote}.

...
```

After compilation you will see the footnote appearing on the bottom of your page. It's imperative, that the label is contained within the footnote itself, otherwise the label will refer to the section (or subsection). Footnotes are numbered automatically. If you want to refer to them later on, you can use the *\ref* command as follows:

```
...

I'm referring to footnote \ref{myfootnote}.

...
```

Summary

- Create footnotes with the *\footnote* command and label them with *\label*
- Make sure that the label is contained within the braces of the footnote command
- Use the *\ref* command to refer to footnotes

Next lesson: 09 Tables

09. LaTeX tables - Tutorial with code examples

Learn to create tables in LaTeX including all features such as multi row, multi column, multipage and landscape tables. All in one place.

In this tutorial we're going to learn how to use the *table* and *tabular* environments to create tables in LaTeX. At first, we're going to create a simple table like this:

Table 1: Your first table.

Value 1	**Value 2**	**Value 3**
α	β	γ
1	1110.1	a
2	10.1	b
3	23.113231	c

After showing you how to modify this table according to your needs, I will also show you how to make your tables prettier and turn the table above into this:

Table 8: Table with aligned units.

Value 1	**Value 2**	**Value 3**
α	β	γ
1	1110.10	a
2	10.10	b
3	23.11	c

Of course it's up to your personal preference, but most of the time, I've found that the second table is much more readable and easier on the eye than the first table.

Afterwards I'm also going to show you, how to do some more elaborate things such as having rows and columns spend multiple cells as well as orienting tables sideways on the page (useful for tables with many columns) and how to have tables span multiple pages (useful for tables with many rows).

I've also created a tool to edit LaTeX tables right in your browser. This feature is still experimental, but if you want to try it, you can find it here. I appreciate any feedback, so I can create a tool that you love using.

Your first table

Tables in LaTeX can be created through a combination of the *table* environment and the tabular *environment*. The *table* environment part contains the caption and defines the float for our table, i.e. where in our document the table should be positioned and whether we want it to be displayed centered. The *caption* and *label* commands can be used in the same way as for pictures. The actual content of the table is contained within the *tabular* environment.

The *tabular* environment uses *ampersands* & as *column separators* and *newline symbols* \\ as row separators. The vertical lines separating the columns of our table (|) are passed as an argument to the tabular environment (e.g.\begin{tabular}{l|c|r}) and the letters tell whether we want to align the content to the left (l), to the center (c) or to the right (r) for each column. There should be one letter for every column and a vertical line in between them or in front of them, if we want a vertical line to be shown in the table. Row separators can be added with the *hline* command.

Now let's take a look at some actual code for a basic table, which you can easily copy-and-paste into your document and modify it to your needs.

```
\documentclass{article}

\begin{document}

\begin{table}[h!]
  \begin{center}
    \caption{Your first table.}
    \label{tab:table1}
    \begin{tabular}{l|c|r} % <-- Alignments: 1st column left, 2nd middle and 3rd right, with vertical lines in between
      \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
      $\alpha$ & $\beta$ & $\gamma$ \\
      \hline
      1 & 1110.1 & a\\
      2 & 10.1 & b\\
      3 & 23.113231 & c\\
    \end{tabular}
  \end{center}
\end{table}

\end{document}
```

The above code will print out the table which I've already shown you in the introduction and it looks like this:

Table 1: Your first table.

Value 1	**Value 2**	**Value 3**
α	β	γ
1	1110.1	a
2	10.1	b
3	23.113231	c

While this table already works, it's not very satisfying and readable that the numbers in the center column are not aligned at the decimal point. Fortunately, we don't have to add spacing somehow manually, but we can use the *siunitx* package for this purpose.

Align numbers at decimal point

The first thing we have to do is to include the *siunitx* package in our preamble and use the command *\sisetup* to tell the package how many digital places it should display:

```
%...

\usepackage{siunitx} % Required for alignment

\sisetup{
  round-mode          = places, % Rounds numbers
  round-precision     = 2, % to 2 places
}

\begin{document}

%...
```

Afterwards we can use a new alignment setting in our tables, so besides left (l), center (c) and right (r), there's now an additional setting S, which will align the numbers automatically. In our previous table, there was an alignment problem with the middle column, so I've now changed the alignment setting of the middle column from (c) to (S):

```
%...

\begin{table}[h!]
  \begin{center}
    \caption{Table with aligned units.}
    \label{tab:table1}
    \begin{tabular}{l|S|r} % <-- Changed to S here.
      \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
      $\alpha$ & $\beta$ & $\gamma$ \\
      \hline
      1 & 1110.1 & a\\
      2 & 10.1 & b\\
      3 & 23.113231 & c\\
    \end{tabular}
  \end{center}
\end{table}

%...
```

We can now observe, that LaTeX will now properly align the numbers at their decimal points and round the numbers to two decimal places:

Table 2: Table with aligned units.

Value 1	**Value 2**	**Value 3**
α	β	γ
1	1110.10	a
2	10.10	b
3	23.11	c

Adding rows and columns

Now that we've setup our table properly, we can focus on adding more rows and columns. As I've mentioned before, LaTeX uses column separators (&) and row separators (\\) to layout the cells of our table. For the 5x3 table shown above we can count five times (\\) behind each row and two times (&) per row, separating the content of three columns.

If we now want to add an additional column, it's as simple as copy and pasting the previous column and changing the contents. I will be reusing the table from above for this example and add an additional column:

```
%...

\begin{table}[h!]
  \begin{center}
    \caption{More rows.}
    \label{tab:table1}
    \begin{tabular}{l|S|r}
      \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
      $\alpha$ & $\beta$ & $\gamma$ \\
      \hline
      1 & 1110.1 & a\\
      2 & 10.1 & b\\
      3 & 23.113231 & c\\
      4 & 25.113231 & d\\ % <-- added row here
    \end{tabular}
  \end{center}
\end{table}

%...
```

This will generate the following output:

Table 3: More rows.

Value 1	**Value 2**	**Value 3**
α	β	γ
1	1110.10	a
2	10.10	b
3	23.11	c
4	25.11	d

Adding an additional column is also possible, but you have to be careful, because you have to add a column separator (&) to every column:

```
%...

\begin{table}[h!]
  \begin{center}
    \caption{More columns.}
    \label{tab:table1}
    \begin{tabular}{l|S|r|l}
      \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3} & \textbf{Value 4}\\ % <-- added & and content for each column
      $\alpha$ & $\beta$ & $\gamma$ & $\delta$ \\ % <--
      \hline
      1 & 1110.1 & a & e\\ % <--
      2 & 10.1 & b & f\\ % <--
      3 & 23.113231 & c & g\\ % <--
    \end{tabular}
  \end{center}
\end{table}

%...
```

We will now see an additional column in our output:

Table 4: More columns.

Value 1	**Value 2**	**Value 3**	**Value 4**
α	β	γ	δ
1	1110.10	a	e
2	10.10	b	f
3	23.11	c	g

Cells spanning multiple rows or multiple columns

a. Using multirow
b. Using multicolumn
c. Combining multirow and multicolumn

Sometimes it's necessary to make a row span several cells. For this purpose we can use the *multirow* package, so the first thing we're going to do is adding the required package to our preamble:

```
%...

