

# Die Klasse `drcschool` (2023-10-10 v1.0.5)

Davide Campagnari  
10. Oktober 2023

Mehr als eine Dokumentation ist dies eine Sammlung von Beispielen, die (hoffentlich) alle wesentlichen Fähigkeiten der Klasse `drcschool` zeigen. Sie sind allerdings nicht vollständig und es gibt in der Tat einige Features, die hier nicht beschrieben werden. Der Grund dafür ist, dass sie meiner Meinung nach noch nicht ganz reif sind und ich sie daher als noch nicht „offiziell“ betrachte. Wer durch den Code gehen will, um diese zu entdecken, soll gewarnt sein, dass nicht beschriebene Features geändert oder gar gestrichen werden könnten. . .

## Eingebettete Dateien

In dieser PDF-Datei sind folgende Dateien eingebettet:

**`drcschool.cls`** Die Klasse selbst.

**`drcschool_template.tex`** Der Quellcode dieser Dokumentation.

**`drcschool.cwl`** Die `cwl`-Datei für `TEXStudio`, damit sich der Editor nicht ständig über nicht-definierte Befehle ärgert. Das muss natürlich an der richtigen Stelle platziert werden: Unter Windows 10 ist es bei mir in

`C:\Users\<User>\AppData\Roaming\texstudio\completion\user`

während unter Linux bei

`~/ .config/texstudio/completion/user/`

(Aber dafür gebe ich keine Garantie.) Ob (und gegebenenfalls wie) das für andere `LATEX`-Editoren geht, weiß ich ehrlich gesagt nicht. Eine mögliche Lösung für `TEXworks` kann unter

<https://tex.stackexchange.com/q/118038>

gefunden werden, ist aber relativ alt (2013). Keine Gewähr!

## 1 Laden der Klasse und Optionen

Die Klasse wird ganz normal mittels

`\documentclass[<Optionen>]{drcschool}`

geladen. Zu Grunde liegt die KOMA-Script-Klasse `scrartcl`, deren entsprechende Optionen als Optionen zu `drcschool` weitergegeben werden können.

Es gibt zwei *eigene* Optionen: `nofonts` und `hyperworksheets`. Die zweite Option beschreibe ich am Ende dieses Dokumentes (s. Abschnitt 7). Die erste hebt die Standardeinstellung für die Schriftart aus. Standardmäßig wird eine serifenlose Schrift gewählt:<sup>1</sup> Der aktuelle Stand der Forschung scheint zu sein, dass Kinder mit LRS serifenlose Schriften besser finden. (Zugegeben, für Kinder *ohne* LRS ist es andersrum. . .) Soll jemand meine Standardeinstellung nicht mögen,<sup>2</sup> so kann die Klasse mit der Option `nofonts` geladen werden; daraufhin kann jede/r sämtliche Lieblingsschriftarten auf gewohnte Weise laden.

## 2 Abhängigkeiten

Hier folgt eine kurze Liste (ohne Abhängigkeiten) der geladenen Pakete:

**Schrift- und Sprachpaketen** `fontenc` mit Option `T1`, `microtype`, `babel` mit Option `ngerman`.

**Tabellen** `tabularx`, `booktabs`, `colortbl`, `longtable`.

**Seitenmanagement** `scrlayer-scrpage`, `geometry`.

<sup>1</sup>Für die Neugierigen: `tgheros` mit `newtxsf`, plus einige persönlichen Vorlieben (ich mag die Ziffern von `newtxsf` nicht).

<sup>2</sup>„drc“ ist eine Abkürzung für meinen Namen („r“ steht für meinen zweiten Vornamen, nicht für den Dokortitel). Also ja, ich gestalte die Klasse so, dass sie *mir* gefällt. Dass andere sie nützlich finden, ist nur ein netter Nebeneffekt ; -)

**Verschiedenes** `amsgen`, `environ`<sup>3</sup>, `paralist`, `icomma`, `tikz`, `pgfplots`.

Die Verwendung von `geometry` mit einer KOMA-Script-Klasse ist etwas ungewöhnlich, war aber aus T<sub>E</sub>Xnischen Gründen nötig. Dies bedeutet allerdings, dass die Fähigkeiten von `typearea` nicht wirklich verwendet werden können.

Ach ja: Ich konnte mich noch nie mit Unicode-Engines anfreunden, daher habe ich gar nichts damit getestet... das Einzige, was problematisch sein könnte, sind eben die Schriftarten, und genau das überprüft die Klasse: Wird das Dokument mit LuaT<sub>E</sub>X oder X<sub>Y</sub>T<sub>E</sub>X kompiliert, so wird automatisch die Option `nofonts` verwendet und keine Änderung an der Schriftart vorgenommen.

**Bemerkung** Das geübte Auge wird sehen, dass ich Einiges ermöglicht habe, was eigentlich nicht der normalen L<sup>A</sup>T<sub>E</sub>X-Syntax entspricht. Es gibt beispielsweise Umgebungen, die auch als Makro mit Argumenten funktionieren können. Dies ist *sehr schlechter* T<sub>E</sub>X-Stil, ist aber aus „geschichtlichen“ Gründen gewachsen: Diese Klasse ist während ihrer Nutzung entstanden und daher haben sich einige im Feuer des Gefechtes getroffenen Erstentscheidungen als ungünstig erwiesen. Aus Kompatibilitätsgründen habe ich allerdings einige dieser Entscheidungen „retten“ müssen. Ich werde hier und dort beschreiben, was die „bessere“ Variante ist, aber auch die andere(n) der Vollständigkeit halber angeben.

## 3 Planung einer Unterrichtsstunde

### 3.1 Die Umgebung `{schedule}`

Eine Tabelle mit der Stundenplanung kann mit Hilfe der Umgebung `{schedule}` erstellt werden. Ein typisches Beispiel könnte sein

```
\begin{schedule}
\time{45}
\goal{Einstieg}
\content{Irgendwas}
\methode{UG}
\material{DokuKamera}
\newblock%%%%%%%%%%%%%%%%%%%%%%%%
\time{45}
\goal{Ü}
\content{Irgendwas Anderes}
\methode{EA/PA}
\end{schedule}
```

Die verschiedenen Unterrichtsphasen (also faktisch die Tabellenzeilen) werden durch `\newblock` getrennt. Hier kommt ein etwas größeres Beispiel:

---

<sup>3</sup>Seit 2018 bietet `xparse` ein `b`-Argument für Umgebungen, die den gesamten Inhalt sammeln sollen; seit 2020 ist `xparse` außerdem im L<sup>A</sup>T<sub>E</sub>X-Kernel integriert. Da aber auf meinem Büro-Rechner noch TeX Live 2017 läuft, muss ich bei `environ` bleiben.

# KLASSE 8 — THEMA — MÖGLICHER UNTERTITEL

Zeit	Ziele	Inhalt	Methode	Material
09:40 +10'	Einstieg	Einstieg: Bilder zeigen und Diskussion Leitfrage: <i>Wie entsteht Schatten?</i> <b>Frage:</b> Was brauchen wir, um mit Licht und Schatten zu experimentieren? → Lichtquelle(n), undurchsichtigen Objekt, Schirm.	UG	Folien
09:50 +5'		Die Reihenfolge der Befehle <code>\time</code> , <code>\goal</code> , <code>\material</code> , <code>\method</code> und <code>\content</code> ist irrelevant. Man kann selbstverständlich auch welche auslassen. (Außer <code>\time</code> , denke ich. Ich habe es ehrlich gesagt nie ausprobiert, weil es sowieso sinnlos ist ;-)) Das Makro <code>\newblock</code> trennt die verschiedenen Unterrichtsphasen.	Baz	Foo
09:55 +10'		Im Argumenten von <code>\content</code> kann man folgende Sachen hinkriegen: Zum Beispiel einen <b>TA</b> Tafelanschrieb. Der dicke Strich am Rand macht   deutlicher, was zum TA gehört und was nicht. Die Umgebung <code>{TA}</code> nimmt auch einen optionalen Argumenten: <code>\begin{TA}[Titel]...</code> ergibt <b>TA</b> <i>TITEL</i>   Ein Tafelanschrieb mit Titel. Die Schriftart vom Titel ist gespeichert im Makro <code>\TAtitlefont</code> (default <code>\scshape\itshape</code> ). <input type="checkbox"/> für Versuchsbeschreibung und <input type="checkbox"/> für Beobachtung. Aus historischen Gründen ist es möglich, den Tafelanschrieb als <i>Makro</i> mit einem optionalen und einem obligatorischen Argumenten anzugeben, d.h. <code>\TA[Überschrift]{Irgendein Text...}</code> erzeugt <b>TA</b> <i>ÜBERSCHRIFT</i>   Irgendein Text, der nur da ist, um etwas Platz zu nehmen, um zu zeigen, wie sich das Ganze verhält, aber ohne einen besonderen tiefen Sinn. Nur Platzhalter halt. Das ist <i>sehr schlechter</i> $\text{\TeX}$ -Stil aber es funktioniert...		
10:05 +35'		Die Tabelle für die Planung ist eigentlich ein <code>{longtable}</code> , d.h. sie kann über mehrere Seiten gehen. Ich fülle hier nur Zeug ein, um Platz zu nehmen, um zu zeigen, wie es auf der folgenden Seite aussieht. (Der Tabellenkopf wird wiederholt.) Es gibt ein Makro <code>\point</code> , der eine Art „poor-person-list“ einführt: • Foo • Bar • Langer Text, um zu zeigen, dass der Text eingerückt wird, wenn er lang ist. Damit die Einrückung weg ist, muss man den Abschnitt explizit mit <code>\par</code> unterbrechen. Mit einem optionalen Parameter hat man etwas ähnlich wie <code>{description}</code> <b>Irgendwas</b> Lorem ipsum dolor sit et amet irgend ein Text, der keine besondere Bedeutung hat. (Das war die erste Idee hinter dem Makro für den Tafelanschrieb.)	EA/PA	

Zeit	Ziele	Inhalt	Methode	Material
10:40 +15'	Wdh	<p>Dem geübten Auge fällt auf, dass das Makro <code>\time</code> eigentlich eine <math>\text{T}_{\text{E}}\text{X}</math>-Primitive ist. Diese wird in der Tabelle umdefiniert, aber innerhalb von <code>\contents{...}</code> wieder hergestellt, so dass <code>\the\time</code> ergibt 1000.</p> <p>Es gibt auch eine Sternform <code>{TA*}</code> der Tafelanschrieb-Umgebung, die das fette „<b>TA</b>“ nicht schreibt und keinen optionalen Titel akzeptiert, sondern nur den dicken grauen Strich am Rand zeichnet.</p>		Foo, bar baz, bla, meh
10:55 +15'	Test	Wenn sich insgesamt keine 90 Minuten ergeben (wie in diesem Fall), gibt es eine Warnung am Ende der Tabelle.		
11:10 +2'		<p>Vorsicht: <code>{schedule}</code> ist letztendlich ein <code>{longtable}</code>: zwischen den verschiedenen Angaben <code>\material</code>, <code>\time</code> usw. sollte <i>keine leere Zeile</i> stehen. Diese wird sonst als neuer Abschnitt interpretiert und wird zu merkwürdigen Ergebnissen führen.</p> <p><b>Nachtrag</b> Das müsste mit v0.3a behoben worden sein, aber es ist trotzdem keine schlechte Idee, leere Zeilen zu vermeiden.</p>		
11:12	<b>!! 2 Minuten zu viel !!</b>			

Zu Beginn der `{schedule}` Umgebung wird erst eine neue Seite gestartet, der Satzspiegel etwas vergrößert, und eine Überschrift in der Form *Klasse — Titel — Untertitel* gedruckt. Diese Inhalte wurden in der Präambel folgendermaßen deklariert:

```
\lesson{Thema}[Möglicher Untertitel]
\class{8g}
```

Der Untertitel ist natürlich optional und kann weggelassen werden. Hätte man die Befehle in der Präambel nicht angegeben, so hätte man als Überschrift wörtlich

Klasse 0 — \lesson{Titel}[Untertitel]

bekommen (als kleine Erinnerung, wie man es verwenden soll).

