Ein grundlegender Einblick in LaTeX

Ein (hoffentlich) einfacher Einstieg

Inhaltsübersicht

Dieses Dokument erhebt den Anspruch einen grundlegend-tiefblickenden Einblick in die ruhigen Oberflächen der ach so tiefen LaTeX-Gewässer zu vermitteln. Es wird im Rahmen des EidI-Tutoriums im Wintersemester 2019/2020 ausgebaut und (hoffentlich) verfeinert. Bei der Erstellung dieses Dokuments kam keine Ente zu schaden.

1 LaTeX herunterladen

1.1 Linux

Auf Linux lässt sich LaTeX einfach über das Paket `texlive` installieren. Dies ist zwar nicht immer *aktuell*, wird aber dafür vollautomatisch für uns eingerichtet. So genügt auf einem `apt`-basierten System zum Beispiel:

```
sudo apt install textlive-full
```

Anschließend können die Dokumente mittels des Terminals durch `pdflatex ⟨Dokumentname⟩` kompiliert werden.
Wer immer die aktuelleste Version möchte, kann auch hier nachsehen `https://www.tug.org/texlive/tlmgr.html`, die Installation ist nicht zwangsläufig aufwändiger.

1.2 Windoof

Auch wenn es `texlive` ebenfalls für Linux gibt, so empfiehlt sich das Verwenden von `MikTeX`. Es lässt sich hier herunterladen: `https://miktex.org/download`. Um nun mit LaTeX arbeiten zu können benötigt man ein weiteres Programm, wobei es hier viele Vertreter gibt. So gibt es: TeXmaker,[1] Kile[2]

[1] `https://www.xm1math.net/texmaker/download.html`
[2] `https://kile.sourceforge.io/`

und TeXstudio,[3] wobei ich letzteres empfehle. Die Programme sind allerdings alle hinsichtlich ihres Funktionsumfangs und ihrer Verwendung vergleichbar. Im Programm angekommen (für später ☺) lässt sich mittels eines Pfeils (oder in der Regel dem Tastaturkürzel `F5`) das LaTeX-Dokument, das es noch zu schreiben gibt, kompilieren und anzeigen. Ich werde im Folgenden natürlich nur auf LaTeX und nicht auf die tollen Features der jeweiligen TeX-Editoren eingehen.

1.3 Apple

Hier gilt es zu Unterscheiden, ob es sich nun um eine Tablet- oder um einen gängigen Laptop handelt. Für letzteren Fall ist wohl folgende Anleitung am ausführlichsten: `https://www.latexbuch.de/latex-apple-mac-os-x-installieren/`. Für IPad und co gibt es nun eine Reihe an mal kostenpflichtigen und mal kostenlosen Apps über deren Qualität ich nichts zu sagen vermag: TeXpad,[4] VerbTeX[5] TeX Writer.[6]

1.4 Ich mags online

Man muss LaTeX auch nicht installieren, was den Vorteil hat, dass dieser Weg auf jeder Plattform funktioniert, die es irgendwie ins Internet schafft. Nebst `https://latexbase.com/` und `https://latexonline.cc/` gilt es die bekanntesten (und mittlerweile fusionierten) zu erwähnen: `https://www.sharelatex.com/` und `https://www.overleaf.com/`.

2 Das Grundgerüst

2.1 Hello World

Jedes LaTeX-Dokument besitzt grob den folgenden Aufbau:

```
\documentclass{⟨Name der Dokumentklasse⟩}

⟨Pakete und son' Gedöns⟩

\begin{document}

⟨Das Dokument⟩

\end{document}
```

Hinweis: Texte in Spitzen Klammern (⟨···⟩) dienen einem Platzhalter und müssen in der Regel ersetzt werden. Es ist weiter bereits hier interessant zu wissen, das der Backslash eine Kontrollsequenz in Latex einleitet, die zum Beispiel die Textformatierung verändern kann. Obwohl es für die Dokumentklasse eine Vielzahl an Optionen[7] gibt, werden wir uns in diesem Beispiel an die Klasse `article` halten. Pakete, die wir generell immer brauchen sind die folgenden:

```
\usepackage[T1]{fontenc} % Kodierung des Texts
\usepackage[utf8]{inputenc} % Damit die Dateikodierung egal ist
\usepackage[ngerman]{babel} % Wortrennungen und so
```

[3] `https://www.texstudio.org/#download`
[4] `https://www.texpad.com/ios`
[5] `https://apps.apple.com/us/app/iverbtex-latex-editor/id560869163`
[6] `https://apps.apple.com/us/app/tex-writer-latex-on-the-go/id552717222`
[7] `https://de.wikibooks.org/wiki/LaTeX-Wörterbuch:_documentclass`

Wie die farbliche Hervorhebung bereits vermuten lässt, startet das Prozentzeichen einen Kommentar in LaTeX. Ein solcher dient lediglich zur Information für den/die Dokumentautor*in/ren und wird bei der Dokumenterstellung ignoriert. Dies ist, neben dem bereits bekannten Backslash (`\`) der eine Kontrollsequenz einleitet eines der wenigen Zeichen die wir in LaTeX nicht direkt verwenden können. Insgesamt sind alle folgenden Zeichen belegt:

```
# $ % ^ & _ { } ~ \
```

Möchten wir nun zum Beispiel ein `&` schreiben, so müssen wir dies durch ein `\`-„escapen“. Abgesehen vom Backslash selbst, können wir so alle reservierten Zeichen auch in LaTeX setzen:

```
\# \$ % \^{} \& \_ \{ \} \~{} \textbackslash{}
```

Dieses Prinzip sollte sich relativ schnell eingewöhnen lassen (zumal die meisten Editoren eine solche Ersetzung automatisch vornehmen oder zumindest davor warnen).

Bemerkung 1 – Und das soll ich jetzt immer kopieren?

Nö. Es gibt hier eine hoffentlich von mir stets aktuell gehaltene Datei, die entsprechend ein Grundgerüst zum kopieren liefert: `https://gist.github.com/EagleoutIce/4a1333dee677dfb6b422a7f7e79b8fe6`.

Unser erstes Hello-World-Dokument generieren wir nun also wie folgt (die Kommentare lasse ich hier zur Übersicht weg, das *oben* verlinkte Gist enthält Kommentare).

```tex
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage[ngerman]{babel}

\begin{document}
Hallo Welt
\end{document}
```

Bemerkung 2 – Leerfelder und -zeilen in LaTeX

Leerfelder (dazu zählen hier auch mal Tabulatoren) kollabieren in LaTeX stets zu einem Leerfeld. Ob wir nun also: `Hallo␣␣␣Welt` (die Leerfelder wurden hier markiert) oder `Hallo␣Welt` schreiben spielt keine Rolle. Für Leerzeilen gilt dasselbe. Wir können den Text in unserer `.tex`-Datei überall umbrechen soviel wir wollen, LaTeX wird die Zeilenenden überlesen.[8]
Leerzeilen folgen in LaTeX einem ähnlichen Muster wie Leerzeichen. Ob wir nun 1, 2 oder 300 einfügen spielt in LaTeX keine Rolle, da sie zu einer kollabieren und einen neuen *Paragraphen* anführen. Wer das jetzt noch nie gehört hat, es handelt sich, vereinfacht, um den Start einer neuen Zeile die (zur besseren Übersicht im Fließtext) ein bisschen eingerückt ist.