\usepackage{multirow} % Required for multirows

\begin{document}

%...
```

We can now use *multirow* and *multicolumn* environments, which allow us to conveniently span multiple rows or columns.

Using multirow

In order for a cell to span multiple rows, we have to use the *multirow* command. This command accepts three parameters:

```
\multirow{NUMBER_OF_ROWS}{WIDTH}{CONTENT}
```

I usually use an asterisk (*) as a parameter for the width, since this basically means, that the width should be determined automatically.

Because we're combining two rows in our example, it's necessary to omit the content of the same row in the following line. Let's look at how the actual LaTeX code would look like:

```
%...

\begin{table}[h!]
  \begin{center}
    \caption{Multirow table.}
    \label{tab:table1}
    \begin{tabular}{l|S|r}
      \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
      $\alpha$ & $\beta$ & $\gamma$ \\
      \hline
      \multirow{2}{*}{12} & 1110.1 & a\\ % <-- Combining 2 rows with arbitrary with (*) and content 12
      & 10.1 & b\\ % <-- Content of first column omitted.
      \hline
      3 & 23.113231 & c\\
      4 & 25.113231 & d\\
    \end{tabular}
  \end{center}
\end{table}

%...
```

The modified table looks like this:

Table 5: Multirow table.

Value 1	**Value 2**	**Value 3**
α	β	γ
12	1110.10	a
	10.10	b
3	23.11	c
4	25.11	d

You can now see, that the cell containing *12* spans two rows.

Using multicolumn

If we want a cell to span multiple columns, we have to use the *multicolumn* command. The usage differs a bit from *multirow* command, since we also have to specify the alignment for our column. The command also requires three parameters:

```
\multicolumn{NUMBER_OF_COLUMNS}{ALIGNMENT}{CONTENT}
```

In our example, we will again combine two neighboring cells, note that in the row where we're using multicolumn to span two columns, there's only one column separator (&) (instead of two for all other rows):

```
%...


\begin{table}[h!]
  \begin{center}
    \caption{Multicolumn table.}
    \label{tab:table1}
    \begin{tabular}{l|S|r}
      \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
      $\alpha$ & $\beta$ & $\gamma$ \\
      \hline
      \multicolumn{2}{c|}{12} & a\\ % <-- Combining two cells with alignment c| and content 12.
      \hline
      2 & 10.1 & b\\
      3 & 23.113231 & c\\
      4 & 25.113231 & d\\
    \end{tabular}
  \end{center}
\end{table}


%...
```

This will result in the following content:

Table 6: Multicolumn table.

Value 1	**Value 2**	**Value 3**
α	β	γ
12		a
2	10.10	b
3	23.11	c
4	25.11	d

Combining multirow and multicolumn

Of course it's also possible to combine the two features, to make a cell spanning multiple rows and columns. To do this, we simply use the *multicolumn* command and instead of specifying content, we add a *multirow* command as the content. We then have to add another *multicolumn* statement for as many rows as we're combining.

Because this is a little hard to explain, it will be much clearer when looking at the code. In this example, we're going to combine two columns and two rows, so we're getting a cell spanning a total of four cells:

```
%...


\begin{table}[h!]
  \begin{center}
    \caption{Multirow and -column table.}
    \label{tab:table1}
    \begin{tabular}{l|S|r}
      \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
      $\alpha$ & $\beta$ & $\gamma$ \\
      \hline
      \multicolumn{2}{c|}{\multirow{2}{*}{1234}} & a\\ % <-- Multicolumn spanning 2 columns, content multirow spanning two rows
      \multicolumn{2}{c|}{} & b\\ % <-- Multicolumn spanning 2 columns with empty content as placeholder
      \hline
      3 & 23.113231 & c\\
      4 & 25.113231 & d\\
    \end{tabular}
  \end{center}
\end{table}


%...
```

Our document will now contain a table, with a huge cell:

Table 7: Multirow and -column table.

Value 1	**Value 2**	**Value 3**
α	β	γ
1234		a b
3	23.11	c
4	25.11	d

Prettier tables with booktabs

Of course beauty is always in the eye of the beholder, but I personally think, that the default *hlines* used by the table environment are not very pretty. For my tables, I always use the *booktabs* package, which provides much prettier horizontal separators and the usage is not harder compared to simply using hlines.

Again, we have to add the according *booktabs* package to our preamble:

```
%...

\usepackage{booktabs} % For prettier tables

\begin{document}

%...
```

We can now replace the hlines in our example table with *toprule*, *midrule* and *bottomrule* provided by the booktabs package:

```
%...

\begin{table}[h!]
  \begin{center}
    \caption{Table using booktabs.}
    \label{tab:table1}
    \begin{tabular}{l|S|r}
      \toprule % <-- Toprule here
      \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
      $\alpha$ & $\beta$ & $\gamma$ \\
      \midrule % <-- Midrule here
      1 & 1110.1 & a\\
      2 & 10.1 & b\\
      3 & 23.113231 & c\\
      \bottomrule % <-- Bottomrule here
    \end{tabular}
  \end{center}
\end{table}

%...
```

You can decide for yourself, if you prefer the *hlines* or the following output:

Table 8: Table with aligned units.

Value 1 α	Value 2 β	Value 3 γ
1	1110.10	a
2	10.10	b
3	23.11	c

Multipage tables

If you have a lot of rows in your table, you will notice that by default, the table will be cropped at the bottom of the page, which is certainly not what you want. The package *longtable* provides a convenient way, to make tables span multiple pages. Of course we have to add the package to our preamble before we can start using it:

```
%...

\usepackage{longtable} % To display tables on several pages

\begin{document}

%...
```

It's actually not harder, but easier to use than the previous code for tables. I will first show you what the code looks like and then explain the differences between longtable and tabular, in case they're not obvious.

```
%...

\begin{longtable}[c]{l|S|r} % <-- Replaces \begin{table}, alignment must be specified here (no more tabular)
  \caption{Multipage table.}
  \label{tab:table1}\\
  \toprule
  \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
  $\alpha$ & $\beta$ & $\gamma$ \\
  \midrule
  \endfirsthead % <-- This denotes the end of the header, which will be shown on the first page only
  \toprule
  \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
  $\alpha$ & $\beta$ & $\gamma$ \\
  \midrule
  \endhead % <-- Everything between \endfirsthead and \endhead will be shown as a header on every page
  1 & 1110.1 & a\\
  2 & 10.1 & b\\
  % ...
  % ... Many rows in between
  % ...
  3 & 23.113231 & c\\
  \bottomrule
\end{longtable}

%...
```

In the previous examples, we've always used the table and tabular environments. The longtable environment replaces both of them or rather combines both of them into a single environment. We now use *\begin{longtable}[POSITION_ON_PAGE]{ALIGNMENT}* as an environment for our

tables. Using this environment, we create a table, that is automatically split between pages, if it has too many rows.

The content on the table on the first page looks like this:

Table 9: Multipage table.

Value 1 α	**Value 2** β	**Value 3** γ
1	1110.10	a
2	10.10	b
...	...	...
...	...	...

Imagine that this table has many rows (...) being a placeholder for more rows and that the table continues the following page like this:

Value 1 α	**Value 2** β	**Value 3** γ
...	...	...
...	...	...
...	...	...

Note that there's again a header on this page, but without the caption. This is because I've added the commands \endfirsthead and \endhead to my table, where everything written before *\endfirsthead* declares the header for the first page the table occurs on and everything between *\endfirsthead* and *\endhead* denotes the header, which should be repeated on every following page. If you don't want a header on the following pages, you can simply remove everything in between *\endfirsthead* and *\endhead* including the *\endhead* command.

Landscape tables

Now that we have a solution for too many rows, we could also be facing the same problem if we had too many columns. If we add too many columns, we might be getting a table that's too wide for the page. In this situation, it's often best to simply rotate the table and print it in sideways. While there are many different ways to rotate the table, the only that I've found to be satisfying was using the *rotating* package.

First, we include the required package in our preamble:

```
%...