Die Schulklasse ist voreingestellt auf die „0x“. Die mit `\class` definierte Klasse wird gespeichert und kann mit dem Befehl `\printclass{<*>}` wiedergegeben werden. Die Sternform gibt nur die Klassenstufe, vergleiche „8g“ und „8“. Das Makro `\classname` speichert den „Namen“ der Klasse: voreingestellt ist natürlich „Klasse“.

Standardmäßig werden Blöcke von 90 Minuten angenommen. Die Dauer eines Unterrichtsblocks kann mittels `\SetDuration` geändert werden, d.h.

```
\SetDuration{45}
```

legt grundsätzlich die Dauer eines Blocks auf 45 Minuten fest.

Im obigen Beispiel wurde die Umgebung mit dem optionalen Argumenten 1 gestartet. Das optionale Argument kann verschiedene Formen annehmen:

- Es kann eine Zahl sein, welche den Block/die Blöcke identifiziert, in der/denen die Stunde stattfindet: 1 für den ersten Block, 2 für den zweiten usw., aber auch 12 für 1. und 2., 134 für 1., 3. und 4., und alle möglichen Kombinationen.<sup>4</sup>
- Alternativ kann man explizit eine Uhrzeit angeben

```
\begin{schedule}[8:15]
```

und dann beginnt die Planung zur gegebenen Uhrzeit.

- Man kann auch eine Key-Value-Syntax verwenden. Die uninteressanten Beispiele sind

```
\begin{schedule}[start=8:15] ist dasselbe wie \begin{schedule}[8:15]
```

und

```
\begin{schedule}[block=13] ist dasselbe wie \begin{schedule}[13]
```

Es gibt aber auch *andere* Optionen, nämlich `duration=...` und `title=true/false`. Man kann also angeben

```
\begin{schedule}[start=8:00,duration=45]
```

wenn man eine Einzelstunde will, die um 8 Uhr startet. Mit der Angabe `title=false` wird die Kopfzeile nicht gedruckt. (Default ist `true`.) Natürlich ist es sinnlos, sowohl `block=...` als auch `start=...` anzugeben: das letzte gewinnt.

Der Defaultwert des optionalen Argumenten ist 1234, d.h. wenn gar kein optionales Argument angegeben wird, so bekommt man den Stundenverlauf viermal gedruckt (einmal pro Block).

---

<sup>4</sup>312 drückt erst den dritten, dann den ersten und zum Schluss den zweiten Block. Mit 111 kriegt man dreimal den ersten Block. Man muss selbst etwas mitdenken...

### 3.2 Neue Stile für die Stundenplanung definieren (NEU! v1.0.0)

Wie im obigen Beispiel gezeigt, stehen innerhalb<sup>5</sup> einer Umgebung `{schedule}` die Befehle `\time`, `\goal`, `\material`, `\method` und `\content` zur Verfügung. Natürlich will jede/r Fachleiter/in am Seminar etwas anderes haben, und so stehen Anpassungsmöglichkeiten zur Verfügung. Jede `{schedule}` wird in einem gegebenen *Stil* gesetzt. Ein Stil wird durch

```
\NewScheduleStyle{<Name>}[<relative Breite der Zeit-Spalte>]{<Spaltendefinitionen>}
```

definiert. Im ersten Argumenten von `\NewScheduleStyle` steht der Name des Stils; das zweite, optionale Argument beschreibe ich später. Im weiteren obligatorischen Argumenten muss eine Reihe von verschiedenen Deklarationen der Form

```
\DeclareColumn[<Extra-Code>]{<Marko>}{<Überschrift>}{<relative Breite>}
```

sein; jede davon legt fest

- (1) das Makro, das den Inhalt der Spalte setzt,
- (2) die entsprechende Spaltenüberschrift,
- (3) die *relative* Breite der Spalte.

Ein Beispiel: Der default-Stil ist in etwa folgendermaßen definiert:

```
\NewScheduleStyle{default}[5]{%
  \DeclareColumn{\goal}{Ziel}{6}%
  \DeclareColumn[\def\\{\newline}\let\time\TeXtime]{\content}{Inhalt}{35}%
  \DeclareColumn{\method}{Methode}{6}%
  \DeclareColumn{\material}{Material}{8}%
}
```

Es wird stets angenommen, dass die erste Tabellenspalte die Zeit beinhaltet und standardmäßig die relative Breite 1 hat: diese „Referenzbreite“ kann mit dem optionalen Argumenten geändert werden, hier im Beispiel 5.<sup>6</sup>

Was passiert dann hier genau? Mit der obigen Definition hat die Tabelle in dem default-Stil eine „Zeit“-Spalte mit Breite 5, eine „Ziel“-Spalte mit Breite 6, eine „Inhalt“-Spalte mit Breite 35, eine „Methode“-Spalte mit Breite 6 und eine „Material“-Spalte mit Breite 8. Zu Beginn der `{schedule}` wird einfach alles addiert:  $5 + 6 + 35 + 6 + 8 = 60$ . Die Klasse rechnet dann die entsprechende Breite jeder Spalte unter der Voraussetzung, die Tabelle sei so breit wie der Satzspiegel (und berücksichtigt natürlich die Tabellenlinien).

Das optionale Argument zu `\DeclareColumn` hat die folgende Funktion: Das wie oben definierte Makro `\content` speichert zuerst seinen Inhalt in einem Makro `\drc@content`. Dieses wird dann in der Tabelle an der geeigneten Stelle platziert. Da aber in der Tabelle das Makro `\\` die neue Tabellenzeile startet, kann man `\\` im Argumenten von `\content` nicht verwenden, was natürlich blöd ist. Dazu ist das optionale Argument zu `\DeclareColumn` da: es ist extra Code, das zu Beginn der Zelle kopiert wird

```
... & <extra Code>\drc@content & \drc@notes ...
```

Das heißt, in der „Inhalt“-Spalte kann man `\\` verwenden, ohne dass eine neue Tabellenzeile gestartet wird (mit dem Chaos, das dabei entstehen würde), und das Makro `\time` bekommt wieder seine ursprüngliche Definition.

Man kann somit z.B. einen anderen Stil, sagen wir *simple*, definieren:

```
\NewScheduleStyle{simple}{%
  \DeclareColumn[\def\\{\newline}\let\time\TeXtime]{\content}{Inhalt}{8}%
  \DeclareColumn{\notes}{Anmerkungen}{3}%
}
```

<sup>5</sup>Und zwar *nur* innerhalb. Außerhalb ist `\time` die bekannte  $\TeX$  Primitive und alle anderen Makros sind nicht definiert.

<sup>6</sup>Wo kommt die 5 her? Historisch. Wenn ich jetzt den default-Stil neu definieren würde, würde ich es vielleicht etwas anders machen, aber ich möchte nicht, dass plötzlich alle meine alten Dateien anders aussehen.

Wie kann man diesen Stil verwenden? Dazu gibt es zwei Möglichkeiten. Man kann entscheiden, dass dieser Stil grundsätzlich angewandt werden soll, indem man in der Präambel

```
\SetScheduleStyle{simple}
```

schreibt. Alternativ kann man im optionalen Argumenten der Umgebung `{schedule}` `style=<Name>` angeben. So ergibt zum Beispiel

```
\begin{schedule*}[style=simple,start=8:00,title=false]
\time{45}
\content{Irgendwas}
\notes{Upps}
\newblock
\time{45}
\content{Irgendwas anderes}
\notes{Mir fällt nichts ein}
\end{schedule*}
```

das folgende Ergebnis:

Zeit	Inhalt	Anmerkungen
08:00 +45'	Irgendwas	Upps
08:45 +45'	Irgendwas anderes	Mir fällt nichts ein
09:30		

Die hier verwendete Sternform `{schedule*}` startet keine neue Seite<sup>7</sup> und ändert den Satzspiegel nicht, verhält sich sonst in allem wie `{schedule}`.<sup>8</sup>

Natürlich kann man auch die Standardeinstellung ändern. Neben `\NewScheduleStyle` existiert `\RenewScheduleStyle`, und man könnte mit

```
\RenewScheduleStyle{default}{%
\DeclareColumn[\vskip-\baselineskip\def\\{\newline}\let\time\TeXtime]%
{\content}{Inhalt}{8}%
\DeclareColumn{\notes}{Anmerkungen}{3}%
}
```

den Defaultstil umdefinieren. Es wäre aber auch denkbar, dass man den Defaultstil umdefinieren aber nicht unbedingt verlieren möchte: Man kann auch Stile kopieren. Mit

```
\CopyScheduleStyle{<neuer Stil>}{<alter Stil>}
```

macht man eine Kopie eines existierenden Stils, der dann geändert werden kann.

### 3.3 Uhrzeiten

Die Uhrzeiten der Blöcke sind zuerst festgelegt auf 07:50, 09:40, 11:25 und 14:00 (die Uhrzeiten meiner Schule halt...), und die Dauer eines Unterrichtsblocks ist wie schon gesagt auf 90 Minuten initialisiert. Wie bereits beschrieben, können die Startzeit und die Unterrichtsdauer einer einzelnen Umgebung `{schedule}` mit den Optionen `start=...` und `duration=...` geändert werden. Die Startzeiten der Blöcke können im Allgemeinen mit Hilfe von `\SetBlockStart` festgelegt werden. Vordefiniert sind

<sup>7</sup>Na ja, manchmal schon. `{schedule}` ist letztendlich ein `{longtable}` und es kann trotzdem eigenmächtig entscheiden, eine neue Seite zu starten.

<sup>8</sup>Beim default-Stil ist aber die „Methode“-Spalte zu eng, und man bekommt entsprechend Warnungen. Das ist wieder ein Relikt aus der Entstehungsgeschichte.

```

\SetBlockStart{1}{07:50}
\SetBlockStart{2}{09:40}
\SetBlockStart{3}{11:25}
\SetBlockStart{4}{14:00}
\SetBlockStart{5}{15:40}

```

und die kann man natürlich nach Belieben umdefinieren.

## 4 Individuelle Anpassungen speichern

Es ist mir klar, dass jede Schule andere Uhrzeiten hat; und es ist mir auch klar, dass es lästig wäre, eigene `\SetDuration` und `\SetBlockStart` in jede Datei zu schreiben. Natürlich kann man alle Einstellungen in eine Datei `meinmacros.tex` speichern und dann `\input{meinmacros}` verwenden (oder—elegant—in einem Paket `meineschule.sty` und dann `\usepackage{meineschule}`), aber die Klasse bietet zwei Möglichkeiten an, eigene Anpassungen zu definieren.

### 4.1 Hauptkonfigurationsdatei

Zuerst überprüft die Klasse immer, ob eine „Hauptkonfigurationsdatei“ namens `drschool.cfg` existiert, und lädt diese gegebenenfalls. Nehmen wir daher an, Frau Maier möchte ihre Einstellungen ein für alle Mal speichern. Insbesondere hat sie in ihrer Schule Blöcke von 55 Minuten mit einer Pause zwischen dritten und vierten, sowie zwischen fünften und sechsten. Außerdem möchte sie eher den Stil `simple` für die Unterrichtsplanung verwenden aber den `default`-Stil nicht überschreiben, denn er kommt bei Unterrichtsbesuchen ihres Wasauchimmer-Fachleiters gut an. Darüber hinaus möchte Frau Meier Times New Roman als Schriftart verwenden.<sup>9</sup> Um das alles zu machen, schreibt sie die Datei `drschool.cfg`

```

\ProvidesFile{drschool.cfg}
\SetLogo{example-image-a}
\SetDuration{55}
\SetBlockStart{1}{8:00}
\SetBlockStart{2}{8:55}
\SetBlockStart{3}{9:50}
\SetBlockStart{4}{11:00}
\SetBlockStart{5}{11:55}
\SetBlockStart{6}{14:00}
\SetBlockStart{7}{14:55}
\SetBlockStart{8}{15:50}
\NewScheduleStyle{simple}{%
  \DeclareColumn[\vskip-\baselineskip\def\{\}\newline}\let\time\TeXtime]%
  {\content}{Inhalt}{8}%
  \DeclareColumn{\notes}{Anmerkungen}{3}%
}
\SetScheduleStyle{simple}
\NoFonts
\RequirePackage{newtxtext,newtxmath}

```

und platziert sie dort, wo  $\TeX$  sie finden kann (am Besten da, wo sich auch `drschool.cls` befindet). Die erste Zeile (`\ProvidesFile`) ist nicht notwendig aber wärmstens empfohlen. Die Bedeutung der zweiten Zeile (`\SetLogo`) wird später bei der Herstellung von Klassenarbeiten oder Tests erklärt. Und da Frau Maier eine Times-Schrift verwenden will, schreibt sie `\NoFonts` in der Konfigurationsdatei, bevor sie die Pakete `newtxtext` und `newtxmath` lädt. Natürlich kann Frau Mayer in dieser Datei auch alle ihre

---

<sup>9</sup>Bleargh...



