

# LaTeX Advanced

Eigene Environments bis hin zu eigenen Packages

Alexander **Phi** Goetz info@phictional.de

14.04.2022

## Inhalt

- Eigene Commands `\newcommand`, `\renewcommand`
  - `\mathbb{R} \Rightarrow \mathbb{R} \Leftarrow \mathbb{R}`
- Eigene Umgebungen `\newenvironment`, `\renewenvironment`
- Andere Schriftarten
  - `\textsc`, `\texttt`,...
  - LaTeX Font Catalogue with Comic Neue
- Eigene Packages
- Markdown & `pandoc`

## Einleitung

Im Verlauf des Workshops wird an der typischen Tübinger-Info TeX-Vorlage herumeditiert.

## Commands

Commands ohne Argument:

`\leq`  $\Rightarrow \leq$

Commands **mit** *[optionalen]* Argumenten:

`\sqrt{4}`  $\Rightarrow \sqrt{4}$

`\sqrt[3]{4}`  $\Rightarrow \sqrt[3]{4}$

Zum Benutzen gibt es ein **LATEX Mathematical Symbols** und zum verstehen der Befehle `\newcommand` und `\renewcommand` gibts die Overleaf Dokumentation<sup>1</sup>.

---

<sup>1</sup>Overleaf Dokumentaiton: Commands

## **\newcommand**

`\newcommand{name}[n][defaultFirst]{body}`

- **name**: Name des Befehls
- **n**: Anzahl von Parametern
- **defaultFirst**: Standardwert für ersten Parameter
- **body**: Hier werden Parameter mittels  $\#x$ ,  $x \in \{1, \dots, n\}$  verwendet

### **Beispiel: \hello**

Wir wollen einen Befehl, der Leute grüßt. Der sieht so aus:

```
\newcommand{\hello}[1][Phi]{Hallo #1, wie geht's so?}
```

und verhält sich so:

```
>> \hello
Hallo Phi, wie geht's so?
>> \hello[Jules]
Hallo Jules, wie geht's so?
```

## **\renewcommand**

99% identisch zu `\newcommand`

`\renewcommand{name}[n][defaultFirst]{body}`

- **name**: Name des Befehls
- **n**: Anzahl von Parametern
- **defaultFirst**: Standardwert für ersten Parameter
- **body**: Hier werden Parameter mittels  $\#x$ ,  $x \in \{1, \dots, n\}$  verwendet

Das Überschreiben von Befehlen geht ohne Probleme.

```
>> $\Box\square$

>> \renewcommand{\square}{^2}
>> $\Box\square$
```

2

## **Environments**

Die bekannte Umgebung **table**:

```
\begin{table}
  \begin{tabular}{l|c}
    Workshop & Teilnehmer \\\hline
    Bash      & 20 \\\end{tabular}
\end{table}
```

```

        Git      & 20 \\
        Python   & 20 \\
        LaTeX    & 20 \\
        \dots
    \end{tabular}
    \caption{Teilnehmer}
\end{table}

```

Workshop	Teilnehmer
Bash	20
Git	20
Python	20
LaTeX	20

Soweit alter Schuh. Jetzt wollen wir selber Umgebungen schreiben.

### **\newenvironment**

Eine Ähnlichkeit zu `\newcommand` existiert.

```
\newenvironment{name}[n]{before}{after}
```

- **name**: Name des Befehls
- **n**: Anzahl von Parametern
- **before**: Der Code der *vor* dem Inhalt kommt
- **body**: Der Code der *nach* dem Inhalt kommt

Zwischen **before** und **after** landet der Code, der zwischen `\begin{name}` und `\end{name}` geschrieben werden soll.

### **\newenvironment mit Nummerierung**

`\able`, `\figure` usw. haben eine Zahl, die mit hochzählt. Sogas wollen wir auch bauen können.

```

\newcounter{name}           % Zähler anlegen
\newenvironment{name}[n][defaultFirst]{
    \refstepcounter{name}    % Referenzieren zum Inkrementieren
    \thename                  % Zahl abrufen
    before
}{after}

```

(Das funktioniert so auch in normalen *Commands*. Siehe `\section`.)