\usepackage{rotating} % To display tables in landscape

\begin{document}

%...
```

This package provides the *sidewaystable* environment, which is very easy to use. Just replace the *table* environment with the *sidewaystable* environment like this:

```
%...


\begin{sidewaystable}[h!] % <--
  \begin{center}
  \caption{Landscape table.}
  \label{tab:table1}
  \begin{tabular}{l|S|r}
        \toprule
        \textbf{Value 1} & \textbf{Value 2} & \textbf{Value 3}\\
        $\alpha$ & $\beta$ & $\gamma$ \\
    \midrule
    1 & 1110.1 & a\\
    2 & 10.1 & b\\
    3 & 23.113231 & c\\
    \bottomrule
  \end{tabular}
  \end{center}
\end{sidewaystable}


%...
```

This will automatically rotate the table for us, so it can be read when flipping the page sideways:

Table 10: Landscape table.

Value 1	**Value 2**	**Value 3**
α	β	γ
1	1110.10	a
2	10.10	b
3	23.11	c

Tables from Excel (.csv) to LaTeX

There are two disadvantages of writing tables by hand as described in this tutorial. While it works for small tables similar to the one in our example, it can take a long time to enter a large amount of data by hand. Most of the time the data will be collected in form of a spreadsheet and we don't want to enter the data twice. Furthermore once put into LaTeX tables, the data cannot be plotted anymore and is not in a useful form in general. For this reason, the next lesson shows you, how to generate tables from .csv files (which can be exported from Excel and other tools) automatically.

Summary

- LaTeX offers the *table* and *tabular* environment for table creation
- The *table environment* acts like a *wrapper* for the *tabular* similar to the *figure environment*
- Alignment and vertical separators are *passed as an argument* to the tabular environment (e.g. *\begin{tabular}{l|c||r}*)
- It's possible to align the content left (l), centered (c) and right (r), where the number of alignment operators has to match the desired number of columns
- The columns can be separated by adding | in between the alignment operators
- Rows can be separated using the *\hline* command and columns using the ampersand & symbol
- The *newline* \\ operator indicates the end of a row
- It's possible to refer to tables using *\ref* and *\label*
- Align numbers at the decimal point using the *siunitx* package
- Combine multiple rows and columns with the *multirow* package
- Prettify your tables using the *booktabs* package
- Make your tables span multiple pages with the *longtable* package
- Display your tables in landscape using the *rotating* package

Next Lesson: 10 Pgfplotstable

10. Tables from .csv in LaTeX with pgfplotstable

Use the package pgfplotstable to read tables from .csv files automatically and add them to your document.

Syntax and usage of pgfplotstable

When writing papers, it is sometimes necessary to present a large amount of data in tables. Writing such tables in LaTeX by hand is a very time-consuming and error-prone task. To avoid this, we can simply import the data directly from .csv (comma-separated value) files. Programs such as Excel, OpenOffice Calc or even emacs org-mode can export data sheets as .csv files. LaTeX can not work with them directly, but we can use the following code, to generate tables from .csv files:

```
\documentclass{article}

\usepackage{booktabs} % For \toprule, \midrule and \bottomrule
\usepackage{siunitx} % Formats the units and values
\usepackage{pgfplotstable} % Generates table from .csv

% Setup siunitx:
\sisetup{
  round-mode          = places, % Rounds numbers
  round-precision     = 2, % to 2 places
}

\begin{document}

\begin{table}[h!]
  \begin{center}
    \caption{Autogenerated table from .csv file.}
    \label{table1}
    \pgfplotstabletypeset[
      multicolumn names, % allows to have multicolumn names
      col sep=comma, % the separator in our .csv file
      display columns/0/.style={
                  column name=$Value 1$, % name of first column
                  column type={S},string type},  % use siunitx for formatting
      display columns/1/.style={
                  column name=$Value 2$,
                  column type={S},string type},
      every head row/.style={
                  before row={\toprule}, % have a rule at top
                  after row={
                            \si{\ampere} & \si{\volt}\\ % the units separated by &
                            \midrule} % rule under units
                            },
                  every last row/.style={after row=\bottomrule}, % rule at bottom
    ]{table.csv} % filename/path to file
  \end{center}
\end{table}

\end{document}
```

This will generate the following output:

Table 1: Autogenerated table from .csv file.

*Value*1	*Value*2
A	V
1	2
11.43	2342.23

This is a fairly long snippet just to include a table, but don't worry, we can just reuse it and don't have to code it every time again. We only have to change this part:

```
...
 display columns/0/.style={
                column name=$Value 1$,
                column type={S},string type},
     display columns/1/.style={
                column name=$Value 2$,
                column type={S},string type},
     every head row/.style={
                before row={\toprule},
                after row={
                        \si{\ampere} & \si{\volt}\\
                        \midrule}
                        },
                every last row/.style={after row=\bottomrule},
    ]{table.csv} % filename/path to file
...
```

Adding new columns

The part that controls our column names and formatting is:

```
...
     display columns/1/.style={
                column name=$Value 2$,
                column type={S},string type},
...
```

The display columns/NUM/ lets us choose the name for a certain column. Note that numbering starts at zero. In order to add a new column, we simply copy the whole block and change the number to:

```
...
     display columns/2/.style={
                column name=$Value 3$,
                column type={S},string type},
...
```

That's all we have to do. Well almost, we also have to add a new unit for this column, which happens here:...