[8] Dies gilt nicht für Befehlssequenzen. Wir können also nicht inmitten eines Befehls wie `\documentclass` eine neue Zeile starten, dies sollte aber logisch sein und wird vom Editor auch entsprechend farbliche markiert!

2.2 Die wichtigsten Formatierungsbefehle

Da wir im Text natürlich auch mal kursiv, fett oder farbig schreiben möchten, gibt es eine Reihe an Befehle, deren Namen alle mit einem `text` angeführt werden und dann von einem englischen Kurzbezeichner gefolgt werden. Im Folgenden ein Beispiel, wobei das entstehende Dokument rechts angezeigt wird:

```
\textbf{Hallo} \textit{Welt, na}
\textsc{wie Geht es} dir denn?
```

Hallo *Welt, na* wie Geht es dir denn?

Der letzte Befehl (*small caps*) sorgt für Großbuchstaben, die allerdings auch als Kleinbuchstaben gesetzt werden können. Damit das Dokument kunterbunt werden kann, benötigen wir noch ein weiteres Paket namens `xcolor`, es erlaubt Text wie folgt farbig zu gestalten:

```
\textcolor{green}{Hallo} \textcolor{purple}{Welt, na}
```

Hallo Welt, na

Einige Farben werden vordefiniert, wie man sich eigene Erzeugen kann kommt (vermutlich) später.

2.3 Die wichtigsten Strukturierungsbefehle

Da ein Artikel für gewöhnliche weder Kapitel noch „Parts“ besitzt, stehen die Befehle `\chapter` und `\part` in der `article`-Klasse *nicht* zur Verfügung. Sie seien hier nur zur Vollständigkeit erwähnt. Es gibt allerdings genug andere Befehle mit denen wir unser Dokument strukturieren können. Die wichtigsten hiervon sind `\section` und `\subsection`, die es erlauben neue Abschnitte wie folgt zu setzen:[9]

```
\section{Ich bin eine Sektion}
% Text...
\subsection{Ich bin ein Unterabschnitt}
% Text...
\subsection{Ein weiterer Unterabschnitt!}
% Text...
\section{Ein weiterer Abschnitt}
% Text...
```

Sie werden automatisch korrekt durchnummeriert und erlauben es so wichtige Abschnitte zu Trennen. Eine Inhaltsübersicht erzeugen wir mittels

```
\tableofcontents
```

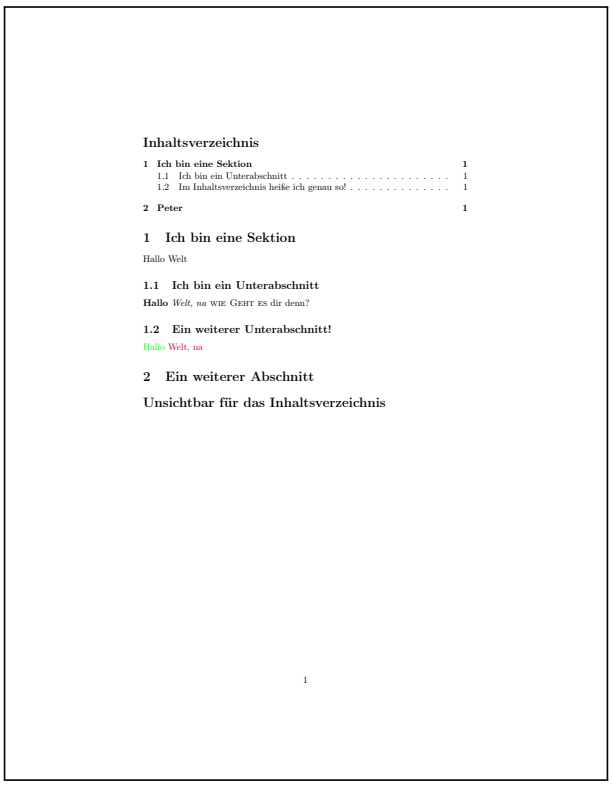
Bemerkung 3 – Hilfsdateien

Hier ist es notwendig zu erwähnen, dass LaTeX (streng genommen TeX) die Datei beim kompilieren (also Text-zu-PDF) nur einmal von oben nach unten liest. Da wir die Inhaltsübersicht ja vermutlich am Beginn des Dokuments möchten hat TeX noch gar nichts was er da reinschreiben könnte. Deswegen wird die Inhaltsübersicht beim ersten Lauf leer sein und erst beim zweiten gefüllt werden. Warum? Beim durchlaufen des Dokuments erzeugt TeX sich eine Hilfsdatei (`.toc`), in der die Vorkommnisse von `\section` usw. festgehalten werden und in der nächsten Runde neu eingelesen werden. Deswegen empfiehlt es sich das Dokument (zumindest vor der Abgabe ☺) zweimal zu kompilieren.

Es ist übrigens auch möglich die Abschnitte für die Inhaltsübersicht anders zu benennen. Wir setzen den gewünschten Text einfach in eckige Klammern. Möchten wir nicht, dass ein Abschnitt ins Inhaltsverzeichnis kommt, so fügen wir ein Sternchen an, was sie natürlich auch von der Nummerierung befreit:

[9] Sofern ich es nicht explizit anders sage, gehören alle hier gezeigten Ausschnitte in die `document`-Umgebung.

```latex
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage[ngerman]{babel}

\usepackage{xcolor}

\begin{document}

\tableofcontents

\section{Ich bin eine Sektion}
Hallo Welt

\subsection{Ich bin ein Unterabschnitt}
\textbf{Hallo} \textit{Welt, na} \textsc{wie Geht es} dir denn?

\subsection[Im Inhaltsverzeichnis heiße ich genau so!]{Ein weiterer Unterabschnitt!}
\textcolor{green}{Hallo} \textcolor{purple}{Welt, na}

\section[Peter]{Ein weiterer Abschnitt}
% Text...

\section*{Unsichtbar für das Inhaltsverzeichnis}

\end{document}
```



Inhaltsverzeichnis

1 Ich bin eine Sektion

Hallo Welt

1.1 Ich bin ein Unterabschnitt

Hallo *Welt, na* WIE GEHT ES dir denn?

1.2 Ein weiterer Unterabschnitt!

Hallo Welt, na

2 Ein weiterer Abschnitt

Unsichtbar für das Inhaltsverzeichnis

1

Das Dokument wurde für die Anzeige nicht beschnitten, allerdings skaliert. Gut zu sehen ist, dass die Abstände vor allem zum Seitenrand entsprechend groß sind. Wir wenden uns nun also der Seitengestaltung zu:

2.4 Die Dimensionen der Seite

Mittels des Pakets `geometry` lassen sich die Seitenränder hervorragend modifizieren. Wir übergeben die gewünschten Einstellungen wieder in eckigen Klammern:

```
\usepackage[left=20mm,right=20mm,top=15mm,bottom=10mm]{geometry}
```

Wir müssen nicht alle diese Optionen angeben und es gibt auch noch einige mehr die sich, in der tollen Dokumentation[10] (oder auf Linux: `texdoc geometry`) nachlesen lassen. Übrigens: das ist eine Anweisung, die nicht in `document` gehört! Generell dürfen in LaTeX Pakete nur außerhalb des Dokuments eingebunden werden!