### **Beispiel \newenvironment**

```

\newenvironment{para}[1]{
    \begin{minipage}{1.5em}

```

```

        \rotatebox{90}{\textsc{#1}}
\end{minipage}\begin{minipage}{\linewidth}
}{
    \end{minipage}\smallskip
}

```

- minipage ermöglicht es horizontal “Boxen” anzulegen.
- \rotatebox[Winkel]{Inhalt} dreht den Inhalt um den angegebenen Winkel.

## Beispiel \newenvironment mit Nummerierung

Lass uns einen Witze-Katalog anlegen. Der soll indiziert sein. Also können wir die Umgebung so definieren.

```

\newcounter{joke}
\newenvironment{joke}[1]{
    \refstepcounter{joke}
    \noindent\colorbox{gray!50!white}{
        \textbf{Witz~\thejoke}
    } \\\[.5em]
}{\medskip}

```

## \renewenvironment

```

\renewenvironment{name}[n][defaultFirst]{before}{after}

```

Auch hier gibt es wieder ein \renewenvironment. Es verhält sich genauso wie \newenvironment und überschreibt die vorher definierte/importierte Umgebung.

# Aufgaben

## Aufgabe 1: Zahlenräume

Es gibt den Befehl \mathbb{} für den Mathe-Modus. \mathbb{R} ⇒ ℝ

Schreibe

$$\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$$

mittels Custom Commands nach dem Muster \bbR ⇒ ℝ oder \bb{R} ⇒ ℝ

## Aufgabe 2: Aufgaben-Section mit Punkten

Ziemlich was der Titel sagt. Entwerfe einen Befehl \aufgabe, der zwei Argumenten erwartet:

- Nummer der Aufgabe

- Punkte für die Aufgabe

Hilfreiche Befehle sind `\section*` und `\hfill`. Optional sind `\small` und `\textcolor{color}{text}`.

## Aufgabe 1

(5 Punkte)

Figure 1: So solls aussehen

## Aufgabe 3: Lösungs-Umgebung

Entwerfe eine Umgebung `lösung`, die den Text “**Lösung:**” als Präfix besitzt. Der Präfix ist der einzige und optionale Parameter.

Hilfreiche Befehle:

- `\medskip` vertikale Lücke
- `\noindent` keine Einrückung
- `\textbf{}` Fett geschrieben

## Fonts

Wir ScHreibeN einEN  
ErPRüGERbRief mit Tex

Figure 2: Mit Font Styles und Sizes freidrehen

Schrift macht ein Dokument erst zugänglich. Wie wir mit Schriftarten, -stilen und -größen arbeiten kommt jetzt.

## Font Styles

Font Sizes in Tex:

<code>\tiny</code>	<code>\scriptsize</code>	<code>\footnotesize</code>	<code>\small</code>	<code>\normalsize</code>
Text	Text	Text	Text	Text

<code>\large</code>	<code>\Large</code>	<code>\LARGE</code>	<code>\huge</code>	<code>\Huge</code>
Text	Text	Text	Text	Text

Verwendung von `\tiny`:

`{\tiny So werden die Schriftgrößen verwendet!}`

So werden die Schriftgrößen verwendet!

Font Styles für Text im Mathe-Modus, sowie den Text-Modus (es gibt da noch mehr):

Mathe Text Style		Text Style	
<code>\$_{\mathcal}{A}\$</code>	$\mathcal{A}$	<code>\textbf{Text}</code>	<b>Text</b>
<code>\$_{\mathbb}{A}\$</code>	$\mathbb{A}$	<code>\textit{Text}</code>	<i>Text</i>
<code>\$_{\mathfrak}{A}\$</code>	$\mathfrak{A}$	<code>\texttt{Text}</code>	<b>Text</b>
<code>\$_{\mathsf}{A}\$</code>	$\mathsf{A}$	<code>\textsf{Text}</code>	Text
<code>\$_{\mathbf}{A}\$</code>	$\mathbf{A}$	<code>\textrm{Text}</code>	Text

Alles groß-/kleinschreiben ist kein Problem:

<code>\uppercase{LaTeX}</code>	<code>\lowercase{LaTeX}</code>
LATEX	latex

## Andere Fonts

“Schriftarten, die nicht”häufig” vorkommen, sind verdammt nervig.”