```
...
                after row={
                        \si{\ampere} & \si{\volt}\\
                        \midrule}
                        },
...
```

We just put a new ampersand & and use the siunitx package to choose a new unit like so:

```
...
                after row={
                        \si{\ampere} & \si{\volt} & \si{\tesla}\\
                        \midrule}
                        },
...
```

In order for this to work, we must make sure that we export the .csv file in the correct format. It must have comma as the column separator and newline as the row separator. We can also change the respective separators in our code above, but I recommend to use this layout to avoid problems. The raw .csv datafile for this example looks like:

The layout of a .csv file

```
column 1,column 2
1,2
11.432,2342.23123123
```

As you can see, the numbers contained are actually longer than what shows up in the table. This is because we have:

```
\sisetup{
  round-mode          = places, % Rounds numbers
  round-precision     = 2, % to 2 places
}
```

in the beginning of our document. Changing these values, will change all units formatted by siunitx automatically, which includes our table. It's pretty straightforward to add a table, this snippet works for all tables, unless they are too long to fit on one page. For this purpose, I will introduce a new command in a different tutorial.

Summary

- LaTeX can generate tables from .csv files automatically
- Copy and paste the above snippet and packages to get it to work hassle free
- To add new columns, simply duplicate the *display column* line and change the number and name
- Add new units using the \siunitx command and the ampersand separator as shown above
- Have a .csv file separated with comma as column separator and newline as row separator
- Does only work for tables smaller than one page

Next Lesson: 11 Pgfplots

11. Plots in LaTeX - Visualize data with pgfplots

The pgfplots package from tikz/pgf enables you to plot data directly from .csv files in LaTeX.

Visualizing your data is easily done with auto-generated plots using the pgfplots package. It's strongly recommended to read Lesson 9 before, since I will omit some basic statements about .csv files and basically use the same file for my plot.

Basic plotting

To plot our data, we will use the following code:

```
\documentclass{article}

\usepackage{siunitx}
\usepackage{tikz} % To generate the plot from csv
\usepackage{pgfplots}

\pgfplotsset{compat=newest} % Allows to place the legend below plot
\usepgfplotslibrary{units} % Allows to enter the units nicely

\sisetup{
  round-mode          = places,
  round-precision     = 2,
}

\begin{document}

\begin{figure}[h!]
  \begin{center}
    \begin{tikzpicture}
      \begin{axis}[
          width=\linewidth, % Scale the plot to \linewidth
          grid=major, % Display a grid
          grid style={dashed,gray!30}, % Set the style
          xlabel=X Axis $U$, % Set the labels
          ylabel=Y Axis $I$,
          x unit=\si{\volt}, % Set the respective units
          y unit=\si{\ampere},
          legend style={at={(0.5,-0.2)},anchor=north}, % Put the legend below the plot
          x tick label style={rotate=90,anchor=east} % Display labels sideways
        ]
        \addplot
        % add a plot from table; you select the columns by using the actual name in
        % the .csv file (on top)
        table[x=column 1,y=column 2,col sep=comma] {table.csv};
        \legend{Plot}
      \end{axis}
    \end{tikzpicture}
    \caption{My first autogenerated plot.}
  \end{center}
\end{figure}

\end{document}
```

This plot will show up in our .pdf file after compilation:

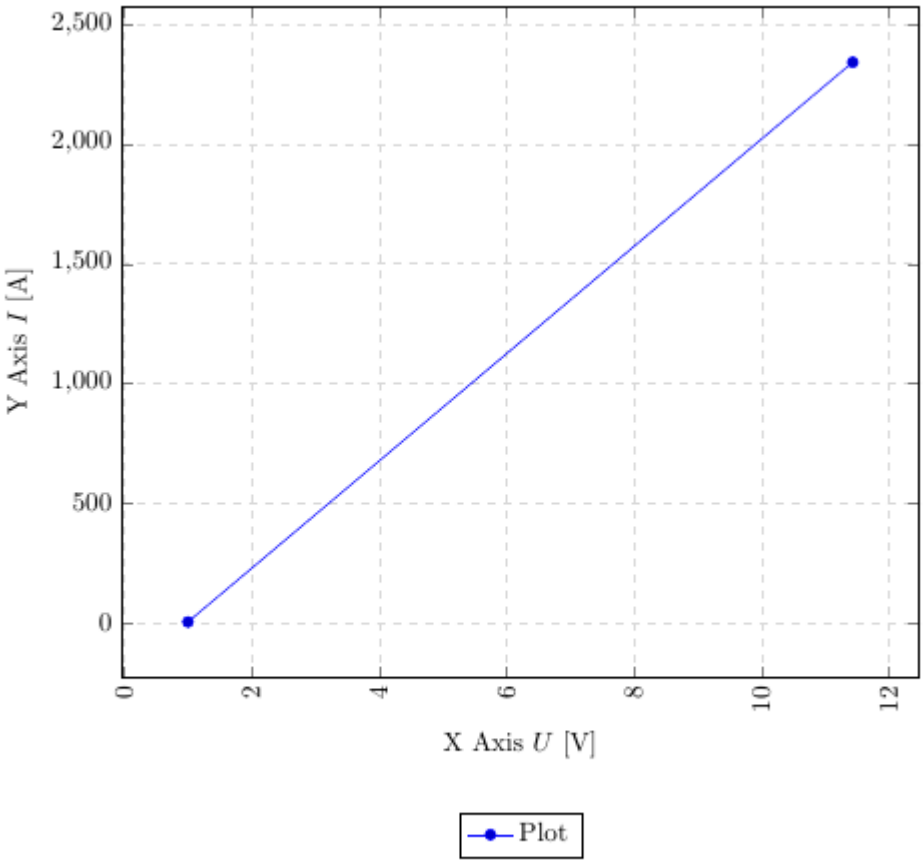


Figure 1: My first autogenerated plot.

Packages and setup

We added the new packages and the following code to generate the plot:

```
...
\usepackage{tikz}
\usepackage{pgfplots}
...
\pgfplotsset{compat=newest}
\usepgfplotslibrary{units}
...
\begin{figure}[h!]
  \begin{center}
    \begin{tikzpicture}
      \begin{axis}[
          width=\linewidth, % Scale the plot to \linewidth
          grid=major,
          grid style={dashed,gray!30},
          xlabel=X Axis $U$, % Set the labels
          ylabel=Y Axis $I$,
          x unit=\si{\volt}, % Set the respective units
          y unit=\si{\ampere},
          legend style={at={(0.5,-0.2)},anchor=north},
          x tick label style={rotate=90,anchor=east}
        ]
        \addplot
        % add a plot from table; you select the columns by using the actual name in
        % the .csv file (on top)
        table[x=column 1,y=column 2,col sep=comma] {table.csv};
        \legend{Plot}
      \end{axis}
    \end{tikzpicture}
    \caption{My first autogenerated plot.}
  \end{center}
\end{figure}
...
```

The first part only includes the necessary packages, the second part has only two commands as well, where \pgfplotsset{compat=newest} disables the backward compatibility for pgfplots, so we can place the legend of our graph below the plot and \usepgfplotslibrary{units} adds two new commands (x unit and y unit), which allows for nice formatting of units in brackets. Most parts from the last part should be self-explanatory. We have width, xlabel, ylabel and more. You should comment them out and explore them on your own.

The most important part is:

```
...
        table[x=column 1,y=column 2,col sep=comma] {table.csv};
...
```

Given a .csv file like:

```
column 1,column 2
1,2
11.432,2342.23123123
```

we have to put the name of one column for our x, in this case x=column 1 and a second column for our y, since there are only two columns, we choose y=column 2. Again, the col sep=comma indicates that we use comma as our column separator. We can copy the whole snippet as shown above and use it over and over again, we only have to change the columns we want to plot and the filename. There are a lot of options to style the plots and have bar charts and so forth. I will show a few more styles in a later tutorial or add an overview of snippets to the misc section.

Summary

- Plotting is easy with pgfplots, it plots data directly from .csv files
- Select a column by the actual name from the .csv file using *table[x=column 1,y=column 2…*

Next lesson: 12 Tikz

12. Draw pictures in LaTeX - With tikz/pgf

The pgf/tikz package allows you to draw pictures from within LaTeX document to keep the style consistent throughout your document.

The package pgf/tikz can be used to create beautiful graphics, especially diagrams in LaTeX. It enables you to create vector graphics from within your document, without the need of external tools such as Inkscape or Adobe Illustrator and thus is much more flexible. But unlike the graphical editors, we don't actually draw, but program the vector graphics using predefined macros. I will first show a little example using basic geometric elements and then explain the syntax of tikz.

A basic example

```
\documentclass{article}


\usepackage{tikz}


\begin{document}
\begin{figure}[h!]
  \begin{center}
    \begin{tikzpicture}
      \draw [red,dashed] (-2.5,2.5) rectangle (-1.5,1.5) node [black,below] {Start}; % Draws a rectangle
      \draw [thick] (-2,2) % Draws a line
      to [out=10,in=190] (2,2)
      to [out=10,in=90] (6,0)
      to [out=-90,in=30] (-2,-2);
      \draw [fill] (5,0.1) rectangle (7,-0.1) node [black,right] {Obstacle}; % Draws another rectangle
      \draw [red,fill] (-2,-2) circle [radius=0.2] node [black,below=4] {Point of interest}; % Draws a circle
    \end{tikzpicture}
    \caption{Example graphic made with tikz.}
  \end{center}
\end{figure}
\end{document}
```

The result looks like this:

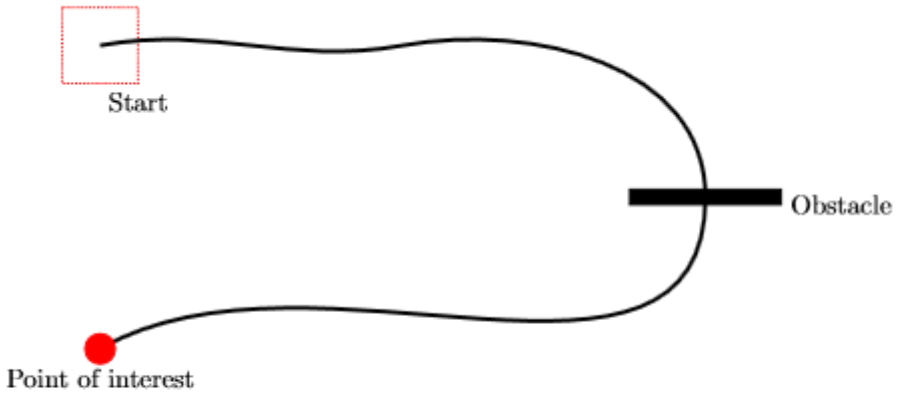


Figure 1: Example graphic made with tikz.

We usually want to embed our *tikzpicture* in a figure environment. By using the *\draw* command, we can add elements. It's necessary to specify options in brackets and the coordinates of the element.

The syntax of Tikz

To demonstrate the basic syntax, we will have a closer look at the line that draws the red rectangle in our picture:

```
\draw [red,dashed] (-2.5,2.5) rectangle (-1.5,1.5) node [black,below] {Start};
```

The options here indicate that the rectangle should be red and dashed, no magic here. Then we define the location of our *rectangle* (-2.5,2.5) in cartesian coordinates. To set the size of the rectangle, we simply add a second pair of coordinates behind our *rectangle* statement. (-1.5,1.5). As you can see, there is also a text below our rectangle. Text is usually placed as a so-called *node* (text-node) in our picture. We define the text to have black color and set the position to below our rectangle. Tikz can usually figure out the size of the rectangle automatically. Other than in plain LaTeX we have to terminate each command with a semicolon inside of the tikzpicture environment.

The next snippet shows how to draw a line by specifying a path in Tikz:

```
\draw [thick] (-2,2) % Draws a line
to [out=10,in=190] (2,2)
to [out=10,in=90] (6,0)
to [out=-90,in=30] (-2,-2);
```

The syntax is the same as for the rectangle, we only use a different command in between the coordinates. The *to* command is used to specify a point of the path. Tikz usually draws straight lines between the points, in the picture below we can clearly see the lines are bent. This can be accomplished by setting the *in* and *out* angle in brackets. There are many more options available and mastering tikz can take a vast amount of time. Yet it is very easy to accomplish basic drawings very fast and they integrate very smoothly with the rest of your document. There are also many extensions for tikz, which provide easy ways to create diagrams, flowcharts, mindmaps and many more.

Summary

- Tikz can be used to draw graphics from within the LaTeX document
- No graphical user interface, instead pictures are programmed
- Extensive documentation available in the Tikz manual
- Many examples can be found on the internet

- Extensions available to easy creation of diagrams, flowcharts and so forth

Next lesson: 13 Listings

13. Highlight source code in LaTeX with the Istlisting package.

The listing package offers source code highlighting for your various languages. Learn by example how to use it in your LaTeX documents.

The listings package is a powerful way to get nice source code highlighting in LaTeX. It's fairly easy to use and there's good documentation available on how to use it.

Example

The output of the listings package will pretty much look like this after some setup:

```
#!/usr/bin/perl
print S(@ARGV);sub S{$r=(@_[0]%4==0&&@_[0]%100!=0)||@_[0]%400=0;}
```

```
\documentclass{article}

\usepackage{listings}
\usepackage{color}

\renewcommand\lstlistingname{Quelltext} % Change language of section name

\lstset{ % General setup for the package
        language=Perl,
        basicstyle=\small\sffamily,
        numbers=left,
        numberstyle=\tiny,
        frame=tb,
        tabsize=4,
        columns=fixed,
        showstringspaces=false,
        showtabs=false,
        keepspaces,
        commentstyle=\color{red},
        keywordstyle=\color{blue}
}

\begin{document}
\begin{lstlisting}
#!/usr/bin/perl
print S(@ARGV);sub S{$r=(@_[0]%4==0&&@_[0]%100!=0)||@_[0]%400=0;}
\end{lstlisting}
\end{document}
```

I first use the include the *color* and *listings* package and then set up the language of the package headings to German using *\renewcommand\lstlistingname{Quelltext}*. This is not necessary if you're planning to use it in English.

Afterwards I set up the general layout for the package with the *\lstset*command. The options I set there should be self-explanatory. Note that it's required to manually set the colors for keywords and comments, otherwise the output would be only black on white. The desired output must then be embedded within a listings environment.

Assuming we have a Perl script saved in a file script.pl, we could also simply use the following syntax to get the same result:

```
\lstinputlisting{script.pl}
```

This will keep your LaTeX source clean and you can still use all features of the package.

Summary

- After some initial setup, all source code can be embedded in a lstlistings environment
- A list of all languages and more documentation is available in the manual of the listings package
- Use the \lstlinputlisting{FILENAME} command to read the content of source files directly into a lstlistings environment.

Next lesson: 14 Circuitikz

14. Circuit diagrams in LaTeX - Using Circuitikz.

Add neat circuit diagrams to your paper with circuitikz, extending tikz with electric components.

The listings package is a powerful way to get nice source code highlighting in LaTeX. It's fairly easy to use and there's good documentation available on how to use it.

While Tikz offers many features and packages to create diagrams and all sorts of other drawings, it unfortunately lacks a good package to layout electric circuits. Using the package *circuitikz* we can easily solve this problem. It extends provides a new environment *circuitikz* in which we can easily draw our circuits in no time. The syntax is exactly the same as shown in the previous lesson, so we can directly start with a simple code example:

```
\documentclass{article}


\usepackage{tikz}
\usepackage{circuitikz}

\begin{document}

\begin{figure}[h!]
  \begin{center}
    \begin{circuitikz}
      \draw (0,0)
      to[V,v=$U_q$] (0,2) % The voltage source
      to[short] (2,2)
      to[R=$R_1$] (2,0) % The resistor
      to[short] (0,0);
    \end{circuitikz}
    \caption{My first circuit.}
  \end{center}
\end{figure}

\end{document}
```

This will create the following circuit diagram in our document:

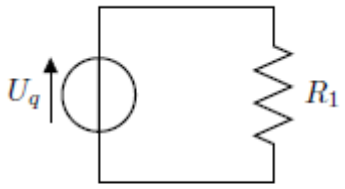


Figure 1: My first circuit.

The circuit will be drawn in the same way as a path in tikz, but we specify special options for the elements:

```
\draw (0,0)
      to[V,v=$U_q$] (0,2) % The voltage source
```

Starting at (0,0) we will draw a voltage source specifying the [V,v=U_q] options to the coordinates (0,2), where *V* chooses the symbol for a voltage source and the v=U_q draws the voltage arrow next to it. Then we proceed to the resistor:

```
      to[short] (2,2)
      to[R=$R_1$] (2,0) % The resistor
```

We first must draw a short circuit from (0,2) to (2,2) and then put the resistor symbol on the path from (2,2) to (2,0) note that this time the label of the element must be specified directly (*R*=R1R1).

A list of all available elements for circuits is available in the circuitikz manual.

But how can we add more elements to the circuit? Let's say we want to add an inductor parallel to the Resistor. The easiest way is to add a new draw command like this:

```
    \begin{circuitikz}
      \draw (0,0)
      to[V,v=$U_q$] (0,2) % The voltage source
      to[short] (2,2)
      to[R=$R_1$] (2,0) % The resistor
      to[short] (0,0);
      \draw (2,2)
      to[short] (4,2)
      to[L=$L_1$] (4,0)
      to[short] (2,0);
   \end{circuitikz}
```

After compilation we'd get the following circuit diagram:

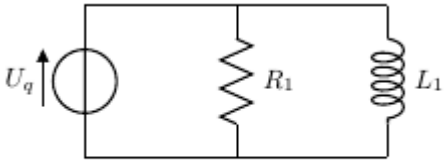


Figure 1: My first circuit.

Adding a capacitor next to it, is just as simple:

```
\begin{circuitikz}
  \draw (0,0)
  to[V,v=$U_q$] (0,2) % The voltage source
  to[short] (2,2)
  to[R=$R_1$] (2,0) % The resistor
  to[short] (0,0);
  \draw (2,2)
  to[short] (4,2)
  to[L=$L_1$] (4,0)
  to[short] (2,0);
  \draw (4,2)
  to[short] (6,2)
  to[C=$C_1$] (6,0)
  to[short] (4,0);
\end{circuitikz}
```

This would give us the following diagram:

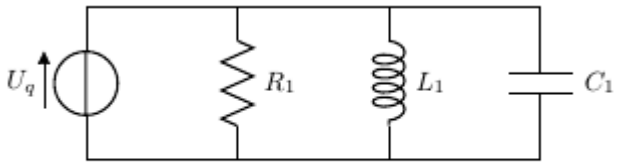


Figure 1: My first circuit.

The circuitikz manual provides examples of all symbols and functions and can also be used for further reference.

Summary

- Circuitikz provides an environment to draw electric circuit diagrams
- The syntax is similar to the plain Tikz syntax
- A list of all symbols is available in the circutikz manual

Next lesson: 15 More Circuitikz

15. Circuit diagrams in LaTeX - advanced features

Learn how to use most features of circuitikz with many examples and explanations.

Introduction

As we learned in lesson 12, circuitikz is a powerful tool for circuit creation. But there are some pitfalls along the way. The circuitikz manual provides a nice reference for all components available but lacks an in-depth explanation on how to use every element. Basically, there are three important types of components available in circuitikz, which are namely monopoles, bipoles and tripoles. The other components mentioned in the manual can be used in a way similar to the classes mentioned in this tutorial.

Lines

Before I step into the various components, let me show you how to change the appearance of the basic lines in circuitikz. Consider the following example:

```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (-1,0) to[short,o-o] (1,0);
\end{circuitikz}
\end{figure}
```

As you can see, the syntax changed a little from the previous lesson. The first line is pretty much the same. The first line will draw the upper line segment with an open connector at each end. The connectors can be selected by changing the o-o part in the code.

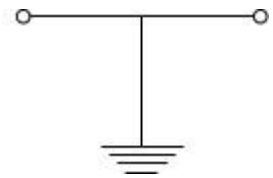
If we'd like to change the connectors, consider the following example, which will lead to this output:

```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (-1,0) to[short,*-] (1,0);
\end{circuitikz}
\end{figure}
```

The notation for connectors is pretty straightforward. You can choose the type of connector at each end of the line by using either the * or o symbol. Besides you can select that only one end or both ends of the line should have this kind of symbol e.g. o-, -o or o-o. If you don't want a connector at all, simply write to[short] without any additional symbols.

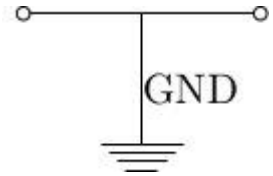
Monopoles

I will start off with monopoles, which are about the most basic component of circuitkz. This class contains symbols such as ground nodes and antennas. So let's take a closer look at an actual example.



```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (-1,0) to[short,o-o] (1,0);
  \draw (0,0) to[short] node[ground] {} (0,-1);
\end{circuitikz}
\end{figure}
```

I took the line from the first section but added a ground node to it. As you can see, the line differs in that the component is not specified as a bipole using the to operator, but as a node of type ground (node[ground]). It is imperative, that every node has a label in tikz. This label can be left empty, but the syntax should must be node[options] {}. If we decided to name our node, we could just write the text in between the braces such as




```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (-1,0) to[short,o-o] (1,0);
  \draw (0,0) to[short] node[ground] {GND} (0,-1);
\end{circuitikz}
\end{figure}
```

That's all there is to monopoles in circuitikz.

Bipoles

I've already covered bipoles in lesson 12, but I want to explain some more advanced features here. Most of the time, we want to have arrows to indicate current or voltages in our circuits. Circuitikz provides easy to use options to add these to your circuits. The examples here were basically taken from the circuitikz manual, so you can find some more examples in there, but I thought they shouldn't be missing in my tutorial, since this is a very important feature.

Current arrows

The most basic way to add a current arrow to your bipole is by specifying the i option for the respective component. By default, the direction of the arrow and the position of the label are determined by circuitikz, but you can override those settings to enhance readability.