Lieblingspakete und persönlichen Definitionen schreiben, und muss somit eine ewig lange Präambel nicht in jede ihrer Dateien kopieren.

## 4.2 Schulkonfigurationsdatei(en)

Hauptsächlich aus Kompatibilitätsgründen existiert eine weitere Möglichkeit, persönliche Konfigurationen zu Laden, nämlich mit Hilfe einer Schulkonfigurationsdatei. Sagen wir, Frau Meyer unterrichtet in zwei verschiedenen Schulen, die verschiedene Uhrzeiten haben. Dann kann sie für jede Schule eine Datei mit Endung `.sco` (steht für *school class option*) schreiben, die auch dort platziert wird, wo  $\text{T}_{\text{E}}\text{X}$  sie finden kann. Sie sieht im Grunde wie die Hauptkonfigurationsdatei aus

```
\ProvidesFile{MeineSchuleA.sco}
% alles, was man will
```

muss aber in der Präambel mit

```
\LoadSchoolOptionFile{MeineSchuleA}
```

explizit geladen werden (*ohne* `.sco` Endung). Ähnliches kann Frau Maier<sup>10</sup> mit ihrer anderen Schule machen.

In so einem Fall bietet sich auch eine Kombination aus beiden Methoden an: In der Hauptkonfigurationsdatei `drschool.cfg` wird der Code geschrieben, der für alle Schulen gelten soll, und in die beiden Schulkonfigurationsdateien kommen die schulbezogenen Einstellungen. Die Hauptkonfigurationsdatei `drschool.cfg` wird am Ende der Klasse geladen, während die `sco`-Dateien in der Präambel geladen werden (genau da, wo `\LoadSchoolOptionFile` verwendet wird): Eine Schulkonfigurationsdatei kann daher Befehle der Hauptkonfigurationsdatei überschreiben.

Vorsicht: Es kann pro  $\text{T}_{\text{E}}\text{X}$ -Datei *nur eine* `sco`-Datei geladen werden!

## 5 Arbeitsblätter, Klassenarbeiten & Co.

Es folgen nur einige Beispiele für die verschiedenen Arten von Arbeitsblättern, die mit der Klasse erstellt werden können:

- Zuerst kommen mehrere Beispiele der Umgebung `{worksheet}`, die ein „normales“ Arbeitsblatt erstellt. Ein Arbeitsblatt wird zweimal gedruckt: einmal mit und einmal ohne Lösung. Die folgenden Beispiele hätten natürlich auch in einer einzelnen `{worksheet}` Umgebung gesetzt werden können; sie aber in mehreren Beispielen aufzuteilen, hat den Vorteil, dass man die Variante mit Lösung nicht erst zehn Seiten später sieht.
- Es kommen dann einige Beispiele von Umgebungen der Form `{print<N>}`, (wobei  $\langle N \rangle$  ist die Zahl 2, 3, oder 4), die ihren Inhalt mehrmals auf einer Seite drücken.
- Es folgen dann Beispiele der Umgebung `{cluecards}`, mit der man z.B. Lösungskärtchen drucken kann.
- Zum Schluss kommt ein Beispiel für die Umgebung `{test}` für Tests/Klassenarbeiten. Die ist weitgehend ähnlich zu einem Arbeitsblatt aber mit der Möglichkeiten, Punktzahlen anzugeben.

---

<sup>10</sup>Ist es jemandem aufgefallen, dass ich alle möglichen Kombinationen (ai/ai/ay/ey) verwendet habe? :-P



## TITEL DES ARBEITSBLATTS

**!! Caveat emptor !!**

Arbeitsblätter werden zweimal abgedruckt: einmal ohne Lösung(en), und einmal mit. Dafür muss die `{worksheet}` Umgebung ihren gesamten Inhalt sammeln, was `\catcodes` einfriert! (Das ist ein ziemlich T<sub>E</sub>Xnisches Detail: wer das versteht, weiß damit umzugehen; wer's nicht versteht, wird vermutlich keine Probleme haben.)

Arbeitsblätter werden mit Hilfe der Umgebung `{worksheet}` hergestellt. Der Titel des Arbeitsblatts kann als optionales Argument zu `\begin{worksheet}` gegeben werden. Das optionale Argument kann allerdings auch eine Key-Value-Liste sein. Mögliche Optionen sind:

**title**=*<Titel>* legt den Titel fest;

**date**=*<Ein-Aus-Wert>* legt fest, ob die Überschrift „Datum“ in der Kopfzeile angezeigt wird;

**name**=*<Ein-Aus-Wert>* legt fest, ob der Name in der Kopfzeile geschrieben werden soll;

**fontsize**=*<Schriftgröße>* ändert die Schriftgröße im Arbeitsblatt;

**geometry**=*<Optionen>* gibt die *<Optionen>* an das geometry Paket weiter.

(Ein *<Ein-Aus-Wert>* ist wie gewohnt `true/yes/on` oder `false/no/off`.) Im Grunde ist also

```
\begin{worksheet}[Titel des Arbeitsblatts]
```

in etwa dasselbe wie

```
\begin{worksheet}[title=Titel des Arbeitsblatts,%
                    date=true,name=false]
```

Wenn man die Standardeinstellungen ändern will, dann stehen die Befehle `\SetWorksheetOptions` und `\AddWorksheetOptions` zur Verfügung. Wenn z.B. alle Arbeitsblätter grundsätzlich den Platz für den Namen haben sollen, so kann man

```
\SetWorksheetOptions{name=true,date=true}
```

in der Präambel oder in einer Konfigurationsdatei schreiben.

**Aufgabe 1 (Titel)**

Eine Aufgabe wird mittels `\exercise` gestartet. Ein Titel kann als optionales Argument gegeben werden: `\exercise[Titel]`. Innerhalb einer Aufgabe können einzelne Teilaufgaben mit Hilfe der Umgebung `{questions}` gestellt werden:

**a.** Bla bla bla bla

*Hinweis:* Hier ein kleiner Hinweis.

**b.** Bla bla bla bla bla bla

Die Umgebung kann unterbrochen werden; wenn sie nochmal startet, läuft der Zähler weiter:

**\*c.** Noch eine.

**d.** Und noch eine.

Wie man hier sieht, gibt es auch eine Sternform `\question*`: Links vom Buchstabe erscheint dann eine Markierung, die in dem Makro `\starredquestionmark` gespeichert ist (default eben `*`).

*Aufpassen!* Ein Hinweis beginnt mit `\hintname` (default „Hinweis“) gefolgt vom `\hintsep` (default „:“). Man kann diese Makros umdefinieren oder man kann das optional Argument zur `{hint}`-Umgebung verwenden.

**Aufgabe 2 (Multiple-Choice-Quiz)**

Es stehen die Symbole `\checkbox` (☐) und `\radiobutton` (☐) zur Verfügung. Beide haben eine Sternform `\checkbox*` (☒) und `\radiobutton*` (☒), die die richtige (bzw. anzukreuzende) Antwort in der Variante mit Lösung druckt.

Auf diesen Symbolen bauen zwei Umgebungen für Multiple-Choice-Aufgaben. Eine verwendet die runden Radiobuttons, die andere dagegen die eckigen Auswahlkästen. Typischerweise werden die Radiobuttons verwendet, wenn *nur eine* Lösung korrekt ist, und die Auswahlkästen, wenn mehrere Antworten möglich sind. Um das Ganze schön verwirrend zu machen, habe ich mich entschieden, für die beiden Umgebungen die *englischen* Namen zu verwenden. Was im Deutschen als „Single-Choice“

bezeichnet wird, ist auf Englisch „multiple choice“; und was auf Deutsch „Multiple-Choice“ ist, heißt auf Englisch „multiple response“.

Entsprechend ihrer gemeinten Nutzung geben beide Umgebungen eine Warnung, wenn eine Frage keine als richtig markierte Antwort hat. Die Umgebung `{multchoice}` gibt darüber hinaus eine Warnung, wenn eine Frage mehr als eine richtige Antwort hat. Beide Umgebungen verwenden immer denselben Zähler wie auch die Umgebungen `{questions}` und `{questions*}`, und können daher zusammen in einer Aufgabe verwendet werden. (Ob das sinnvoll ist, ist eine andere Geschichte...)

Die Umgebung `{multchoice}` verwendet Radiobuttons

- a. Fragen werden durch `\question` gestellt,
  - ☐ mögliche Antworten durch `\choice`.
  - ☐ Falsch.
  - ☐ Die richtige Wahl wird im Quellcode als `\choice*` gegeben.
- \*b. Noch eine Frage. Auch hier besteht die Möglichkeit, Fragen mit `*` zu markieren.
  - ☐ Falsch.
  - ☐ Richtig.
  - ☐ Nutzt man auch hier `\choice*`, so kriegt man einen Fehler.

während `{multresponse}` Auswahlkästen benutzt und mehrere Antworten zulässt:

- c. Frage
  - ☐ Richtig.
  - ☐ Falsch.
  - ☐ Auch richtig.
- d. Noch eine:
  - ☐ Falsch.
  - ☐ Richtig.
  - ☐ Falsch.

### Aufgabe 3 (Kommas als Dezimaltrenner)

Das Paket `icomma` wird automatisch geladen. Man kann also das Komma als Dezimaltrenner im Mathe-Modus verwenden, ohne unangenehme Leerräume zu kriegen: 2,5 cm. Will man allerdings das Komma als Interpunktionszeichen haben, dann muss man im Quellcode einen Leerraum lassen: Man vergleiche

`$(a,b)$` →  $(a,b)$

`$(a, b)$` →  $(a, b)$

## TITEL DES ARBEITSBLATTS

### !! Caveat emptor !!

Arbeitsblätter werden zweimal abgedruckt: einmal ohne Lösung(en), und einmal mit. Dafür muss die `{worksheet}` Umgebung ihren gesamten Inhalt sammeln, was `\catcodes` einfriert! (Das ist ein ziemlich T<sub>E</sub>Xnisches Detail: wer das versteht, weiß damit umzugehen; wer's nicht versteht, wird vermutlich keine Probleme haben.)

Arbeitsblätter werden mit Hilfe der Umgebung `{worksheet}` hergestellt. Der Titel des Arbeitsblatts kann als optionales Argument zu `\begin{worksheet}` gegeben werden. Das optionale Argument kann allerdings auch eine Key-Value-Liste sein. Mögliche Optionen sind:

**title**=*⟨Titel⟩* legt den Titel fest;

**date**=*⟨Ein-Aus-Wert⟩* legt fest, ob die Überschrift „Datum“ in der Kopfzeile angezeigt wird;

**name**=*⟨Ein-Aus-Wert⟩* legt fest, ob der Name in der Kopfzeile geschrieben werden soll;

**fontsize**=*⟨Schriftgröße⟩* ändert die Schriftgröße im Arbeitsblatt;

**geometry**=*⟨Optionen⟩* gibt die *⟨Optionen⟩* an das geometry Paket weiter.

(Ein *⟨Ein-Aus-Wert⟩* ist wie gewohnt true/yes/on oder false/no/off.) Im Grunde ist also

```
\begin{worksheet}[Titel des Arbeitsblatts]
```

in etwa dasselbe wie

```
\begin{worksheet}[title=Titel des Arbeitsblatts,%  
date=true,name=false]
```

Wenn man die Standardeinstellungen ändern will, dann stehen die Befehle `\SetWorksheetOptions` und `\AddWorksheetOptions` zur Verfügung. Wenn z.B. alle Arbeitsblätter grundsätzlich den Platz für den Namen haben sollen, so kann man

```
\SetWorksheetOptions{name=true,date=true}
```

in der Präambel oder in einer Konfigurationsdatei schreiben.