**MiKTeX**, **TeX Live** werden mit einer Auswahl von Schriftarten ausgeliefert.<sup>2</sup>

---

<sup>2</sup>The LaTeX Font Catalogue

# The L<sup>A</sup>T<sub>E</sub>X Font Catalogue

[\[FRONT PAGE\]](#) [\[SERIF FONTS\]](#) [\[SERIF FONTS, SUB-CATEGORISED\]](#) [\[SANS SERIF FONTS\]](#) [\[TYPEWRITER FONTS\]](#) [\[CALLIGRAPHICAL AND HANDWRITTEN FONTS\]](#) [\[UNCIAL FONTS\]](#) [\[BLACKLETTER FONTS\]](#) [\[OTHER FONTS\]](#)  
[\[FONTS WITH MATH SUPPORT\]](#) [\[FONTS WITH OPENTYPE OR TRUETYPE SUPPORT\]](#) [\[ALL FONTS, BY CATEGORY\]](#) [\[ALL FONTS, ALPHABETICALLY\]](#) [\[ABOUT THE L<sup>A</sup>T<sub>E</sub>X FONT CATALOGUE\]](#) [\[PACKAGES THAT PROVIDE MATH SUPPORT\]](#)

## Finding the right font

[Fonts with math support](#)  
[Serif Fonts](#)  
[Sans Serif Fonts](#)  
[Typewriter Fonts](#)  
[Calligraphical and Handwritten Fonts](#)  
[Uncial Fonts](#)  
[Blackletter Fonts](#)  
[Other Fonts](#)  
[Fonts in upper case only](#)  
[Decorative Initials](#)  
[Other \(mostly decorative\) Fonts](#)

## Dokumenten-Schriftart

Der Standart-Weg um **dokumentenweit** Fonts einzustellen (unter *pdf(La)TeX* und *MiKTeX*) ist folgender:

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

Die PDF-Engines *Xe(La)TeX* und *Lua(La)TeX* verwenden das Package `fontspec` und sind iA. besser im Umgang mit Schriftarten.

### Beispiel (Comic Sans-ish):

Das wir jetzt Schriftarten verändern können mißbrauchen wir direkt:

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

## Inline Schriftart

In **Word** kann man einzelne Textschnipsel in verschiedenen Schriftarten schreiben. Geht das in TeX auch? **Ja, aber umständlicher.**

### Beispiel (Inline):

Hier steht was  
`{\fontfamily{ComicNeue-TLF}\selectfont}` Hier Comic Neue.  
Hier wieder nicht.

Wir können auch einen (Text-)Befehl anlegen.

```
\newcommand{\comicneue}{\fontfamily{ComicNeue-TLF}\selectfont}
\DeclareTextFontCommand{\textcn}{\comicneue}
```

Der Befehl `\textcn` wird dann so verwendet wie `\textbf`, `\textrm`, `\textsc`,  
...

Die Frage ist jetzt aber: Wie kommen wir überhaupt auf **ComicNeue-TLF**?

1. Gewünschte Schriftart als Standard setzen (`\usepackage{...}`)
2. Im Text `\familydefault`
3. Dieser String ist die gesuchte `fontfamily`

Beispiel (`\comicneue` / `\textcn`)

```
{\comicneue Hier steht etwas auf diese Weise} \\
\textcn{Hier steht etwas auf die andere Weise}
```

## Aufgaben

### Aufgabe 6: Neue Monospaced Schriftart

Suche dir im LaTeX Font Catalogue eine neue Typewriter-Font aus, die die **Computer Modern** Monospace Schriftart ersetzt. (Ich empfehle **Fira Mono** oder **DejaVu Sans Mono**)

Standart	Fira Mono	DejaVu Sans Mono
Test123	Test123	Test123

Figure 3: Vergleich der Schriftarten

### Aufgabe 5: Awesome Fonts

Neben den normalen Schriftarten gibts auch andere witzige Dinge.

Deine Aufgabe ist es folgende Sequenz von Symbolen anzugeben:



Figure 4: Die awesome Symbolsequenz

Dein Startpunkt ist CTAN.

## Packages

### Motivation

Warum der Spaß?