```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,i=$i_1$] (2,0);
\end{circuitikz}
\end{figure}
```

If you wanted to change the direction of the arrow, you could simply use the following code instead. The < or > operator can be used to indicate the direction of the arrow.

```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,i<=$i_1$] (2,0);
\end{circuitikz}
\end{figure}
```

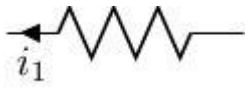
Furthermore we can manipulate the position of the label i_1 by using either the ^ or _ operator, which stands for above and below the line. First look at the result for the first operator:

```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,i^=$i_1$] (2,0);
\end{circuitikz}
\end{figure}
```

Now we want to place the label below:

```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,i_=$i_1$] (2,0);
\end{circuitikz}
\end{figure}
```

The two kinds of operators can be combined to change the position of the arrow to either the left or right hand side of the bipole. To place the arrow on the left hand side, simply write



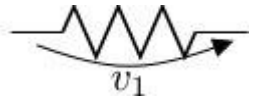
```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,i<_=$i_1$] (2,0);
\end{circuitikz}
\end{figure}
```

and change the direction to place it on the right hand side like this


```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,i_>=$i_1$] (2,0);
\end{circuitikz}
\end{figure}
```

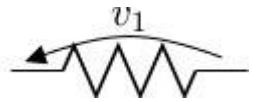
Voltage arrows

The voltage arrows can be used in just the same manner as the current arrows, but this time, use the v option instead.




```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,v_>=$v_1$] (2,0);
\end{circuitikz}
\end{figure}
```

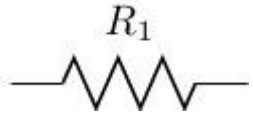
As you can see, the same operators apply to manipulate position and direction of the arrow




```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,v^<=$v_1$] (2,0);
\end{circuitikz}
\end{figure}
```

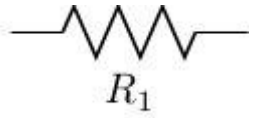
Labels

Of course it's possible to add a label for the element itself as well. We do that by using the l option this time.




```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,l^=$R_1$] (2,0);
\end{circuitikz}
\end{figure}
```

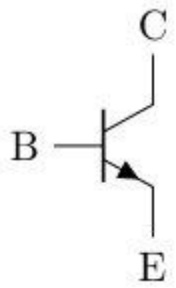
We might as well place the label below the element as well, again using the _ operator:




```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) to[R,l_=$R_1$] (2,0);
\end{circuitikz}
\end{figure}
```

Tripoles

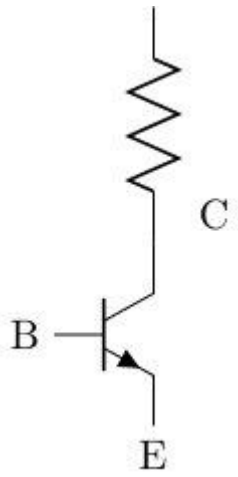
The last but not least group of components are tripoles. The most important group of tripoles are of course all kinds of transistors. You can find a whole list of examples for transistors along with this one in the circuitikz manual, but let me point out how the usage is different from bipoles.




```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) node[npn](npn1) {}
  (npn1.base) node[anchor=east] {B}
  (npn1.collector) node[anchor=south] {C}
  (npn1.emitter) node[anchor=north] {E};
\end{circuitikz}
\end{figure}
```

Tripoles are nodes in circuitikz just like monopoles, but you have to place several anchors to the respective connectors of the tripole. First you specify a name for the node in parenthesis. In this case I've chosen the name npn1. This name will be used to reference to our transistor later on. Afterwards, the three anchors must be set as seen above.

We can now refer to the nodes and attach some other elements to the transistor.




```
\begin{figure}[h!]
\begin{circuitikz}
  \draw (0,0) node[npn](npn1) {}
  (npn1.base) node[anchor=east] {B}
  (npn1.collector) node[anchor=south,xshift=0.5cm] {C}
  (npn1.emitter) node[anchor=north] {E};
  \draw (npn1.collector) to[R] ++(0,2);
\end{circuitikz}
\end{figure}
```

Note that I added an xshift option to the node, to prevent the label from overlapping with the new element. I hope you enjoyed my explanations and that you find using circuitikz a lot less confusing from now on.

Summary

- There are three main classes of components in circuitikz
- All other classes can be used in a way similar to those main classes
- Usage examples are provided in the circuitikz manual

Next lesson: 16 Hyperlinks

16. How to make clickable links in LaTeX

Setting hyperlinks in LaTeX is easy with the hyperref package. Link to any website or add your email address to any document.

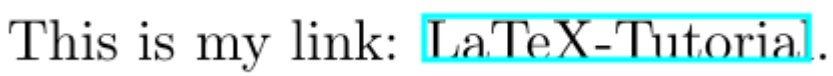
Adding clickable links to LaTeX documents is very straightforward, you only have to add the *hyperref* package to your preamble. This package allows you to set links with a description as well as add bare urls to your document.

```
\documentclass{article} % or any other documentclass

%...

\usepackage{hyperref}

%...

\begin{document}

%...

\end{document}
```

After setting this up, you're ready to go and add links anywhere to your document. In order to add a link with a description (i.e. making a word clickable), you should use the *href* command like so:

```
%...

\begin{document}

This is my link: \href{http://www.latex-tutorial.com}{LaTeX-Tutorial}.

\end{document}
```

This will lead to the following output in your PDF:

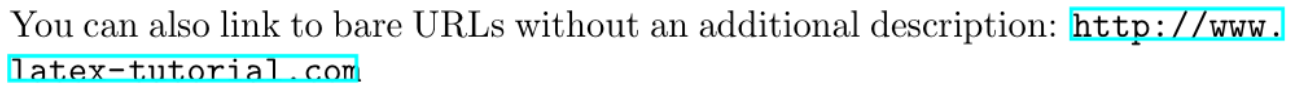


You will notice, that there's a colored box shown around the word. Don't worry, this box is *not* going to show up in your printed document, but only if you view it on your computer.

If you simply want to embed a bare URL, you should use the *url* command instead, which usage is even simpler:

```
%...


\begin{document}


You can also link to bare URLs without an additional description:
\url{http://www.latex-tutorial.com}


\end{document}
```

This will show the following clickable link in your PDF:



If you want to add your *email address* to your document, so that it automatically opens your readers email program whenever they click on it, you can also use the url package like so:

```
%...


\begin{document}


My email address is: \href{mailto:claudio.vellage@latex-
tutorial.com}{claudio.vellage@latex-tutorial.com}