Bemerkung 4 – Struktur eines LaTeX-Dokuments

Grob lässt sich ein jedes LaTeX-Dokument in zwei Abschnitte aufteilen:

- Die *Präambel*: Dies ist der Teil vor `\begin{document}`. Hier wird die Dokumentklasse (mittels `\documentclass`) festgelegt, es werden Pakete eingebunden (mittels `\usepackage`), Makros konfiguriert und gegebenenfalls definiert und deklariert (kommt noch ☺).
- Der *Dokumententeil*: Dies ist der Abschnitt des Dokuments, den wir durch `document`-markieren. Hier dürfen keine Pakete mehr eingebunden werden, dafür können wir allerdings Text schreiben, der dann auch so ins Dokument übernommen wird!

3 Mathe, $e^{\mathbf{i}\cdot\pi} = -1$

3.1 Einfache Formeln

Mathe kann in LaTeX nicht einfach in den Text gesetzt werden. Alle folgenden Ausdrücke können wir entweder mit `\( ... \)` oder `\[ ... \]` einrahmen. Ersteres wird den gewünschten Ausdruck wie normalen Text in die Zeile setzen: `\(2 + 3 = 4\)` ergibt: $2 + 3 = 4$, zweiteres wird den Ausdruck zentriert in einer sogenannten Display-Variante setzen (das Ergebnis ist in der Regel dasselbe allerdings können zum Beispiel Exponenten anders formatiert werden, da ja auch Vertikal mehr Platz genommen werden kann). So liefert `\[2 + 3 = 4\]` in diesem Fall:

$$2 + 3 = 4$$

Die Abstände zwischen den Symbolen sind übrigens lediglich optischer Natur. In Mathe-Umgebungen werden standardmäßig *alle* Leerfelder geschluckt und die Symbole mit vordefinierten Abständen gesetzt. Hierbei sind die meisten Formatierungsbefehle intuitiv:

- Einen *Exponent* setzen wir durch das Hütchen (`^`), Indizes durch den Unterstrich (`_`). Möchten wir einen ganzen Block Formatieren so setzen wir ihn in geschwungene Klammern, die Reihenfolge ist unerheblich:

[10] `https://ctan.space-pro.be/tex-archive/macros/latex/contrib/geometry/geometry.pdf`

```
\(a^1 + b^2 = c^2\) \\
\(a^2 + b_{144} = c^2_4\) \\
\(a_3 + b^i_{23} = c_{i_{42}}^{a^3}\)
```

$a^1 + b^2 = c^2$
$a^2 + b_{144} = c^2_4$
$a_3 + b^i_{23} = c_{i_{42}}^{a^3}$

Wie die letzte Zeile demonstriert, lassen sich diese Formatierungen auch beliebig tief verschachteln. Die hier jeweils angeführten doppelten Backslashes (`\\`) forcieren übrigens den Start einer neuen Zeile.

- *Griechische Symbole* existieren als gleichnamiger LaTeX-Befehl und können somit gesetzt werden:

```
\(e^{i * \pi} = -1\) \\
\(12 * \lambda + \xi_\Omega\) \\
\(\Psi - \Omega * \chi^\rho\)
```

$e^{i*\pi} = -1$
$12 * \lambda + \xi_\Omega$
$\Psi - \Omega * \chi^\rho$

Eine ausführliche Liste aller Symbole befindet sich im Anhang unter *Mathematische Symbole*.

- *Mathematische Operatoren* lassen sich in der Regel einfach durch ihren bekannten Kurzbezeichner setzen. Unendlich wird durch `\infty`, ein (mathematischer) Pfeil durch `\to` gekennzeichnet. Die folgende Gleichung dient als Beispiel, nicht als große Erleuchtung der Mathematik:

```
\[ \sum_{i=0}^\infty \frac{4\pi_2}{12 \pm \sqrt{4-3-2-i}} =
      \lim_{k \to \infty} \frac{\prod_{n=0}^k \sqrt{k}}{x} \]
```

Ergibt:

$$\sum_{i=0}^{\infty} \frac{4\pi_2}{12 \pm \sqrt{4-3-2-i}} = \lim_{k \to \infty} \frac{\prod_{n=0}^{k} \sqrt{k}}{x}$$

- *Klammern*, und generell die meisten „Begrenzer“, lassen sich durch ein Anführen von `\left` und `\right` an die Höhe des eingeschlossenen Inhalts anpassen:

```
\[ \forall_{x \in M} \exists! x : \left( \frac{2x}{42} \right) =
      \left|\frac{(4x)}{5}\right| \]
```

Ergibt:

$$\forall_{x \in M} \exists! x : \left(\frac{2x}{42} \right) = \left| \frac{(4x)}{5} \right|$$

Mit `\middle` können so auch noch weitere Symbole innerhalb von `\left` und `\right` angepasst werden.

- Um *Mengen* oder andere Objekte zu dotieren existieren `\mathbb` und `\mathcal`. So ergibt `\(\mathbb{N}\)`: $\mathbb{N}$ und `\(\mathcal{N}\)`: $\mathcal{N}$.

3.2 Weitere wichtige und nützliche Befehle

Erst einmal eine Ansammlung an Befehlen, die man vermutlich relativ oft brauchen wird (*Hinweis, auch wenn sie benötigt werden, sind hier die Befehle für die Mathe-Umgebungen nicht aufgeführt.*): `\cdot` ($\cdot$), `\pm` ($\pm$), `\sum` ($\sum$), `\prod` ($\prod$), `\to` ($\to$), `\infty` (∞), `\int` ($\int$), `\neq` ($\neq$), `\mapsto` ($\mapsto$), `\leq` ($\leq$) und `\geq` ($\geq$). Wie bereits aus den obigen Gleichungen ablesbar, ist es möglich mittels `\frac{⟨Zähler⟩}{⟨Nenner⟩}` einen Bruch, mit `\binom{⟨n⟩}{⟨k⟩}` einen Binomialkoeffizienten und mit `\sqrt{⟨Mathe⟩}` eine Wurzel zu setzen. Diese Befehle lassen sich beliebig verschachteln:

```
\(\frac{n!}{k!(n-k)!} = \binom{n}{k}\) \\
\(\sum_{i = 1}^n i =
   \frac{n(n-1)}{2} \cdot \sqrt{1}\)
```

$\frac{n!}{k!(n-k)!} = \binom{n}{k}$
$\sum_{i=1}^{n} i = \frac{n(n-1)}{2} \cdot \sqrt{1}$

Für Mengenoperationen gibt es ebenfalls einige nützliche Befehle: `\in` ($\in$), `\notin` ($\notin$), `\subset` ($\subset$), `\supset` ($\supset$), `\subseteq` ($\subseteq$), `\supseteq` ($\supseteq$), `\setminus` ($\setminus$), `\times` ($\times$), `\emptyset` ($\emptyset$) und `\varnothing` ($\varnothing$). Betrachten wir auch hierzu ein kleines Beispiel:

```
Sei \(x \in \mathbb{N}\) sowie
   \(M \subseteq \mathbb{R}^+\).
```

Sei $x \in \mathbb{N}$ sowie $M \subseteq \mathbb{R}^+$.