### Aufgabe 1 (Titel)

Eine Aufgabe wird mittels `\exercise` gestartet. Ein Titel kann als optionales Argument gegeben werden: `\exercise[Titel]`. Innerhalb einer Aufgabe können einzelne Teilaufgaben mit Hilfe der Umgebung `{questions}` gestellt werden:

**a.** Bla bla bla bla

*Hinweis:* Hier ein kleiner Hinweis.

**b.** Bla bla bla bla bla bla bla

Die Umgebung kann unterbrochen werden; wenn sie nochmal startet, läuft der Zähler weiter:

**\*c.** Noch eine.

**d.** Und noch eine.

Wie man hier sieht, gibt es auch eine Sternform `\question*`: Links vom Buchstabe erscheint dann eine Markierung, die in dem Makro `\starredquestionmark` gespeichert ist (default eben \*).

*Aufpassen!* Ein Hinweis beginnt mit `\hintname` (default „Hinweis“) gefolgt vom `\hintsep` (default „:“). Man kann diese Makros umdefinieren oder man kann das optional Argument zur `{hint}`-Umgebung verwenden.

### Lösung

Lösungen werden innerhalb der *Umgebung* `{solution}` geschrieben. Falls es `{questions}` gab, kann man entsprechend Teilantworten mittels `\answer` angeben.

**a.** Lösung der Teilaufgabe Lösung der Teilaufgabe

$$x = 2$$

Lösung der Teilaufgabe Lösung der Teilaufgabe Lösung der Teilaufgabe.

**b.** Lösung der Teilaufgabe Lösung der Teilaufgabe Lösung der Teilaufgabe Lösung der Teilaufgabe Lösung der Teilaufgabe.

Lösung

\*c. Wurde eine Frage mit der Sternform `\question*` angegeben, so erscheint der Stern automatisch bei der entsprechenden Antwort.

Am Ende der `{solution}` Umgebung wird überprüft, ob die Anzahl der `\answers` der Anzahl der `\questions` entspricht; falls nicht (wie in diesem Fall) wird eine Warnung herausgegeben.

## Aufgabe 2 (Multiple-Choice-Quiz)

Es stehen die Symbole `\checkbox` (☐) und `\radiobutton` (☐) zur Verfügung. Beide haben eine Sternform `\checkbox*` (☒) und `\radiobutton*` (☒), die die richtige (bzw. anzukreuzende) Antwort in der Variante mit Lösung druckt.

Auf diesen Symbolen bauen zwei Umgebungen für Multiple-Choice-Aufgaben. Eine verwendet die runden Radiobuttons, die andere dagegen die eckigen Auswahlkästen. Typischerweise werden die Radiobuttons verwendet, wenn *nur eine* Lösung korrekt ist, und die Auswahlkästen, wenn mehrere Antworten möglich sind. Um das Ganze schön verwirrend zu machen, habe ich mich entschieden, für die beiden Umgebungen die *englischen* Namen zu verwenden. Was im Deutschen als „Single-Choice“ bezeichnet wird, ist auf Englisch „multiple choice“; und was auf Deutsch „Multiple-Choice“ ist, heißt auf Englisch „multiple response“.

Entsprechend ihrer gemeinten Nutzung geben beide Umgebungen eine Warnung, wenn eine Frage keine als richtig markierte Antwort hat. Die Umgebung `{multchoice}` gibt darüber hinaus eine Warnung, wenn eine Frage mehr als eine richtige Antwort hat. Beide Umgebungen verwenden immer denselben Zähler wie auch die Umgebungen `{questions}` und `{questions*}`, und können daher zusammen in einer Aufgabe verwendet werden. (Ob das sinnvoll ist, ist eine andere Geschichte...)

Die Umgebung `{multchoice}` verwendet Radiobuttons

- a. Fragen werden durch `\question` gestellt,
- ☐ mögliche Antworten durch `\choice`.
  - ☐ Falsch.
  - ☒ Die richtige Wahl wird im Quellcode als `\choice*` gegeben.
- \*b. Noch eine Frage. Auch hier besteht die Möglichkeit, Fragen mit `*` zu markieren.
- ☐ Falsch.
  - ☒ Richtig.
  - ☐ Nutzt man auch hier `\choice*`, so kriegt man einen Fehler.

während `{multresponse}` Auswahlkästen benutzt und mehrere Antworten zulässt:

- c. Frage
- ☒ Richtig.
  - ☐ Falsch.
  - ☒ Auch richtig.
- d. Noch eine:
- ☐ Falsch.
  - ☒ Richtig.
  - ☐ Falsch.

## Aufgabe 3 (Kommas als Dezimaltrenner)

Das Paket `icomma` wird automatisch geladen. Man kann also das Komma als Dezimaltrenner im Mathe-Modus verwenden, ohne unangenehme Leerräume zu kriegen: 2,5 cm. Will man allerdings das Komma als Interpunktionszeichen haben, dann muss man im Quellcode einen Leerraum lassen: Man vergleiche

`$(a,b)$`  $\rightarrow (a,b)$

`$(a, _b)$`  $\rightarrow (a, b)$

Datum:

## HORIZONTALE AUFLISTUNGEN

In Mathe-Büchern werden üblicherweise Teilaufgaben horizontal aufgelistet. Ankreuzaufgaben mit relativ kurzen Texten könnten auch platzsparender sein und auf Spalten verteilt werden. Zu diesem Zwecke gibt es jeweils eine Sternform `{questions*}`, `{multresponse*}` und `{multchoice*}`: diese erwarten die Anzahl der Spalten als obligatorisches Argument.

### Aufgabe 1 (Kurzfragen wie im Lambacher-Schweizer)

Zum Beispiel, mit `\begin{questions*}{4}` bekommt man

- |        |         |        |         |
|--------|---------|--------|---------|
| a. AAA | *b. BBB | c. CCC | *d. DDD |
| e. EEE | f. FFF  | g. GGG |         |

Auch diese Umgebung kann unterbrochen werden, und eine weitere Umgebung zählt einfach weiter:

- |           |          |          |
|-----------|----------|----------|
| h. HHHHHH | i. IIIII | j. JJJJJ |
|-----------|----------|----------|

Die Sternform `\question*` funktioniert selbstverständlich auch hier.

Sowohl `{questions}` als auch `{questions*}` nutzen denselben Zähler, so dass beide in einer Aufgabe zusammen verwendet werden können:

- k. Text einer „normalen“ Aufgabe, die in einer gewöhnliche Liste gesetzt wird.  
l. Text einer weiteren „normalen“ Aufgabe, die in einer gewöhnliche Liste gesetzt wird.

### Aufgabe 2 (Ankreuzaufgaben)

Dasselbe funktioniert mit `{multchoice*}`

- |                         |                         |                         |
|-------------------------|-------------------------|-------------------------|
| a. Foo                  | b. Foo                  | c. Foo                  |
| <input type="radio"/> A | <input type="radio"/> A | <input type="radio"/> A |
| <input type="radio"/> B | <input type="radio"/> B | <input type="radio"/> B |
| <input type="radio"/> C | <input type="radio"/> C | <input type="radio"/> C |

sowie mit `{multresponse*}`

- |                            |                            |                            |
|----------------------------|----------------------------|----------------------------|
| d. Foo                     | e. Foo                     | f. Foo                     |
| <input type="checkbox"/> A | <input type="checkbox"/> A | <input type="checkbox"/> A |
| <input type="checkbox"/> B | <input type="checkbox"/> B | <input type="checkbox"/> B |
| <input type="checkbox"/> C | <input type="checkbox"/> C | <input type="checkbox"/> C |
| g. Foo                     | h. Foo                     |                            |
| <input type="checkbox"/> A | <input type="checkbox"/> A |                            |
| <input type="checkbox"/> B | <input type="checkbox"/> B |                            |
| <input type="checkbox"/> C | <input type="checkbox"/> C |                            |

**Bemerkung:** Alle diese horizontalen Auflistungen sind letztendlich Tabellen, und entsprechend kann zwischen den Zeilen *kein Seitenumbruch* erfolgen. Ich habe auch nicht vor, in nächster Zeit die Möglichkeit eines Seitenumbruchs zuzulassen, denn ich finde sie nicht wirklich sinnvoll. (Na gut, bei Ankreuzfragen vielleicht schon...)

## HORIZONTALE AUFLISTUNGEN

In Mathe-Büchern werden üblicherweise Teilaufgaben horizontal aufgelistet. Ankreuzaufgaben mit relativ kurzen Texten könnten auch platzsparender sein und auf Spalten verteilt werden. Zu diesem Zwecke gibt es jeweils eine Sternform `{questions*}`, `{multresponse*}` und `{multchoice*}`: diese erwarten die Anzahl der Spalten als obligatorisches Argument.

### Aufgabe 1 (Kurzfragen wie im Lambacher-Schweizer)

Zum Beispiel, mit `\begin{questions*}{4}` bekommt man

- |        |         |        |         |
|--------|---------|--------|---------|
| a. AAA | *b. BBB | c. CCC | *d. DDD |
| e. EEE | f. FFF  | g. GGG |         |

Auch diese Umgebung kann unterbrochen werden, und eine weitere Umgebung zählt einfach weiter:

- |           |           |           |
|-----------|-----------|-----------|
| h. HHHHHH | i. IIIIII | j. JJJJJJ |
|-----------|-----------|-----------|

Die Sternform `\question*` funktioniert selbstverständlich auch hier.

Sowohl `{questions}` als auch `{questions*}` nutzen denselben Zähler, so dass beide in einer Aufgabe zusammen verwendet werden können:

- k. Text einer „normalen“ Aufgabe, die in einer gewöhnliche Liste gesetzt wird.  
l. Text einer weiteren „normalen“ Aufgabe, die in einer gewöhnliche Liste gesetzt wird.

### Aufgabe 2 (Ankreuzaufgaben)

Dasselbe funktioniert mit `{multchoice*}`

- |                                    |                                    |                                    |
|------------------------------------|------------------------------------|------------------------------------|
| a. Foo                             | b. Foo                             | c. Foo                             |
| <input checked="" type="radio"/> A | <input type="radio"/> A            | <input type="radio"/> A            |
| <input type="radio"/> B            | <input type="radio"/> B            | <input checked="" type="radio"/> B |
| <input type="radio"/> C            | <input checked="" type="radio"/> C | <input type="radio"/> C            |

sowie mit `{multresponse*}`

- |                                       |                                       |                                       |
|---------------------------------------|---------------------------------------|---------------------------------------|
| d. Foo                                | e. Foo                                | f. Foo                                |
| <input checked="" type="checkbox"/> A | <input type="checkbox"/> A            | <input checked="" type="checkbox"/> A |
| <input type="checkbox"/> B            | <input type="checkbox"/> B            | <input checked="" type="checkbox"/> B |
| <input checked="" type="checkbox"/> C | <input checked="" type="checkbox"/> C | <input type="checkbox"/> C            |
| g. Foo                                | h. Foo                                |                                       |
| <input type="checkbox"/> A            | <input type="checkbox"/> A            |                                       |
| <input checked="" type="checkbox"/> B | <input checked="" type="checkbox"/> B |                                       |
| <input checked="" type="checkbox"/> C | <input type="checkbox"/> C            |                                       |

**Bemerkung:** Alle diese horizontalen Auflistungen sind letztendlich Tabellen, und entsprechend kann zwischen den Zeilen *kein Seitenumbruch* erfolgen. Ich habe auch nicht vor, in nächster Zeit die Möglichkeit eines Seitenumbruchs zuzulassen, denn ich finde sie nicht wirklich sinnvoll. (Na gut, bei Ankreuzfragen vielleicht schon...)



Datum:





## SCHWIERIGKEITSSYMBOLS UND WAHR-FALSCH-TABELLEN

### ● Aufgabe 1

Schwere Aufgaben können mit `\hard\exercise` erzeugt werden. Die Schwierigkeitssymbole sind denen aus dem Lambacher-Schweizer nachempfunden.