\end{document}
```

Usually, using the default settings and color etc are just fine, but these can also be customized if you want to. This can be done using the *hypersetup* command in your preamble. Since I've never used this feature in any document, I won't explain how to use it, but you can find a more detailed documentation of the hyperref package here, if you're curious.

Summary

- Add the *hyperref* package to your preamble
- Links will show up in a colored box which will be *invisible* when you print it.
- Use *\href{URL}{DESCRIPTION}* to add a link with description
- Use *\url{URL}* to add a link without a description
- Prepend your email address with *mailto:* to make it clickable and open your mail program.
- In case you want to customize the appearance, read the documentation on ctan

Next lesson: 17 Lists

17. LaTeX list - Enumerate and Itemize

Learn how to use the enumerate and itemize environments to add ordered, unordered and nested lists to your document.

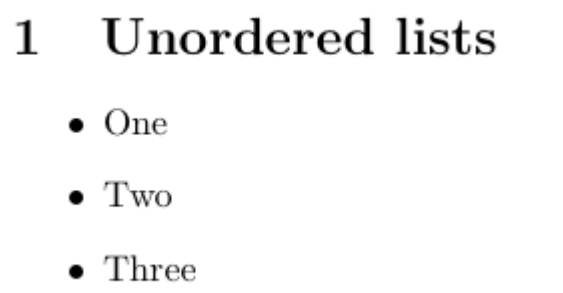
Using lists in LaTeX is pretty straightforward and doesn't require you do add any additional packages. For unordered lists, LaTeX provides the *itemize* environment and for ordered lists there is the *enumerate* environment. The elements within both environments have to be declared beginning with the *\item* command. The following code examples show how to use the most common types of lists you're going to use in your document.

Unordered lists

As I've mentioned above, unordered lists use the *itemize* environment and works without any additonal packages:

```
\begin{itemize}
        \item One
        \item Two
        \item Three
\end{itemize}
```

This will generate the following output:

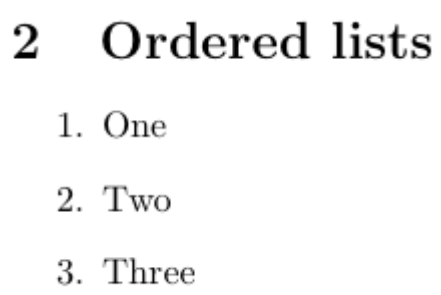


Ordered lists

If you want to add an ordered list, you simply have to replace *itemize* with *enumerated* environment and LaTeX will take care of the enumeration for you:

```
\begin{enumerate}
        \item One
        \item Two
        \item Three
\end{enumerate}
```

As you can see, LaTeX will automatically get the numbers right:

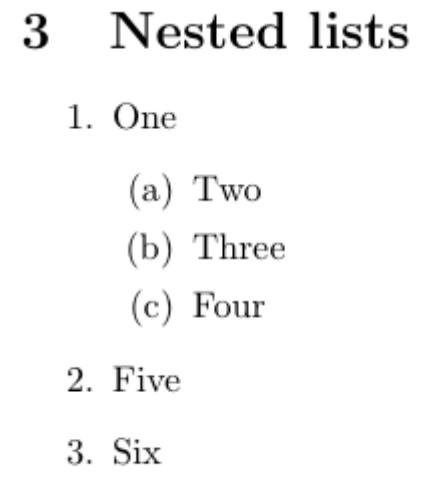


Nested lists

Sometimes you also have to list things, which have some kind of sub-category. For this reason, LaTeX allows you to nest list environments and it will fix the indentation and numbering accordingly.

```
\begin{enumerate}
        \item One
    \begin{enumerate}
        \item Two
        \item Three
        \item Four
    \end{enumerate}
    \item Five
    \item Six
\end{enumerate}
```

The output will be formatted like this:



Changing the numbering / bullets

Sometimes it's necessary to change the numbering scheme of a list, e.g. you want to use a different symbol and so forth. You can easily modify the output of the list.

Unordered lists

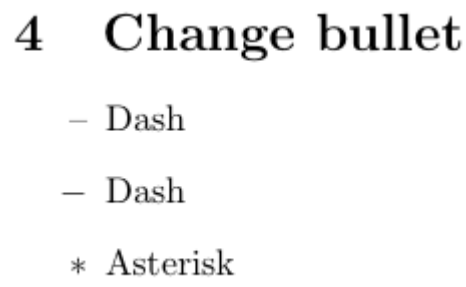
You can make the following changes easily without loading a package:

```
%From bullet to dash
\item[--] or \item[$-$]


% From bullet to asterisk
\item[$\ast$]


%Use any math character
\item[$\CHARACTER$]
```

A full working code could look like this:

```
\begin{itemize}
        \item[--] Dash
    \item[$-$] Dash
    \item[$\ast$] Asterisk
\end{itemize}
```

And the output will look as follows:



If you want to change the symbol for all items of the list, you should preferably use the *enumitem* environment, which I will explain using the example of ordered lists.

Ordered lists

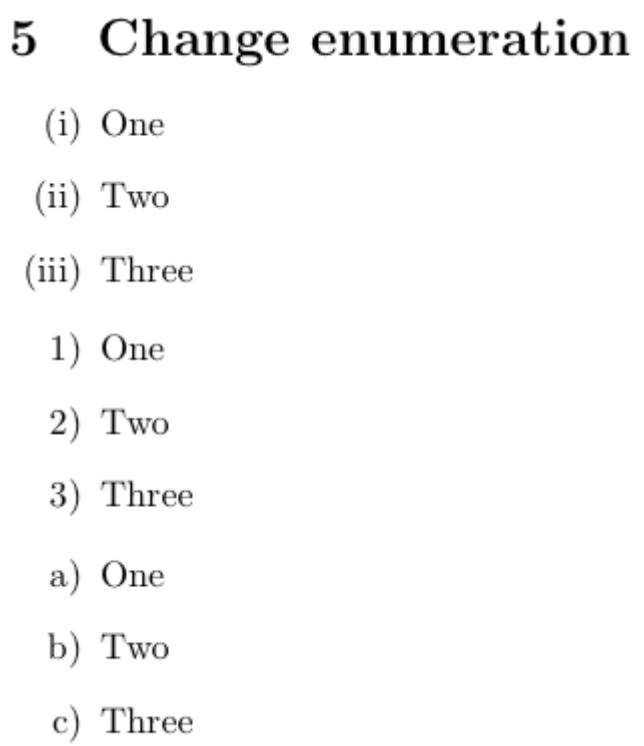
Changing this environment is a little more tricky, because there's a lot more logic involved and the easiest solution is probably using the *enumerate* or *enumitem* environments. I will use the enumerate environment for this purpose. So I will first add this environment to my preamble:

```
\documentclass{article}


% ...


\usepackage{enumitem}


\begin{document}
```

We can now use the following options on the *enumerate* environment:

```
%Roman numbers
\begin{enumerate}[label=(\roman*)]
%...

% Arabic numbers
\begin{enumerate}[label=\arabic*)]
%...

% Alphabetical
\begin{enumerate}[label=\alph*)]
%...
```

The output will look like this:



You can likewise use this to change the symbol of unordered lists:

```
\begin{itemize}[label=$\ast$]
        \item One
    \item Two
    \item Three
\end{itemize}
```

Which will consistently change the symbol of all items:

* One
* Two
* Three

Summary

- Unordered lists can be created using the *itemize* environment.
- Ordered lists can be created using the *enumerate* environment.
- Lists can be nested and will be aligned and enumerated properly.
- Use the *enumitem* package to customize the symbols or enumeration.