Übrigens, für komplexe Zahlen besteht `\Re` ($\Re$) und `\Im` ($\Im$).

Für Logik-Operationen haben wir für Junktoren die folgenden Befehle `\neg` ($\neg$), `\implies` ($\implies$), `\iff` ($\iff$), `\land` ($\land$) und `\lor` ($\lor$)oder die Alternativen (ich mag diese teilweise mehr, primär augrund der Abstände ☺) `\leftarrow` ($\leftarrow$), `\Leftarrow` ($\Leftarrow$), `\rightarrow` ($\rightarrow$), `\Rightarrow` ($\Rightarrow$), `\leftrightarrow` ($\leftrightarrow$), `\leftrightarrow` ($\leftrightarrow$) und `\nearrow` ($\nearrow$)und für Quantoren: `\exists` ($\exists$), `\nexists` ($\nexists$) und `\forall` ($\forall$).

Auch hierfür möchten wir natürlich ein Beispiel:

```
\(\forall h \in \text{Hamster}
  \exists m \in \text{Menschen}:
  \text{mag}(m,h) \)
```

$\forall h \in \text{Hamster} \exists m \in \text{Menschen} : \text{mag}(m,h)$

Positive Abstände können wir mittels `\,` sowie `\:` und `\;` definieren, negative Abstände durch `\!`. Betrachten wir hierzu einmal ein weiteres Beispiel:

```
\(\forall h \in \text{Hamster}\,
  \exists m \in \text{Menschen}
  \::\: \text{mag}(m,h) \)
```

$\forall h \in \text{Hamster} \exists m \in \text{Menschen} \: : \: \text{mag}(m,h)$

3.3 Nützliche Umgebungen

Für die im Folgenden vorgestellten Umgebungen empfiehlt es sich, das Paket `amsmath` einzubinden (wer möchte kann auch `mathtools` verwenden). Dieses liefert uns weiter den Befehl `\text`, damit wir in Formeln auch normalen Text schreiben können:

```
\[ Hallo Welt \cdot \frac{\text{Hallo We} lt}{14\pi} \]
```

Ergibt:

$$HalloWelt \cdot \frac{\text{Hallo Welt}}{14\pi}$$

Weiter sinnvoll ist die Umgebung `align`, die es mittels des Und-Symbols (`&`) erlaubt Gleichungen horizontal zu mitteln. *Übrigens: Eine Umgebung besitzt in Latex einen Namen (wie `align`), der den Befehlen `\begin` und `\end` übergeben werden muss. Diese Befehle begrenzen dann die jeweilige Umgebung.*Eine neue Zeile startet hierbei `\\`:

```
\begin{align}
   42 * 3 &= 21 * 6 \\
   5 * 3 &= 1 \\
   &= 42 \pi + 3 * 2 - 6
\end{align}
```

$$\begin{aligned} 42 * 3 &= 21 * 6 && (1)\\ 5 * 3 &= 1 && (2)\\ &= 42\pi + 3 * 2 - 6 && (3) \end{aligned}$$

Die seitliche Nummerierung kann durch das Setzen eines Sternchens und somit dem Verwenden von `align*` unterdrückt werden.

Eine Matrix können wir mittels `pmatrix` erzeugen. Möchten wir eine Matrix ohne Klammern (oder mit eigenen Klammern), so können wir komplett Analog mittels `matrix` eine Matrix ohne diese erzeugen.

Innerhalb dieser Umgebung können wir mittels einem Und-Symbol (&) Spalten und \ Zeilen trennen. Hilfreich sollt ein diesem Kontext das folgende Beispiel sein:

```
\(\begin{pmatrix}
   1 & 2 & 3 & 42 \\
   219 & \cdots & 1 & 2 \\
   x_1 & 3 & 3 & 7
\end{pmatrix}\)
```

$$\begin{pmatrix} 1 & 2 & 3 & 42 \\ 219 & \cdots & 1 & 2 \\ x_1 & 3 & 3 & 7 \end{pmatrix}$$

Möchten wir eine partiell definierte Funktion anzeigen, so empfiehlt sich `cases`, das an sich noch eine geschweifte Klammer auf der linken Seite hinzufügt:

```
\(\min(x,y) = \begin{cases}
   x, & \text{ falls } x < y, \\
   y, & \text{ sonst.}
\end{cases}\)
```

$$\min(x,y) = \begin{cases} x, & \text{falls } x < y, \\ y, & \text{sonst.} \end{cases}$$

Übrigens, interessante Umgebungen die es auch noch lohnt anzusehen sind: gather, gather*, alignat, alignat*, array und equation.

Bemerkung 5 – Aktueller Mathestand

Glückwunsch, du hast den aktuellen Stand des Mathe-Ausblicks erreicht!

4 Grafiken, Tabellen und Auflistungen

4.1 Grafiken einbinden

Im Folgenden verwenden wir das Paket `graphicx`, welches es uns ermöglicht PDFs, PNGs, JPGs und viele andere Grafikformate einzubinden und anzuzeigen. Vorrausgesetzt, wir haben rechts abgebildete Verzeichnisstruktur, können wir unser Bild wie folgt einsetzen:


```
\includegraphics{images/example.png}
```

Da das Bild in der Regel viel zu groß/klein (auf jedenfall nicht richtig 😉) rüberkommt, können wir diesem Befehl wieder *optionale* Parameter mittels den eckigen Klammern übergeben. So können wir es entweder auf eine gewünschte Breite (`width=XXcm`), höhe (`height=XXcm`) oder ein gewünschtes Seitenverhältnis (`width=XXcm,height=XXcm`) skalieren. Möchten wir, dass das Bild nicht verzerrt wird, so genügt die zusätzliche Angabe von `keepaspectratio`. So liefert:

```
\includegraphics[width=3.5cm]{images/example.png}
```

4.2 Grafiken betiteln und auflisten

Von Haus aus liefert uns LaTeX die Umgebung `figure`, mit der wir Bilder einfach beschriften und auch (ähnlich zu `\tableofcontents`) auflisten können. Mittels `\caption` lassen sich die Bildunterschriften angeben:

```
\begin{figure}\centering
   \includegraphics[width=3.5cm]{images/example.png}
   \caption{It is, the sigil!}
\end{figure}
```

Abbildung 1: It is, the sigil!

Das `\centering` wurde hier angefügt, damit die Grafik zentriert über der Beschriftung angezeigt wird, ist allerdings optional. `\caption` akzeptiert (wie `\section`) ein optionales Argument in eckigen Klammern unter dem es dann in der zugehörigen Liste aufgeführt wird. Diese generieren wir durch `\listoffigures`:

Abbildungsverzeichnis

4.3 Schnelle Tabellen

Tabellen werden über die Umgebung `tabular` erzeugt, wobei diese ein weiteres Argument benötigt: die Spalten, beziehungsweise genauer: die Ausrichtung der Spalten. Die wichtigsten hier sind `l`, `c`, `r`, `m` und `p`, wobei die ersten drei jeweils der links-bündigen (engl. *left*), zentriertern (engl. *center*) und rechts-bündigen (engl. *right*) Spalte entsprechen, eine neue Zeile wird mit `\\` eingeleitet, Spalten werden durch `&` getrennt:

```
\begin{tabular}{lrc}
  Hallo & Welt & na \\
  wie & geht & es dir denn? \\
  Geht es dir & gut? & Frag ich mich \\
  Ich brauch & nen neuen & Dummytext.
\end{tabular}
```

Hallo	Welt	na
wie	geht	es dir denn?
Geht es dir	gut?	Frag ich mich
Ich brauch	nen neuen	Dummytext.