### ● Aufgabe 2 (Bla)

Mittelschwere Aufgaben kommen aus `\medium\exercise[Bla]`.

Zur Verfügung stehen auch die Symbole `\calculator`  und `\nocalculator` . Diese nehmen TikZ-Optionen als optionales Argument: `\calculator[scale=1.5]` →  `\nocalculator[baseline]` → 

(Standardmäßig liegen die Symbole etwas unter der Baseline).

### ○ Aufgabe 3

Eine einfache Aufgabe kriegt man mit `\easy\exercise`.

Bitte beachten: Die drei Makros `\hard`, `\medium` und `\easy` dürfen *nur* vor dem Befehl `\exercise` verwendet werden. Der Code

```
\hard\SomeOtherMacro
```

wird einen Fehler erzeugen. Will man die Symbole irgendwo haben, so gibt es dafür die Befehle `\easysymbol` (○), `\mediumsymbol` (●), und `\hardsymbol` (●).

### Aufgabe 4 (Wahr-Falsch-Tabelle)

Eine wahr/falsch Tabelle wird mittels der Umgebung `{TF}` erzeugt: Dies ist im Grunde eine `{tabularx}`, und jede Aussage muss mit `\true` oder `\false` beendet werden. Neue Zeilen werden automatisch hinzugefügt (es sind also keine `\\` nötig), horizontale Linien (mit `\hline` oder `\midrule`) sind nach Geschmack natürlich möglich.

	wahr	falsch
Aussage 1	○	○
Aussage 2, die sehr sehr lang ist, so dass der Text mehr als eine Zeile braucht, um gesetzt zu werden	○	○
Aussage 3	○	○
Aussage 4	○	○

Natürlich kann man die ganzen `\midrules` (oder `\hlines`, wenn man will) weglassen, wenn man sie nicht will... Die Gesamtbreite der Tabelle kann als optionaler Parameter angegeben werden (default `\linewidth`):

	wahr	falsch
Aussage A	○	○
Aussage B	○	○





## SCHWIERIGKEITSSYMBOLS UND WAHR-FALSCH-TABELLEN

### ● Aufgabe 1

Schwere Aufgaben können mit `\hard\exercise` erzeugt werden. Die Schwierigkeitssymbole sind denen aus dem Lambacher-Schweizer nachempfunden.

### ● Aufgabe 2 (Bla)

Mittelschwere Aufgaben kommen aus `\medium\exercise[Bla]`.

Zur Verfügung stehen auch die Symbole `\calculator`  und `\nocalculator` . Diese nehmen TikZ-Optionen als optionales Argument: `\calculator[scale=1.5]` →  `\nocalculator[baseline]` → 

(Standardmäßig liegen die Symbole etwas unter der Baseline).

### ○ Aufgabe 3

Eine einfache Aufgabe kriegt man mit `\easy\exercise`.

Bitte beachten: Die drei Makros `\hard`, `\medium` und `\easy` dürfen *nur* vor dem Befehl `\exercise` verwendet werden. Der Code

```
\hard\SomeOtherMacro
```

wird einen Fehler erzeugen. Will man die Symbole irgendwo haben, so gibt es dafür die Befehle `\easysymbol` (○), `\mediumsymbol` (●), und `\hardsymbol` (●).

### Aufgabe 4 (Wahr-Falsch-Tabelle)

Eine wahr/falsch Tabelle wird mittels der Umgebung `{TF}` erzeugt: Dies ist im Grunde eine `{tabularx}`, und jede Aussage muss mit `\true` oder `\false` beendet werden. Neue Zeilen werden automatisch hinzugefügt (es sind also keine `\\` nötig), horizontale Linien (mit `\hline` oder `\midrule`) sind nach Geschmack natürlich möglich.

	wahr	falsch
Aussage 1	●	○
Aussage 2, die sehr sehr lang ist, so dass der Text mehr als eine Zeile braucht, um gesetzt zu werden	○	●
Aussage 3	○	●
Aussage 4	●	○

Natürlich kann man die ganzen `\midrules` (oder `\hlines`, wenn man will) weglassen, wenn man sie nicht will... Die Gesamtbreite der Tabelle kann als optionaler Parameter angegeben werden (default `\linewidth`):

	wahr	falsch
Aussage A	●	○
Aussage B	○	●

Datum:

## LÜCKENTEXTE UND ZUSATZAUFGABEN

### Aufgabe 1 (Einfache Lückentexte)

Platz für Lückentexte wird mit dem Makro `\fillhere` eingeführt: \_\_\_\_\_. Standardmäßig ist der Strich zweimal so lang wie der gesetzte Inhalt (eine Art Handschriftkorrektur). Mit einem optionalen Parameter kann dieser „Streckfaktor“ geändert werden. Alternativ kann man als optionalen Parameter eine explizite Länge angeben, und dann wird diese verwendet:

`\fillhere{hallo}` → \_\_\_\_\_  
`\fillhere[2]{hallo}` → \_\_\_\_\_ (dasselbe wie oben)  
`\fillhere[2.5]{hallo}` → \_\_\_\_\_ (Dezimalwerte sind auch möglich)  
`\fillhere[2,5]{hallo}` → \_\_\_\_\_ (Punkt oder Komma ist egal)  
`\fillhere[3cm]{hallo}` → \_\_\_\_\_ (explizite Länge geht auch)

Es gibt auch eine Sternform `\fillhere*{...}`: \_\_\_\_\_.  
Der erzeugte Strich reicht bis zur Ende der aktuellen Zeile. Es gibt absolut *keine* Kontrolle darüber, dass der Text reinpasst. Es könnte also so etwas passieren: \_\_\_\_\_

### Aufgabe 2 (Die Umgebung {cloze})

Wie man in der vorausgehenden \_\_\_\_\_ sieht, kann die \_\_\_\_\_ der \_\_\_\_\_ möglicherweise mit den Buchstaben der unterliegenden \_\_\_\_\_ kollidieren.

Außerdem ist es mit festen \_\_\_\_\_ sehr schwierig, einen bündig ausgerichteten Text hinzukriegen.

Aus diesem Zweck steht die Umgebung `{cloze}` zur Verfügung. Im Grunde ist es eine `{flushleft}` Umgebung mit einem etwas erhöhten \_\_\_\_\_. Dieser wird standardmäßig um einen Faktor 1.4 erhöht, was mit einem optionalen Parameter geändert werden kann. Für einen \_\_\_\_\_ Zeilenabstand kann man z.B. `\begin{cloze}[1.6]` verwenden.

### Aufgabe 3 (Lückentexte als graue Boxen)

Das Makro `\fillhere` gibt ihren Inhalt immer im Textmodus wieder. Für den Mathe-Modus ist es eh nicht geeignet, da der Unterstrich in Formeln missverstanden werden könnte. Aus diesem Grunde gibt es für Mathe-Ausdrücke (aber es funktioniert natürlich auch im Text) das Makro `\fillbox`: Dies druckt einen grauen Kasten, der standardmäßig auch zweimal so breit als die „natürliche“ Größe des Textes ist; genau wie bei `\fillhere` kann der Streckfaktor mit Hilfe des optionalen Argumenten geändert werden bzw. explizit als Länge deklariert werden: siehe \_\_\_\_\_, oder siehe \_\_\_\_\_. In Gegensatz zu `\fillhere` skaliert allerdings `\fillbox` in Mathe-Modus

$$3^{\blacksquare} = 9, \quad \blacksquare^{1/2} = 7$$

und ist somit für mathematische Ausdrücke besser geeignet.

### Aufgabe 4 (Karierte Felder)

Noch etwas Nützliches: Der Befehl `\grid<(x-dimen),(y-dimen)>{...}` erzeugt ein Kastenfeld, dessen Inhalt nur in der Lösung angezeigt wird. Die Baseline der ersten Zeile ist dieselbe des umgebenden Textes. Zum Beispiel erzeugt `\grid(2,1){Text hier drin.}` dies: 


 ein Kastenfeld 2 cm breit

und 1 cm hoch. Man kann auch explizite Größen angeben, aber Vorsicht! Das Gitter hat 5mm-Schritte, und wenn man kein Vielfaches eines halben Zentimeters angibt, passiert dies: 

--	--	--	--	--	--

.

Man kann `\grid` auch ohne explizite Maße verwenden: das Kastenfeld ist dann fast so breit wie die Textbreite (trunkiert auf halbe Zentimeter) und 3,5 cm hoch:



## LÜCKENTEXTE UND ZUSATZAUFGABEN

### Aufgabe 1 (Einfache Lückentexte)

Platz für Lückentexte wird mit dem Makro `\fillhere` eingeführt: ein Wort. Standardmäßig ist der Strich zweimal so lang wie der gesetzte Inhalt (eine Art Handschriftkorrektur). Mit einem optionalen Parameter kann dieser „Streckfaktor“ geändert werden. Alternativ kann man als optionalen Parameter eine explizite Länge angeben, und dann wird diese verwendet:

`\fillhere{hallo}` → hallo  
`\fillhere[2]{hallo}` → hallo (dasselbe wie oben)  
`\fillhere[2.5]{hallo}` → hallo (Dezimalwerte sind auch möglich)  
`\fillhere[2,5]{hallo}` → hallo (Punkt oder Komma ist egal)  
`\fillhere[3cm]{hallo}` → hallo (explizite Länge geht auch)

Es gibt auch eine Sternform `\fillhere*{...}`: bla bla

Der erzeugte Strich reicht bis zur Ende der aktuellen Zeile. Es gibt absolut *keine* Kontrolle darüber, dass der Text reinpasst. Es könnte also so etwas passieren: ein sehr langer Text, der so lange ist, dass er eigentlich aus der

### Aufgabe 2 (Die Umgebung {cloze})

Wie man in der vorausgehenden Aufgabe sieht, kann die Tiefe der Unterstriche möglicherweise mit den Buchstaben der unterliegenden Zeile kollidieren. Außerdem ist es mit festen Lücken sehr schwierig, einen bündig ausgerichteten Text hinzukriegen.

Aus diesem Zweck steht die Umgebung `{cloze}` zur Verfügung. Im Grunde ist es eine `{flushleft}` Umgebung mit einem etwas erhöhten Zeilenabstand. Dieser wird standardmäßig um einen Faktor 1.4 erhöht, was mit einem optionalen Parameter geändert werden kann. Für einen größeren Zeilenabstand kann man z.B. `\begin{cloze}[1.6]` verwenden.

### Aufgabe 3 (Lückentexte als graue Boxen)

Das Makro `\fillhere` gibt ihren Inhalt immer im Textmodus wieder. Für den Mathe-Modus ist es eh nicht geeignet, da der Unterstrich in Formeln missverstanden werden könnte. Aus diesem Grunde gibt es für Mathe-Ausdrücke (aber es funktioniert natürlich auch im Text) das Makro `\fillbox`: Dies druckt einen grauen Kasten, der standardmäßig auch zweimal so breit als die „natürliche“ Größe des Textes ist; genau wie bei `\fillhere` kann der Streckfaktor mit Hilfe des optionalen Argumenten geändert werden bzw. explizit als Länge deklariert werden: siehe hier, oder siehe hier. In Gegensatz zu `\fillhere` skaliert allerdings `\fillbox` in Mathe-Modus

$$3^2 = 9, \quad 49^{1/2} = 7$$

und ist somit für mathematische Ausdrücke besser geeignet.

### Aufgabe 4 (Karierte Felder)

Noch etwas Nützliches: Der Befehl `\grid<(x-dimen),(y-dimen)>{...}` erzeugt ein Kastenfeld, dessen Inhalt nur in der Lösung angezeigt wird. Die Baseline der ersten Zeile ist dieselbe des umgebenden Textes. Zum Beispiel erzeugt `\grid(2,1){Text hier drin.}` dies:

Text hier	
drin.	

und 1 cm hoch. Man kann auch explizite Größen angeben, aber Vorsicht! Das Gitter hat 5mm-Schritte, und wenn man kein Vielfaches eines halben Zentimeters angibt, passiert dies: Hello.

Man kann `\grid` auch ohne explizite Maße verwenden: das Kastenfeld ist dann fast so breit wie die Textbreite (trunkiert auf halbe Zentimeter) und 3,5 cm hoch:

Hier kann eine längere Lösung stehen. Gleichungen gehen auch:

$$x = r \cos \phi$$

VORSICHT! Es gibt (noch) keine Kontrolle, dass der Text im Gitter hereinpasst!