- Die Präambel läuft über / ist zu lang
- Viele Dokumente mit der selben / ähnlichen Präambel (Übungsblätter)
- Befehle mit anderen teilen

## Idee

Präambel in eine `settings.tex` packen. `\input{settings.tex}` anstatt der alten Präambel.

## Einschub

### `\input` vs `\include`

<code>\input{filename}</code>	<code>\include{filename}</code>
Importiert <code>filename.tex</code>	Importiert <code>filename.tex</code>
Als ob Code in aufrufender Datei stehen würde	In Kompilation eigene Datei
Befehle verwendbar	Befehle nicht verwendbar
	nützlich in großen Projekten

## Was ist ein Package?

Eine Sammlung von Befehlen und Umgebungen, die in anderen Dokumenten eingesetzt werden kann. Nicht viel anders zu `settings.tex`-Lösung.

Die Unterschiede sind:

1. Dateiendung `.sty`
2. Einbindung mittels `\usepackage{packagename}`
3. Angabe des “Headers”

`\NeedsTeXFormat{LaTeX2e}`

`\ProvidesPackage{packagename}[YYYY/MM/dd package description]`

4. `\usepackage` → `\RequirePackage`

Besonderheiten:

- `\newcommand` kann weiter verwendet werden
- `\renewcommand` kann weiter verwendet werden
- `\providecommand` definiert Befehl, falls nicht schon vorher vorhanden
- `\CheckCommand` genau wie `\newcommand`, falls Befehl vorhanden und anders definiert als in `\CheckCommand` wird ein Fehler geworfen

Das lässt sich ausführlicher in der Overleaf Dokumentation nachlesen.<sup>3</sup>

<sup>3</sup>Overleaf Dokumentation: Packages

# Aufgaben

TODO

## Markdown & pandoc

# Was ist Markdown? Das ist Markdown!

- Markdown ist eine `_simple_` Notation für Text, Notizen, usw.
- Wird von Git-hostern unterstützt (`.md`, `.markdown`)
- Ist als "Source-Code" menschenlesbar
- [\[Spezifikation\]](https://github.github.com/gfm) (<https://github.github.com/gfm>)

Gibt verschiedene leicht unterschiedliche Spezifikationen, eine der meist geschriebenen Varianten ist Github Flavored Markdown (GFM) oder Common-Mark. Diese unterscheiden sich kaum.

Zum rendern in PDF, Website, Ebook, Präsentationen, etc. verwenden wir Pandoc<sup>4</sup>.

Der Befehl den wir in der Shell verwenden ist `pandoc`:

```
>>> pandoc --help
pandoc [OPTIONS] [FILES]
-f FORMAT, -r FORMAT      --from=FORMAT, --read=FORMAT
-t FORMAT, -w FORMAT      --to=FORMAT, --write=FORMAT
-o FILE                   --output=FILE
```

## Beispiele

Die folgenden Beispiele sind im `Makefile` verfügbar. Die Befehle stehen beim Beispiel. Im `Makefile` sind auch noch ein paar weitere Befehle zum rumprobieren.

### Beispiel: Markdown zu Website

```
>>> pandoc -f markdown -t html -o info.html info.md
```

Makefile: `make info`

### Beispiel: Markdown zu PDF

```
>>> pandoc -f markdown -t pdf \
>     --pdf-engine=xelatex \
>     -o default.pdf shownotes.md
```

Makefile: `make default` oder `make shownotes` (mit anderer Schriftart)

---

<sup>4</sup>Pandoc "a universal document converter"

### Beispiel: Markdown zu Beamer Präsentation

```
>>> pandoc -f markdown -t beamer -o slides.pdf slides.md
```

Makefile: make beamer oder make fancybeamer (mit anderer Schriftart)

### Beispiel: Markdown zu Slidy Präsentation

```
>>> pandoc -f markdown -t slidy -s --mathjax  
> -o slidy.html slides.md
```

Makefile: make slidy

### Beispiel: Markdown zu revealJs Präsentation

```
>>> pandoc -f markdown -t revealjs -s --mathjax  
> -o revealjs.html slides.md
```

Markdown: make revealjs

Das war der Inhalt des Workshops. Danke fürs Teilnehmen!