Vertikale Linien können in den Spalten mit einem `|` (senkrechten Strich) angegeben werden, horizontale Linien werden mittels `\hline` angegeben. Die Spalten `m` und `p` benötigen jeweils noch ein Argument über die Breite. Die `m`-Spalte wird vertikal gemittelt, die `p`-Spalte wird einfach oben gesetzt. Jetzt einmal alles zusammen:

```
\begin{tabular}{|r|p{1cm}||m{2cm}|}
  Hallo & Welt & na \\
  \hline
  wie & geht & es dir denn? \\
  \hline \hline
  Geht es dir & gut? & Frag ich mich \\
  Ich brauch & nen neuen & Dummytext.\\
  \hline
\end{tabular}
```

Hallo	Welt	na
wie	geht	es dir denn?
Geht es dir	gut?	Frag ich mich
Ich brauch	nen neuen	Dummytext.

Hinweis: Es mag hier vielleicht wirken, als würde die `m`-Spalte gar nicht mittig ausgerichtet werden, allerdings kontrolliert die längste Spalte die Orientierung, es empfiehlt sich also nur entweder `m` oder `p`-Spalten zu verwenden.
Übrigens: Analog zu `figure` existiert die Umgebung `table`, die ebenfalls mittles `\caption` einen Untertitel und mittels `\listoftables` eine Auflistung erlaubt.

4.4 Hübsche Tabellen

Bemerkung 6 – Aktueller Tabellenstand

Glückwunsch, du hast den aktuellen Stand des Tabellen-Ausblicks erreicht! Ich empfehle das Paket `booktabs`[11] für die weitere Erkundungsreise. Interessiert man sich für Tabellen, die über mehrere Seiten gehen, so gilt es `longtable`[12] zu bestaunen ☺.

4.5 Auflistungen

Man kennt sie, sie sind fast überall: Listen (Schreckens-Musik-Einspieler 😄). LaTeX unterscheidet drei Arten von Listen (die ich hier innerhalb einer beschrifteten Liste vorstelle):

Unsortierte Listen: Diese erzeugen wir mittels der `itemize`-Umgebung:[13]

```
\begin{itemize}
  \item Ich bin ein Punkt
  \item Ich bin auch ein Punkt
  \item Ich bin eine Biene
\end{itemize}
```

- Ich bin ein Punkt
- Ich bin auch ein Punkt
- Ich bin eine Biene

Sortierte Listen: Diese funktionieren im Prinzip analog zu `itemize`, allerdings werden diese durchnummeriert:

```
\begin{enumerate}
  \item Ich bin ein Punkt
  \item Ich bin auch ein Punkt
  \item Ich bin eine Biene
\end{enumerate}
```

1. Ich bin ein Punkt
2. Ich bin auch ein Punkt
3. Ich bin eine Biene

Beschriftete Liste Hier Kann bei jedem Element mittels Eckigen Klammern noch der Bezeichner für den Punkt übergeben werden (*Genau genommen kann dies bei jeder Liste gemacht werden, aber naja, es gibt nur selten Grund dazu*):

```
\begin{description}
  \item[Punkt A] Ich bin ein Punkt
  \item[B] Ich bin auch ein Punkt
  \item[Die Biene] Ich bin eine Biene
\end{description}
```

Punkt A Ich bin ein Punkt

B Ich bin auch ein Punkt

Die Biene Ich bin eine Biene

[11] `https://www.ctan.org/pkg/booktabs/`
[12] `https://www.ctan.org/pkg/longtable`
[13] Die unsortierten Listen habe ich in diesem Dokument *stark* modifiziert, sie weichen deswegen vom „normalen“ aussehen ab.

Alle diese Listen vollführen automatisch Zeilen- und Seitenumbrüche sowie Einrückungen. Möchte man zum Beispiel die Gestalt der Enumerierung von `enumerate` so empfiehlt sich das Paket `enumitem`,[14] mit diesem ist zum Beispiel folgendes möglich:

```
\begin{enumerate}[label=\alph*)]
  \item Ich bin ein Punkt
  \item Ich bin auch ein Punkt
  \item Ich bin eine Biene
\end{enumerate}
```

a) Ich bin ein Punkt

b) Ich bin auch ein Punkt

c) Ich bin eine Biene

5 Source- und Pseudocode

5.1 Java-Code

Zur farblichen Hervorhebung von Quellcode eignet sich das Paket `listings`, welches nebst der Umgebung `lstlisting` noch Befehle wie `\lstinline` und `\lstinputlisting` hinzufügt. Standardmäßig sind Umlaute ein Problem, dem werden wir uns allerdings auch noch annehmen:

```
\begin{lstlisting}
Ich bin Pseudocode
Ist das nicht wunderbar?
Zuuuug zuuuug.
\end{lstlisting}
```

```
Ich bin Pseudocode
Ist das nicht wunderbar?
Zuuuug zuuuug.
```

Eine Sprache hinzuzufügen ist denkbar einfach, wir können der Umgebung in eckigen Klammern Argumente übergeben, wie zum Beispiel:

```
\begin{lstlisting}[language=java]
public static void
       main(String[] args){
   System.out.println("Hi");
}
\end{lstlisting}
```

```
public static void
        main(String[] args){
    System.out.println("Hi");
}
```

Vermutlich wird das Highlighting nicht farbig sein,[15] wir werden uns diesem aber gleich annehmen. Zuerst allerdings noch, wie wir Zeilennummern hinzufügen:

```
\begin{lstlisting}[language=java,↩
   numbers=left]
public static void
       main(String[] args){
   System.out.println("Hi");
}
\end{lstlisting}
```

```
1public static void
2        main(String[] args){
3    System.out.println("Hi");
4}
```

Bemerkung 7 – Die inline und input-Variante

Da wir im Folgenden im wesentlichen nur an den Argumenten arbeiten,die wir der Umgebung übergeben, gilt zu beachten, dass auch `\lstinline` und `\lstinputlisting` diese Argumente übernehmen.

[14] https://www.ctan.org/pkg/enumitem

[15] Es ist nur immer so aufwändig alle meine Einstellungen für Source-Code über den Haufen zu werfen.

Ersterer Befehl funktioniert analog zu `\verb`, so können wir zum Beispiel Schreiben:

```
Verwenden wir hier
\lstinline[language=java]|int crop|
anstelle von \lstinline|String crop|,↩
    ...
```

Verwenden wir hier **int** crop anstelle von `String crop`, ...

`\lstinputlisting` erwartet anstelle von Code den Pfad zu einer Quellcodedatei an.