Das `\grid` beginnt immer mit einem `\noindent`, so wird es nie eine Einrückung geben.

Will man ein Gitter mit der maximalen Breite aber einer anderen Höhe, kann man die Breite als `*` angeben und die Höhe explizit: `\grid(*,0.5){...}` erzeugt

Dies wurde durch `\grid(*,.5){Dies wurde...}` erzeugt.

Alternativ kann das erste Argument `+` sein: dann wird das Gitter nicht trunziert:

Freilich könnte man dasselbe mit `\grid(\linewidth,0.5){...}` erreichen.

Wie bereits erwähnt, wird der Inhalt von `\grid` nur in der Variante mit Lösung gedruckt. Manchmal will man allerdings doch etwas haben (z.B. „Merke“, oder „Beobachtung“, oder...): dafür kann man das optionale Argument verwenden, z.B. `\grid[\bfseries Merke: ](*,1){Bla bla}` ergibt

**Merke:** Bla bla.

### Aufgabe 5 (Linien)

Ähnlich zu `\grid` gibt es `\lines`. Es funktioniert gleich, hat die gleiche Syntax, und versucht sogar, den Zeilenabstand anzupassen (funktioniert aber nur für reinen Text):

**Schreibe hier etwas:** Und was genau soll ich schreiben? Irgendein Text kommt hier, so als Füller, sozusagen, ohne besonders tiefe Bedeutung.

Ach ja, hier gibt es keinen Unterschied zwischen `+` und `*` für die Breite, denn für Linien ist die Trunkierung nicht relevant.

### Aufgabe 6\* (Zusatzaufgabe)

Die Variante `\exercise*` (z.B. für Zusatzaufgaben) markiert die Aufgabe mit einem Symbol, das im Makro `\starredexercisemark` gespeichert ist. Das verwendete Symbol kann auf übliche Weise mit `\renewcommand` umdefiniert werden, z.B. mit

```
\renewcommand*{\starredexercisemark}{\textsuperscript{+}}
```

bekommt man in der folgenden Aufgabe ein Plus anstatt von einem Sternchen.

### ● Aufgabe 7+ (Klar?)

Man kann natürlich `\hard & Co.` auch verwenden, sowie einen Titel angeben. Es wäre aber auch möglich, dass das Sternchen nicht eindeutig ist und lieber „Zusatzaufgabe“ verwendet wird. Geht auch mit:

```
\renewcommand*{\starredexercisemark}{  
\renewcommand*{\starredexercisename}{Zusatzaufgabe}}
```

wird die nächste Zusatzaufgabe so aussehen:

### Zusatzaufgabe 8 (Ist es nun klar?)

Übrigens: Genau so wie es ein Makro `\starredexercisename` gibt, so gibt es auch `\exercisename`. Beide sind zuerst auf „Aufgabe“ initialisiert. Ähnlich gibt es ein Makro `\solutionname`, das auf „Lösung“ initialisiert wird.

## VERSCHIEDENES

**Aufgabe 1 (Zuordnung-Quizzes)**

Eine weitere nützliche (?) Umgebung ist `{matching}` für Zuordnung-Quizzes. Innerhalb der Umgebung werden mehrere Befehle `\match` mit zwei Argumenten angegeben, als in

```
\begin{matching}[\langle key=val \rangle]
\match{aaa}{AAA}
\match{bbb}{BBB}
\match{ccc}{CCC}
\match{ddd}{DDD}
\match{eee}{EEE}
\end{matching}
```

Diese Paare werden in zwei Spalten geordnet; die rechte Spalte wird mit dem Fisher-Yates-Algorithmus zufällig angeordnet und (in der Lösung) mit Pfeilen verbunden:

aaa	AAA
bbb	EEE
ccc	DDD
ddd	CCC
eee	BBB

Die möglichen Optionen sind:

**xsep**=*<Länge>* setzt den Abstand zwischen den beiden Spalten (default 3 cm);

**ysep**=*<Länge>* setzt den Linienabstand (default 1,3\baselineskip);

**bent** verwendet gebogene statt gerade Linien;

**shuffle**=*<right or left or both>* legt fest, welche Spalte umsortiert wird (shuffle allein ist dasselbe wie shuffle=both);

**seed**=*<Zahl>* initialisiert den Zufallszahlengenerator (default `\time · \year`: das ändert sich daher von Minute zu Minute).

Also mit

```
\begin{matching}[shuffle,bent,seed=13974,xsep=2cm,ysep=.6cm]
```

bekommt man für das Beispiel oben

eee	AAA
ddd	BBB
bbb	DDD
ccc	EEE
aaa	CCC

Bemerkung: `{matching}` startet keine neue Zeile oder Abschnitt, sondern setzt die Tabelle einfach da, wo der Code aufgerufen wird.

**Aufgabe 2 (Neue Tabellenspalten)**

Die Klasse definiert einige besondere Tabellenspalten: ähnlich zu `l`, `c` und `r` gibt es `L`, `C` und `R`, die ihren Inhalt direkt in (`\displaystyle`) Mathe-Modus setzen, so dass man nicht in jeder Zelle `\displaystyle...` tippen muss.

Eine `s`-Spalte ist eine `c`-Spalte, die nur mit der Lösung angezeigt wird (`s` steht für *solution*). Analog ist `S` eine `s`-Spalte in Mathe-Modus. Diese sind bequem, damit man nicht `\solution` in jeder Zelle einer Spalte tippen muss. Es ergibt sich allerdings ein Problem: eine auszufüllende Tabelle hat meistens eine Kopfzeile, deren Inhalt immer sichtbar sein sollte. Das Makro `\scolumnheader` ist dafür gedacht, eben

im Header einer Tabelle verwendet zu werden, damit der Inhalt immer gedruckt wird. Freilich kann man das Makro etwas missbrauchen und in einer beliebigen Zelle (einer s oder S Spalte) verwenden.

Ganz analog gibt es dann auch f- und F-Spalten, denen `\fillhere` zu Grunde liegt. Diese brauchen einen obligatorischen Parameter für die Breite.

Am einfachsten ist ein explizites Beispiel:

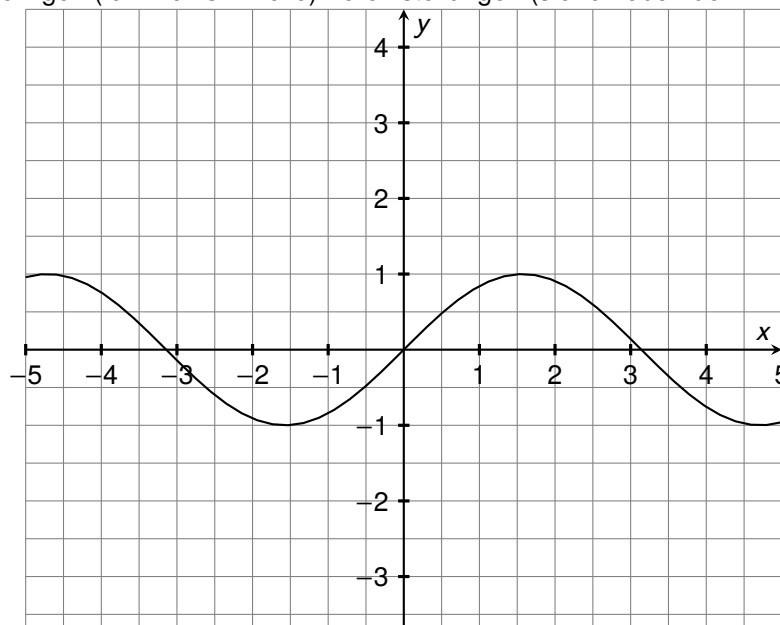
I	c	r	L	C	R	Header hier	Headerhier
II	cc	rr	$L = L$	$C = C$	$R = R$		
III	ccc	rrr	$LL = LL$	$CC = CC$	$RR = RR$		

### Aufgabe 3 (Arbeitsblatt ohne Lösung)

Es gibt auch eine Umgebung `{worksheet*}`, für die *keine* Lösung angegeben wird. Ich verwende sie z.B. wenn ich einfach einen längeren Text austeilen will. Freilich, dafür bräuchte man keine besondere Umgebung und man könnte einfach in der `.tex`-Datei den Text aufschreiben. Genau so wie `{worksheet}` bietet `{worksheet*}` allerdings die Möglichkeit an, Titel/Name/Datum/Schriftgröße festzulegen.

### Aufgabe 4 (Plots)

Die Klasse lädt automatisch `pgfplots`: die Umgebung `{plot}` ist ein dünner Wrapper um `{axis}` mit einigen (für mich sinnvolle) Voreinstellungen (siehe neben dem Plot).



#### Voreinstellungen:

```
compat=1.16,%
grid style=gray,%
axis line style=thick,%
no markers,%
x=1cm,%
y=1cm,%
xlabel style={right},%
ylabel style={right},%
xlabel={x},%
ylabel={y},%
grid=both,%
samples=50,%
axis lines=middle,%
xtick={-10,-9,...,10},%
minor x tick num={1},%
ytick={-10,-9,...,10},%
minor y tick num={1},%
major tick style=
{very thick,black},%
minor tick style={draw=none}
```

Alle `tikz` und `pgfplots` Optionen können angegeben werden.

Wie man in dem Code für das Plot sieht, gibt es ein Makro `\IfSolutionT`, das seinen Inhalt nur in der Variante mit Lösung zeigt. Das Makro kann überall verwendet werden. In der Tat, ist es eins von vier verwandten Makros:

- `\IfSolutionT{arg}` zeigt das Argument nur in der Variante mit Lösung,
- `\IfSolutionF{arg}` zeigt das Argument nur in der Variante ohne Lösung,
- `\IfSolutionTF{arg1}{arg2}` zeigt das Argument `arg1` nur in der Variante mit Lösung, und das Argument `arg2` nur in der Variante ohne Lösung,
- `\IfSolutionFT{arg1}{arg2}` zeigt das Argument `arg1` nur in der Variante ohne Lösung, und das Argument `arg2` nur in der Variante mit Lösung.



### Aufgabe 5 (Vierfeldertafeln)

Das Makro `\crosstable` setzt ihr Argument in einer Vierfeldertafel

	$B$	$\bar{B}$	
$A$	1	2	3
$\bar{A}$	4	5	9
	5	7	12

Der Inhalt der Vierfeldertafel wird erst gemessen, damit alle Zellen die gleiche Größe haben (s. nächstes Beispiel). Die zwei Ereignisse sind standardmäßig  $A$  und  $B$ , aber man kann sie mit dem optionalen Argumenten ändern: das muss zwei von einem Komma getrennten Bezeichnungen haben: Will man also  $R$  und  $M$ , so muss man `\crosstable[R,M]` verwenden:

	$M$	$\bar{M}$	
$R$	$\frac{2}{3}$	$\frac{1}{6}$	
$\bar{R}$	<i>zweite</i>	<i>Zeile</i>	<i>hier</i>
	<i>hier</i>	<i>die</i>	<i>Summe</i>

Natürlich funktionieren in der Tabelle `\IfSolutionT` & Freunde.

### Aufgabe 6 (Schmutzige Tricks...)

Wie in der Lösung von Aufgabe 1 gesagt, bekommt man eine Warnung, wenn die Anzahl der `\questions` und die der `\answers` nicht übereinstimmt. Das kann passieren, wenn z.B. in der gleichen Aufgabe eine Multiple-Choice-Aufgabe und eine `{questions(*)}` Umgebung gibt.

**a.** Frage A

- ☐ Antwort 1
- ☐ Antwort 2
- ☐ Antwort 3

**b.** Frage B

- ☐ Antwort 1
- ☐ Antwort 2
- ☐ Antwort 3

**c.** Frage C

- ☐ Antwort 1
- ☐ Antwort 2
- ☐ Antwort 3

**d.** Und hier noch was.

Nicht, dass das sinnvoll wäre, aber muss man mit der Warnung leben?

### Aufgabe 7 (!! Experimentell !!)

Es gibt ein böses, böses Makro `\addbackgroundgrid`, das ein Gitter auf der *ganzen* Seite druckt. Das Makro nimmt auch ein optionales Argument, das als Option an TikZ weitergegeben wird, d.h.

```
\addbackgroundgrid[<TikZ Optionen>]
```

fügt den folgenden Code zum Background hinzu

```
\tikz[remember picture,overlay]{%
  \draw[gray,step=5mm,<TikZ Optionen>]
    (current page.south west)grid(current page.north east);
}
```

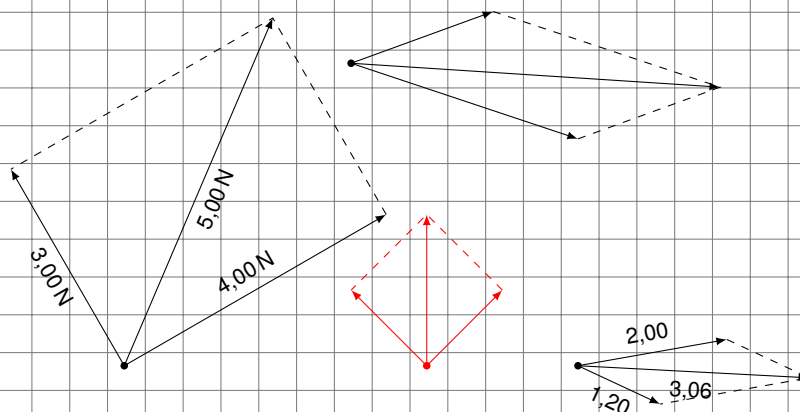
Damit kann man z.B. Farbe oder Gittergröße ändern. Das Hintergrundgitter bleibt bis zum Ende der aktuellen `{worksheet}` Umgebung, oder bis das Makro `\removebackgroundgrid` verwendet wird.

### Aufgabe 8 (Vektorsummen)

Das Makro `\vecsum` nimmt zwei TikZ-Koordinaten und bildet die Summe mit Parallelogramm. Die Syntax ist

```
\vecsum[<TikZ Optionen>](<coord1>)(<coord2>)[<Einheit>];
```

Wird das letzte optionale Argument nicht angegeben, so sieht man nur die Vektoren. Wird etwas angegeben (es ist für eine Einheit gedacht), so wird die Länge der Vektoren ausgegeben. Die TikZ-Optionen können Verschiebungen und Farben enthalten, aber *keine* Skalierung:



Will man die Längen ohne Einheiten haben, so muss man ein leeres optionales Argument angeben.

Das Makro `\vecsum` besitzt auch eine Variante mit Stern `\vecsum*`, die die Summe nur in den Lösungen drückt:



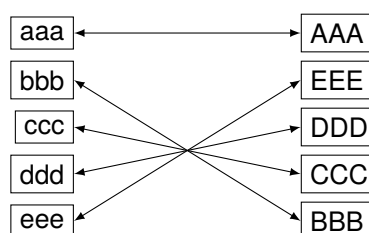
## VERSCHIEDENES

### Aufgabe 1 (Zuordnung-Quizzes)

Eine weitere nützliche (?) Umgebung ist `{matching}` für Zuordnung-Quizen. Innerhalb der Umgebung werden mehrere Befehle `\match` mit zwei Argumenten angegeben, als in

```
\begin{matching}[\langle key=val \rangle]
\match{aaa}{AAA}
\match{bbb}{BBB}
\match{ccc}{CCC}
\match{ddd}{DDD}
\match{eee}{EEE}
\end{matching}
```

Diese Paare werden in zwei Spalten geordnet; die rechte Spalte wird mit dem Fisher-Yates-Algorithmus zufällig angeordnet und (in der Lösung) mit Pfeilen verbunden:



Die möglichen Optionen sind:

**xsep**=*<Länge>* setzt den Abstand zwischen den beiden Spalten (default 3 cm);

**ysep**=*<Länge>* setzt den Linienabstand (default 1,3\baselineskip);

**bent** verwendet gebogene statt gerade Linien;

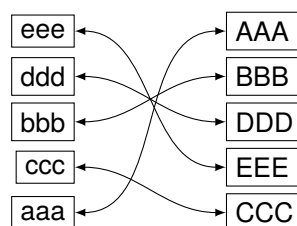
**shuffle**=*<right or left or both>* legt fest, welche Spalte umsortiert wird (shuffle allein ist dasselbe wie shuffle=both);

**seed**=*<Zahl>* initialisiert den Zufallszahlgenerator (default `\time · \year`: das ändert sich daher von Minute zu Minute).

Also mit

```
\begin{matching}[shuffle,bent,seed=13974,xsep=2cm,ysep=.6cm]
```

bekommt man für das Beispiel oben



Bemerkung: `{matching}` startet keine neue Zeile oder Abschnitt, sondern setzt die Tabelle einfach da, wo der Code aufgerufen wird.

### Aufgabe 2 (Neue Tabellenspalten)

Die Klasse definiert einige besondere Tabellenspalten: ähnlich zu `l`, `c` und `r` gibt es `L`, `C` und `R`, die ihren Inhalt direkt in (`\displaystyle`) Mathe-Modus setzen, so dass man nicht in jeder Zelle `\displaystyle...` tippen muss.

Eine `s`-Spalte ist eine `c`-Spalte, die nur mit der Lösung angezeigt wird (`s` steht für *solution*). Analog ist `S` eine `s`-Spalte in Mathe-Modus. Diese sind bequem, damit man nicht `\solution` in jeder Zelle einer Spalte tippen muss. Es ergibt sich allerdings ein Problem: eine auszufüllende Tabelle hat meistens eine Kopfzeile, deren Inhalt immer sichtbar sein sollte. Das Makro `\scolumnheader` ist dafür gedacht, eben

im Header einer Tabelle verwendet zu werden, damit der Inhalt immer gedruckt wird. Freilich kann man das Makro etwas missbrauchen und in einer beliebigen Zelle (einer s oder S Spalte) verwenden.

Ganz analog gibt es dann auch f- und F-Spalten, denen `\fillhere` zu Grunde liegt. Diese brauchen einen obligatorischen Parameter für die Breite.

Am einfachsten ist ein explizites Beispiel:

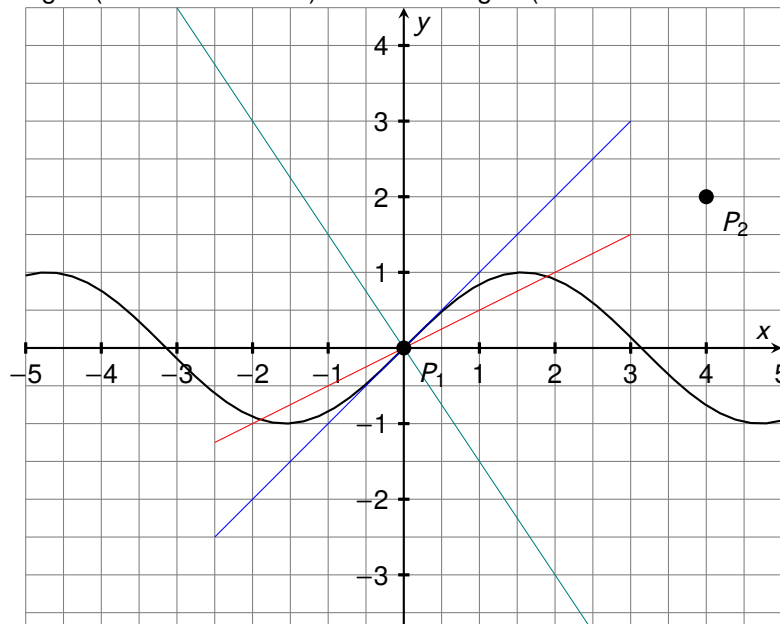
I	c	r	L	C	R	Header hier	<i>Headerhier</i>
II	cc	rr	$L = L$	$C = C$	$R = R$	Lösung hier	$a^b$
III	ccc	rrr	$LL = LL$	$CC = CC$	$RR = RR$	Lösung hier	$a^b$

### Aufgabe 3 (Arbeitsblatt ohne Lösung)

Es gibt auch eine Umgebung `{worksheet*}`, für die *keine* Lösung angegeben wird. Ich verwende sie z.B. wenn ich einfach einen längeren Text austeilen will. Freilich, dafür bräuchte man keine besondere Umgebung und man könnte einfach in der `.tex`-Datei den Text aufschreiben. Genau so wie `{worksheet}` bietet `{worksheet*}` allerdings die Möglichkeit an, Titel/Name/Datum/Schriftgröße festzulegen.

### Aufgabe 4 (Plots)

Die Klasse lädt automatisch `pgfplots`: die Umgebung `{plot}` ist ein dünner Wrapper um `{axis}` mit einigen (für mich sinnvolle) Voreinstellungen (siehe neben dem Plot).



#### Voreinstellungen:

```
compat=1.16,%
grid style=gray,%
axis line style=thick,%
no markers,%
x=1cm,%
y=1cm,%
xlabel style={right},%
ylabel style={right},%
xlabel={\$x\$},%
ylabel={\$y\$},%
grid=both,%
samples=50,%
axis lines=middle,%
xtick={-10,-9,...,10},%
minor x tick num={1},%
ytick={-10,-9,...,10},%
minor y tick num={1},%
major tick style={very thick,black},%
minor tick style={draw=none}
```

Alle `tikz` und `pgfplots` Optionen können angegeben werden.

Wie man in dem Code für das Plot sieht, gibt es ein Makro `\IfSolutionT`, das seinen Inhalt nur in der Variante mit Lösung zeigt. Das Makro kann überall verwendet werden. (Zum Beispiel hier.) In der Tat, ist es eins von vier verwandten Makros:

- `\IfSolutionT{arg}` zeigt das Argument nur in der Variante mit Lösung,
- `\IfSolutionF{arg}` zeigt das Argument nur in der Variante ohne Lösung,
- `\IfSolutionTF{arg1}{arg2}` zeigt das Argument `arg1` nur in der Variante mit Lösung, und das Argument `arg2` nur in der Variante ohne Lösung,
- `\IfSolutionFT{arg1}{arg2}` zeigt das Argument `arg1` nur in der Variante ohne Lösung, und das Argument `arg2` nur in der Variante mit Lösung.

### Aufgabe 5 (Vierfeldertafeln)

Das Makro `\crosstable` setzt ihr Argument in einer Vierfeldertafel

	$B$	$\bar{B}$	
$A$	1	2	3
$\bar{A}$	4	5	9
	5	7	12

Der Inhalt der Vierfeldertafel wird erst gemessen, damit alle Zellen die gleiche Größe haben (s. nächstes Beispiel). Die zwei Ereignisse sind standardmäßig  $A$  und  $B$ , aber man kann sie mit dem optionalen Argumenten ändern: das muss zwei von einem Komma getrennten Bezeichnungen haben: Will man also  $R$  und  $M$ , so muss man `\crosstable[R,M]` verwenden:

	$M$	$\bar{M}$	
$R$	$\frac{2}{3}$	$\frac{1}{6}$	$\frac{5}{6}$
$\bar{R}$	<i>zweite</i>	<i>Zeile</i>	<i>hier</i>
	<i>hier</i>	<i>die</i>	<i>Summe</i>

Natürlich funktionieren in der Tabelle `\IfSolutionT` & Freunde.

### Aufgabe 6 (Schmutzige Tricks...)

Wie in der Lösung von Aufgabe 1 gesagt, bekommt man eine Warnung, wenn die Anzahl der `\questions` und die der `\answers` nicht übereinstimmt. Das kann passieren, wenn z.B. in der gleichen Aufgabe eine Multiple-Choice-Aufgabe und eine `{questions(*)}` Umgebung gibt.

a. Frage A

- ☐ Antwort 1
- ☒ Antwort 2
- ☐ Antwort 3

b. Frage B

- ☒ Antwort 1
- ☐ Antwort 2
- ☐ Antwort 3

c. Frage C

- ☐ Antwort 1
- ☐ Antwort 2
- ☒ Antwort 3

d. Und hier noch was.

Nicht, dass das sinnvoll wäre, aber muss man mit der Warnung leben?