Natürlich sind wir hochindividuell, und möchte, dass der Code genau so markiert wird, wie wir dies wünschen. Hierfür schreiben wir einfach folgendes in der Präambel/vor der Benutzung:[16]

```
\lstdefinestyle{MeinStil}{
   breaklines    = true, % erlaube Zeilenumbrüche
   % backgroundcolor = \color{yellow!15},
   % rulecolor    = \color{green!15},
   stringstyle   = \color{teal}, % Zeichenketten in Teal
   keywordstyle  = \color{orange}, % Keywords in Orange
   keywordstyle  = [2]\itshape, % Secondary Keywords in kursiv
   basicstyle    = \ttfamily, % Schreibmaschinenschrift
   commentstyle  = \color{gray}\itshape, % Kommentare in grau und kursiv
   extendedchars = true,
   numberstyle   = \scriptsize, % Kleine Nummerierung
   % frame         = single, % wenn rahmen gewünscht
   numbers       = left, % wir wollen immer Zeilennumern
   numbersep     = 7pt % Diese sollen weiter sein
}
```

Nun können wir den Stil verwenden (hier wird `xcolor` ebenfalls benötigt, für die Farben):

```
\begin{lstlisting}[language=java,↩
   style=MeinStil]
public static void
      main(String[] args){
   System.out.println("Hi");
}
\end{lstlisting}
```

```
public static void
        main(String[] args){
    System.out.println("Hi");
}
```

Aber halt, String wird ja zum Beispiel gar nicht farblich markiert, und das ist so besonders wichtig. Wir wollen nun also die Sprache `java` erweitern:

```
\lstdefinelanguage{MeinJava}{
   language=java, % Nutze implizit Java
   alsoletter={@_}, % Methoden/Interfaces können diese Zeichen enthalten
   comment=[l]{//}, % Ein Zeilenkommentar beginnt mit '//'
   morecomment=[s]{/*}{*/}, % Mehrzeiliger Kommenta
   keywordsprefix={@}, % Für interfaces
   morekeywords={String, var}, % Markiere farblich
   morekeywords=[2]{System} % Markiere kursiv
}
```

Wir können die Sprache nun ebenfalls nutzen:

[16] Information: Es sollte keine Leerzeile in diesen Befehl eingefügt werden! Der Befehl ist simpel definiert und erlaubt keine Paragraphen innerhalb des Arguments. Analoges gilt für `\lstdefinelanguage`.

```
\begin{lstlisting}[language=MeinJava,↩
   style=MeinStil]
public static void
       main(String[] args){
   System.out.println("Hi");
}
\end{lstlisting}
```

```
public static void
        main(String[] args){
    System.out.println("Hi");
}
```

Jetzt ist es noch lästig, jedesmal Sprache und Stil anzugeben, wir rufen also `\lstset` auf und setzen damit die Standardargumente für unsere Umgebungen:

```
\lstset{language=MeinJava,style=MeinStil}
```

Damit genügt nun:

```
\begin{lstlisting}
public static void
       main(String[] args){
   System.out.println("Hi");
}
\end{lstlisting}
```

```
public static void
        main(String[] args){
    System.out.println("Hi");
}
```

Ein Problem haben wir noch: Umlaute, deswegen verwenden wir *literates*, dabei handelt es sich um Ersetzungsregeln: Wenn der Befehl ein solches Symbol liest, ersetzt er es durch ein entsprechend anderes. Hier setze ich exemplarisch auch noch ein weiteres replacement, damit man sieht wie man diese erweitern muss:

```
\lstset{
   literate={á}{{\'a}}1 {é}{{\'e}}1 {í}{{\'i}}1 {ó}{{\'o}}1 {ú}{{\'u}}1
   {Á}{{\'A}}1 {É}{{\'E}}1 {Í}{{\'I}}1 {Ó}{{\'O}}1 {Ú}{{\'U}}1
   {à}{{\`a}}1 {è}{{\`e}}1 {ì}{{\`i}}1 {ò}{{\`o}}1 {ù}{{\`u}}1
   {À}{{\`A}}1 {É}{{\'E}}1 {Ì}{{\`I}}1 {Ò}{{\`O}}1 {Ù}{{\`U}}1
   {ä}{{\"a}}1 {ë}{{\"e}}1 {ï}{{\"i}}1 {ö}{{\"o}}1 {ü}{{\"u}}1
   {Ä}{{\"A}}1 {Ë}{{\"E}}1 {Ï}{{\"I}}1 {Ö}{{\"O}}1 {Ü}{{\"U}}1
   {â}{{\^a}}1 {ê}{{\^e}}1 {î}{{\^i}}1 {ô}{{\^o}}1 {û}{{\^u}}1
   {Â}{{\^A}}1 {Ê}{{\^E}}1 {Î}{{\^I}}1 {Ô}{{\^O}}1 {Û}{{\^U}}1
   {ã}{{\~a}}1 {ẽ}{{\~e}}1 {ĩ}{{\~i}}1 {õ}{{\~o}}1 {ũ}{{\~u}}1
   {Ã}{{\~A}}1 {Ẽ}{{\~E}}1 {Ĩ}{{\~I}}1 {Õ}{{\~O}}1 {Ũ}{{\~U}}1
   {œ}{{\oe}}1 {Œ}{{\OE}}1 {æ}{{\ae}}1 {Æ}{{\AE}}1 {ß}{{\ss}}1
   {ű}{{\H{u}}}1 {Ű}{{\H{U}}}1 {ő}{{\H{o}}}1 {Ő}{{\H{O}}}1
   {ç}{{\c c}}1 {Ç}{{\c C}}1 {ø}{{\o}}1 {å}{{\r a}}1 {Å}{{\r A}}1
   {€}{{\euro}}1 {£}{{\pounds}}1 {«}{{\guillemotleft}}1
   {»}{{\guillemotright}}1 {ñ}{{\~n}}1 {Ñ}{{\~N}}1 {¿}{{?`}}1 {¡}{{!`}}1
   % Hier noch das Beispiel:
   {←}{{\( \gets \)}}1
}
```

Die 1 steht übrigens jeweils dafür, dass nur ein Zeichen entsteht. Dies lässt sich nun einfach kopieren und verwenden, beziehungsweise, wo wir schon beim kopieren sind, der Beispielcode für das Dokument befindet sich hier: `https://gist.github.com/EagleoutIce/1490bfd4d71eef73b032670921eab69a`.

5.2 Pseudocode

Zum Schreiben von Pseudocode gibt es das Paket `algorithm2e`, welches die Umgebung `algorithm` liefert, innerhalb der wir eine Reihe von Befehlen an die Hand bekommen.

Innerhalb der Umgebung, können wir einfach drauf los schreiben.[17] Wichtig ist allerdings: wir sollten einen Algorithmus stets mittels `\caption` benennen (das ist nicht wirklich nötig, empfiehlt sich allerdings ☺)

```
\begin{algorithm}
   Hallo Welt
   \caption{Der Titel}
\end{algorithm}
```

Ergibt:

Algorithmus 1: Der Titel

1 Hallo Welt

Wir haben hier die Befehle `\KwIn` beziehungsweise `\KwData` und `\KwOut` oder `\KwResult` für die Deklaration der Ein- und Ausgabe für den Algorithmus:

```
\begin{algorithm}
   \KwIn{Eine epische Eingabe}
   \KwOut{Eine unglaubliche Ausgabe}
   Hallo Welt
   \caption{Der Titel}
\end{algorithm}
```

Ergibt:

Algorithmus 2: Der Titel

Input: Eine epische Eingabe
Output: Eine unglaubliche Ausgabe

1 Hallo Welt

Schleifen können wir mit `\For{⟨Bedingung⟩}{⟨Rumpf⟩}` und `\While{⟨Bedingung⟩}{⟨Rumpf⟩}` erzeugen. Hier das Beispiel mit `\For`:

```
\begin{algorithm}
   \KwIn{Eine epische Eingabe}
   \KwOut{Eine unglaubliche Ausgabe}
   Hallo Welt
   \For{i = 0 \KwTo 42}{
      Füttere die Katze
   }
   \caption{Der Titel}
\end{algorithm}
```

Ergibt:

Algorithmus 3: Der Titel

Input: Eine epische Eingabe
Output: Eine unglaubliche Ausgabe

1 Hallo Welt **for** $i = 0$ **to** 42 **do**
2 | Füttere die Katze
3 **end**

[17] Für weitere Informationen wie man dieses Paket optisch einrichtet bitte einfach in die Dokumentation sehen ☺.

Hm doof, jetzt hat er die For-Schleife in der gleichen Zeile begonnen wie auch das Hallo Welt. Dies liegt daran, dass ein Zeilen-Ende eigentlich mit `\;` gekennzeichnet werden muss. Verbinden wir dies nun noch mit dem Wissen, dass es `\If{⟨Bedingung:ran⟩}{⟨Wahr-Rumpf⟩},\Else{⟨Falsch-Rumpf⟩}` sowie `\eIf{⟨Bedingung:ran⟩}{⟨Wahr-Rumpf⟩}{⟨Falsch-Rumpf⟩}` gibt und wir haben die Möglichkeit schon ganz tolle Algorithmen zu setzen.:

```
\begin{algorithm}
   \KwIn{Eingabe als Liste von \(n\) Zahlen: \(a_1, \ldots, a_n\)}
   \KwData{Eine Definition die es so gibt}
   \KwResult{Ein Ergebnis}
   \For{i = 1 \KwTo n}{
      \(gefunden \gets\) falsch\;
      \(j \gets i\)\;
      \While{\(j < n\)}{
         \If{\(a_j > a_i\)}{
             \(gefunden \gets\) wahr\;
             break\;
         }
         \(j \gets j+1\)\;
      }
       \leIf{gefunden}{
          Gebe aus: ja
      }{
          Gebe aus: nein
      }
   }
   \caption{Ich bin der Titel}
\end{algorithm}
```

Ergibt:

Algorithmus 4: Ich bin der Titel

Input: Eingabe als Liste von n Zahlen: $a_1, \ldots, a_n$
Data: Eine Definition die es so gibt
Result: Ein Ergebnis

```
1  for i = 1 to n do
2  |  gefunden ← falsch;
3  |  j ← i;
4  |  while j < n do
5  |  |  if a_j > a_i then
6  |  |  |  gefunden ← wahr;
7  |  |  |  break;
8  |  |  end
9  |  |  j ← j + 1;
10 |  end
11 |  if gefunden then Gebe aus: ja else Gebe aus: nein ;
12 end
```

Information: wir können vor zum Beispiel `\If` noch ein `l` packen (also `\lIf`, damit das If in nur einer Zeile gesetzt wird.)

Zuletzt wollen wir uns noch kurz ansehen wie Kommentare funktionieren. Hier haben wir zwei Befehle: `\tcc` (C-Stil) und `\tcp` (C++-Stil) die uns Kommentare setzen:

```
\begin{algorithm}
   \KwIn{Eine epische Eingabe}
   \KwOut{Eine unglaubliche Ausgabe}
   \tcc{Kommentar}
   \For{i = 0 \KwTo 42}{
      Füttere die Katze
   }
   \tcp{Epischer Kommentar}
   \caption{Der Titel}
\end{algorithm}
```

Ergibt:

Algorithmus 5: Der Titel

Input: Eine epische Eingabe
Output: Eine unglaubliche Ausgabe
/* Kommentar */
1 **for** $i = 0$ **to** 42 **do**
2 | Füttere die Katze
3 **end**
// Epischer Kommentar

Diese können wir auch mittels einiger Modifikationen (wieder: konsultiert für genaueres die Dokumentation) in der gleichen Zeile wie Bedingungen/Schleifen etc. setzen. Hierzu schreiben wir sie in runde Klammern nach dem Befehl:

```
\begin{algorithm}
   \KwIn{Eine epische Eingabe}
   \KwOut{Eine unglaubliche Ausgabe}
   \For(\tcp*[h]{Epischer Kommentar}){i = 0 \KwTo 42}{
      Füttere die Katze
   }
   \caption{Der Titel}
\end{algorithm}
```

Ergibt:

Algorithmus 6: Der Titel

Input: Eine epische Eingabe
Output: Eine unglaubliche Ausgabe
1 **for** $i = 0$ **to** 42 **do** // Epischer Kommentar
2 | Füttere die Katze
3 **end**

A Tabellen und Listen

Die folgenden Tabellen und Listen stellen *lediglich einen Ausschnitt* und somit *nicht* alle verfügbaren Symbole dar. Weiter ist es durchaus möglich, dass die gezeigten Beispielsymbole, abhängig von der Schriftart und weiteren Konfigurationen in ihrer Gestalt abweichen können!

A.1 Mathematische Symbole

► *Griechische Variablen:*

- ► \(\alpha\) (α)
- ► \(\beta\) (β)
- ► \(\Gamma\) (Γ)
- ► \(\gamma\) (γ)
- ► \(\Delta\) (Δ)
- ► \(\delta\) (δ)
- ► \(\epsilon\) (ϵ)
- ► \(\zeta\) (ζ)
- ► \(\eta\) (η)
- ► \(\Theta\) (Θ)
- ► \(\theta\) (θ)
- ► \(\vartheta\) (ϑ)
- ► \(\iota\) (ι)
- ► \(\kappa\) (κ)
- ► \(\varkappa\) ($\varkappa$)
- ► \(\Lambda\) (Λ)
- ► \(\lambda\) (λ)
- ► \(\mu\) (μ)
- ► \(\nu\) (ν)
- ► \(\Xi\) (Ξ)
- ► \(\xi\) (ξ)
- ► \(\Pi\) (Π)
- ► \(\pi\) (π)
- ► \(\varpi\) (ϖ)
- ► \(\rho\) (ρ)
- ► \(\varrho\) (ϱ)
- ► \(\Sigma\) (Σ)
- ► \(\sigma\) (σ)
- ► \(\varsigma\) (ς)
- ► \(\tau\) (τ)
- ► \(\Upsilon\) (Υ)
- ► \(\upsilon\) (υ)
- ► \(\Phi\) (Φ)
- ► \(\phi\) (ϕ)
- ► \(\chi\) (χ)
- ► \(\Psi\) (Ψ)
- ► \(\psi\) (ψ)
- ► \(\Omega\) (Ω)
- ► \(\omega\) (ω)

► *Mathematische Operatoren* (Paket: amsmath):

- ► \(\leq\) ($\leq$)
- ► \(\geq\) ($\geq$)
- ► \(\prec\) ($\prec$)
- ► \(\succ\) ($\succ$)
- ► \(\doteq\) ($\doteq$)
- ► \(\sim\) ($\sim$)
- ► \(\simeq\) ($\simeq$)
- ► \(\approx\) ($\approx$)
- ► \(\ll\) ($\ll$)
- ► \(\gg\) ($\gg$)
- ► \(\lll\) ($\lll$)
- ► \(\ggg\) ($\ggg$)
- ► \(\propto\) ($\propto$)
- ► \(\neq\) ($\neq$)
- ► \(\subset\) ($\subset$)
- ► \(\supset\) ($\supset$)
- ► \(\subseteq\) ($\subseteq$)
- ► \(\supseteq\) ($\supseteq$)
- ► \(\parallel\) ($\parallel$)
- ► \(\in\) ($\in$)
- ► \(\ni\) ($\ni$)
- ► \(\notin\) ($\notin$)
- ► \(\times\) ($\times$)
- ► \(\pm\) ($\pm$)
- ► \(\cdot\) ($\cdot$)
- ► \(\vee\) ($\vee$)
- ► \(\wedge\) ($\wedge$)
- ► \(\cap\) ($\cap$)
- ► \(\cup\) ($\cup$)
- ► \(\exists\) ($\exists$)
- ► \(\nexists\) ($\nexists$)
- ► \(\forall\) ($\forall$)
- ► \(\lor\) ($\lor$)
- ► \(\land\) ($\land$)
- ► \(\top\) ($\top$)
- ► \(\bot\) ($\bot$)
- ► \(\neg\) ($\neg$)
- ► \(\to\) ($\to$)
- ► \(\mapsto\) ($\mapsto$)
- ► \(\gets\) ($\gets$)
- ► \(\implies\) ($\implies$)
- ► \(\iff\) ($\iff$)
- ► \(\emptyset\) ($\emptyset$)
- ► \(\lceil\) ($\lceil$)
- ► \(\rceil\) ($\rceil$)
- ► \(\lfloor\) ($\lfloor$)
- ► \(\rfloor\) ($\rfloor$)
- ► \(\int\) ($\int$)
- ► \(\Re\) ($\Re$)
- ► \(\Im\) ($\Im$)
- ► \(\aleph\) ($\aleph$)
- ► \(\sin\) (sin)
- ► \(\cos\) (cos)
- ► \(\tan\) (tan)
- ► \(\nabla\) (∇)

B wie Bonus

B.1 Übungsblätter gestalten

Der folgende Abschnitt hat nicht zum Ziel ein umwerfend-pittoresk im Licht der Schönheit strahlendes Übungsblatt zu entwerfen. Ich habe eine neue Umgebung und eine neue Liste für das Dokument erstellt, die ich hier leider nur teilweise erklären kann (Grundlagendokument und so), bei Fragen einfach melden.

Wir beginnen, indem ich den gesamten Code für drei Aufgaben zeige und diesen dann aufdrösele, der komplette Code (also ein kompilierfähiges Dokument, findet sich hier: `https://gist.github.com/EagleoutIce/b7992a3a613a8b445cd92c31d664addf`)

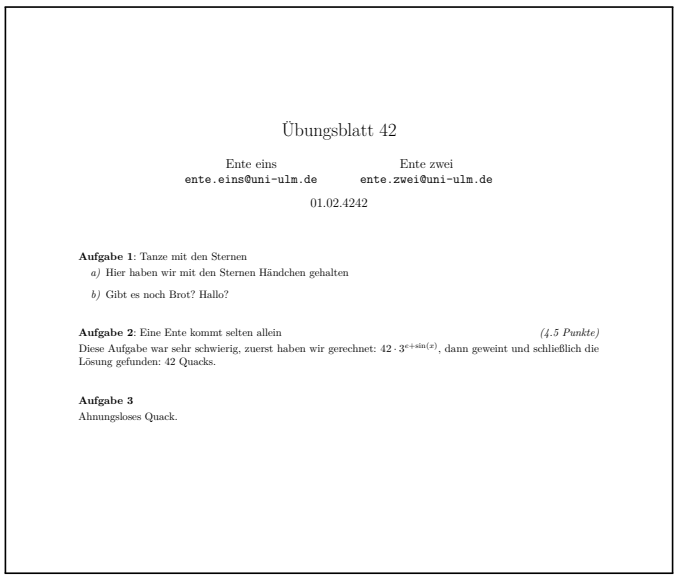
```
\title{Übungsblatt 42}
\author{Ente eins \\ \texttt{ente.eins@uni-ulm.de} \and
      Ente zwei \\ \texttt{ente.zwei@uni-ulm.de}}
\date{01.02.4242}

\maketitle

\begin{aufgabe}{Tanze mit den Sternen}
   \begin{aufgaben}
      \item Hier haben wir mit den Sternen Händchen gehalten
      \item Gibt es noch Brot? Hallo?
   \end{aufgaben}
\end{aufgabe}

\begin{aufgabe}[4.5 Punkte]{Eine Ente kommt selten allein}
   Diese Aufgabe war sehr schwierig, zuerst haben wir gerechnet:
   \(42 \cdot 3^{e + \sin(x)}\), dann geweint und schließlich die
   Lösung gefunden: 42 Quacks.
\end{aufgabe}

\begin{aufgabe}{}
   Ahnungsloses Quack.
\end{aufgabe}
```

Mittels `\title`, `\author` und `\date` setzten wir die Daten für `\maketitle`. Sie kännen eigentlich so ersetzt werden, was sie tun sollte intuitiv klar sein ☺.

Es gibt zwei weitere Umgebungen zu entdecken: `aufgabe` und `aufgaben`. Erstere notiert eine neue Aufgabe, wobei diese automatisch durchnummeriert wird. In geschwungene Klammern dahinter gilt es den Titel zu setzen, man kann auch nichts eintragen, in diesem Fall wird auch kein Doppelpunkt gesetzt. Als optionales Argument kann in eckigen Klammern die Punktzahl übergeben werden (*siehe: `\begin{aufgabe}[4.5 Punkte]{Eine Ente...}`*).

Innerhalb der `aufgabe`-Umgebung kann man dann mittels `aufgaben` analog zu `enumerate` seine Teilaufgaben notieren (*siehe: Tanze mit den Sternen*).

Das Ergebnis sieht wie folgt aus (beschnitten):



Übungsblatt 42

Ente eins
ente.eins@uni-ulm.de

Ente zwei
ente.zwei@uni-ulm.de

01.02.4242

Aufgabe 1: Tanze mit den Sternen

a) Hier haben wir mit den Sternen Händchen gehalten

b) Gibt es noch Brot? Hallo?

Aufgabe 2: Eine Ente kommt selten allein *(4.5 Punkte)*

Diese Aufgabe war sehr schwierig, zuerst haben wir gerechnet: $42 \cdot 3^{e^{i \cdot \sin(x)}}$, dann geweint und schließlich die Lösung gefunden: 42 Quacks.

Aufgabe 3

Ahnungsloses Quack.

Hinweis: Der folgende Teil ist nur dann interessant, wenn man sich dafür interssiert wie `aufgabe` *und* `aufgaben` *implementiert wurde.* Hier arbeiten noch die engagierten, eloquenten Enten 🐤.

Schlagworte