### Lösung

Es gibt mehrere Auswege. Man kann natürlich so was schreiben wie:

`\answer (Siehe Text.)`

und das dreimal. Die vierte `\answer` schreibt man wie gewohnt. Man kann aber auch die Kontrolle am Ende der `{solution}` Umgebung überspringen, indem man in der Lösung (egal wo) den Befehl `\NoCheck` schreibt.

Allerdings nimmt `\answer` auch ein optionales Argument, das ein Kleinbuchstabe sein muss. In diesem Fall z.B. brauchen wir die Lösung ab Teil d, also machen wir...

d. ... und alles ist in Ordnung.

Nein, es gibt keinen Automatismus: man muss selbst zählen! Es gibt zu viele (unsinnige) Kombinationen, dass alle Fälle abgedeckt werden könnten.

### Aufgabe 7 (!! Experimentell !!)

Es gibt ein böses, böses Makro `\addbackgroundgrid`, das ein Gitter auf der *ganzen* Seite druckt. Das Makro nimmt auch ein optionales Argument, das als Option an TikZ weitergegeben wird, d.h.

```
\addbackgroundgrid[<TikZ Optionen>]
```

fügt den folgenden Code zum Background hinzu

```
\tikz[remember picture,overlay]{%
  \draw[gray,step=5mm,<TikZ Optionen>]
    (current page.south west)grid(current page.north east);
}
```

Damit kann man z.B. Farbe oder Gittergröße ändern. Das Hintergrundgitter bleibt bis zum Ende der aktuellen `{worksheet}` Umgebung, oder bis das Makro `\removebackgroundgrid` verwendet wird.

### Lösung

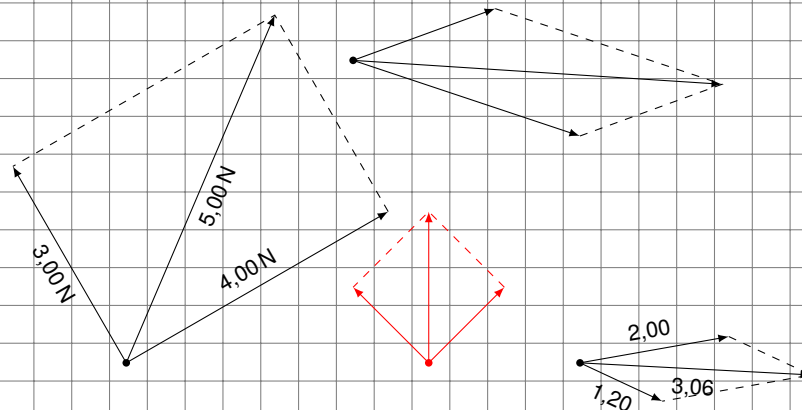
ICH HABE ES NOCH NICHT GANZ DURCHGETESTET, ALSO MIT VORSICHT GENIESSEN!

### Aufgabe 8 (Vektorsummen)

Das Makro `\vecsum` nimmt zwei TikZ-Koordinaten und bildet die Summe mit Parallelogramm. Die Syntax ist

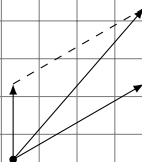
```
\vecsum[<TikZ Optionen>](<coord1>)(<coord2>)[<Einheit>];
```

Wird das letzte optionale Argument nicht angegeben, so sieht man nur die Vektoren. Wird etwas angegeben (es ist für eine Einheit gedacht), so wird die Länge der Vektoren ausgegeben. Die TikZ-Optionen können Verschiebungen und Farben enthalten, aber *keine* Skalierung:



Will man die Längen ohne Einheiten haben, so muss man ein leeres optionales Argument angeben.

Das Makro `\vecsum` besitzt auch eine Variante mit Stern `\vecsum*`, die die Summe nur in den Lösungen drückt:



Datum:

## SCHÜLERVERSUCHE

### !! VORSICHT !! ÄNDERUNG MIT v1.0.0 !!

In früheren Varianten der Klasse gab es eine Umgebung `{experiment}`, die eigentlich dasselbe wie `{worksheet}` war, nur dass der Befehl `\experiment` zur Verfügung stand. Das habe ich im Grunde nie verwendet, aber eine Kleinigkeit habe ich behalten: In einer `{worksheet}` Umgebung kann man auch eben einen Versuch mit `\experiment` angeben. Die Syntax ist die gleiche wie bei `\exercise` (außer Schwierigkeitsgrad), und die Zähler sind unabhängig, d.h. wir können Folgendes haben:

#### **Versuch 1 (Eingagsversuch)**

Tu irgendwas.

#### **Aufgabe 1 (Foo)**

Text.

#### **Aufgabe 2 (Bar)**

Text.

#### **Versuch 2 (Noch ein Versüchschchen...)**

Tu was anderes.

#### **Aufgabe 3 (Zähler geht weiter)**

Die Titel der Aufgaben/Versuche sind standardmäßig in runden Klammern und mit einem Abstand von `\enskip` von der Zahl platziert. Dies kann natürlich geändert werden, und zwar mit Hilfe des Makros

$$\settitleseparators[\langle Abstand \rangle]{\langle linkes Zeichen \rangle}{\langle rechtes Zeichen \rangle}$$

Die obligatorischen Argumente sind die Zeichen links und rechts vom Titel (default runde Klammern); das optionale Argument ist der Abstand zwischen Nummer und Titel (default eben `\enskip`). Mit

$$\settitleseparators{[]}{\$\rangle\$}$$

kriegt man zum Beispiel

#### **Aufgabe 4 [Sieht blöd aus]**

Interessanter wäre zum Beispiel mit

$$\settitleseparators[]{\quad---\quad}\{\}$$

dann hat man

#### **Versuch 3 — Klar?**

Sowohl `\exercise` als auch `\experiment` verwenden die gleichen Klammer/Abstände.

**!! VORSICHT !! ÄNDERUNG MIT v1.0.0 !!**

In früheren Varianten der Klasse gab es eine Umgebung `{experiment}`, die eigentlich dasselbe wie `{worksheet}` war, nur dass der Befehl `\experiment` zur Verfügung stand. Das habe ich im Grunde nie verwendet, aber eine Kleinigkeit habe ich behalten: In einer `{worksheet}` Umgebung kann man auch eben einen Versuch mit `\experiment` angeben. Die Syntax ist die gleiche wie bei `\exercise` (außer Schwierigkeitsgrad), und die Zähler sind unabhängig, d.h. wir können Folgendes haben:

**Versuch 1 (Eingagsversuch)**

Tu irgendwas.

**Aufgabe 1 (Foo)**

Text.

**Aufgabe 2 (Bar)**

Text.

**Versuch 2 (Noch ein Versüchschchen...)**

Tu was anderes.

**Aufgabe 3 (Zähler geht weiter)**

Die Titel der Aufgaben/Versuche sind standardmäßig in runden Klammern und mit einem Abstand von `\enskip` von der Zahl platziert. Dies kann natürlich geändert werden, und zwar mit Hilfe des Makros

`\SetTitleSeparators[⟨Abstand⟩]{⟨linkes Zeichen⟩}{⟨rechtes Zeichen⟩}`

Die obligatorischen Argumente sind die Zeichen links und rechts vom Titel (default runde Klammern); das optionale Argument ist der Abstand zwischen Nummer und Titel (default eben `\enskip`). Mit

`\SetTitleSeparators{[]{$\rangle$}`

kriegt man zum Beispiel

**Aufgabe 4 [Sieht blöd aus]**

Interessanter wäre zum Beispiel mit

`\SetTitleSeparators[]{\quad---\quad}{}`

dann hat man

**Lösung**

Freilich, man könnte auch `\SetTitleSeparators[\quad]{---\quad}{}` schreiben.

**Versuch 3 — Klar?**

Sowohl `\exercise` als auch `\experiment` verwenden die gleichen Klammer/Abstände.



## TITEL, WENN MAN WILL

Die Umgebung `{print2}` ist nützlich, um etwas doppelt auf der Seite zu drucken, das man den Schülern ausgeben kann, z.B. kurze Texte mit Bildern, Aufgabenstellungen oder was auch immer:



Der Titel kann als optionaler Parameter angegeben werden. Standardmäßig ist der horizontale Rand 2 cm und der vertikale Rand 1,5 cm. Die Ränder können durch Optionen geändert werden, z.B.

```
\begin{print2}[hmargin=3cm,vmargin=1cm]
```

Die Option `margin=<Länge>` verwendet den gleichen Wert für beide Richtungen. Will man diese Optionen angeben, so muss der Titel natürlich auch mit Hilfe von `title=<Titel>` angegeben werden. Mit der Option `fontsize=<Schriftgröße>` kann man die Schriftgröße ändern.

Die Umgebung `{print2+}` funktioniert ähnlich wie `{print2}`, aber man kann auch eine Lösung dazu schreiben. Dann wird erst die Variante für Schüler (doppelt) gedruckt, und dann die Variante mit Lösung (natürlich nur einmal).

Die Umgebung `{print2-}` ist dagegen leicht anders: Sie druckt *auf dieselbe* Seite ihren Inhalt, einmal mit und einmal ohne Lösung. Ehrlich gesagt weiß ich nicht mehr, wozu ich sie gebraucht habe...

## TITEL, WENN MAN WILL

Die Umgebung `{print2}` ist nützlich, um etwas doppelt auf der Seite zu drucken, das man den Schülern ausgeben kann, z.B. kurze Texte mit Bildern, Aufgabenstellungen oder was auch immer:



Der Titel kann als optionaler Parameter angegeben werden. Standardmäßig ist der horizontale Rand 2 cm und der vertikale Rand 1,5 cm. Die Ränder können durch Optionen geändert werden, z.B.

```
\begin{print2}[hmargin=3cm,vmargin=1cm]
```

Die Option `margin=<Länge>` verwendet den gleichen Wert für beide Richtungen. Will man diese Optionen angeben, so muss der Titel natürlich auch mit Hilfe von `title=<Titel>` angegeben werden. Mit der Option `fontsize=<Schriftgröße>` kann man die Schriftgröße ändern.

Die Umgebung `{print2+}` funktioniert ähnlich wie `{print2}`, aber man kann auch eine Lösung dazu schreiben. Dann wird erst die Variante für Schüler (doppelt) gedruckt, und dann die Variante mit Lösung (natürlich nur einmal).

Die Umgebung `{print2-}` ist dagegen leicht anders: Sie druckt *auf dieselbe* Seite ihren Inhalt, einmal mit und einmal ohne Lösung. Ehrlich gesagt weiß ich nicht mehr, wozu ich sie gebraucht habe...



Die Umgebung `{print3}` ist ähnlich wie `{print2}`, nur dass sie selbstverständlich ihren Inhalt dreimal druckt. Es gibt auch eine Variante `{print3+}`, die danach auch den Text mit Lösung druckt. Auch hier kann man Aufgaben setzen:

● **Aufgabe 1 (Irgendwas)**

Text text



Die zwei horizontalen Linien sollen helfen, das Blatt zu schneiden.

Analog gibt es die Umgebungen `{print4}` und `{print4+}`: Sie machen genau, was ihr Name nahelegt.

Vorsicht: Verwendet man (wie hier) `\exercise` in einer `{printX}` Umgebung und wird `hyperref` geladen, so hat man Warnungen über uneindeutige Bookmarks. Ich habe keine Ahnung, wie man das vermeiden kann, aber das ist kein großes Problem. (Nur eine nervige Warnung halt.)

---

Die Umgebung `{print3}` ist ähnlich wie `{print2}`, nur dass sie selbstverständlich ihren Inhalt dreimal druckt. Es gibt auch eine Variante `{print3+}`, die danach auch den Text mit Lösung druckt. Auch hier kann man Aufgaben setzen:

● **Aufgabe 1 (Irgendwas)**

Text text



Die zwei horizontalen Linien sollen helfen, das Blatt zu schneiden.

Analog gibt es die Umgebungen `{print4}` und `{print4+}`: Sie machen genau, was ihr Name nahelegt.

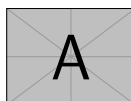
Vorsicht: Verwendet man (wie hier) `\exercise` in einer `{printX}` Umgebung und wird `hyperref` geladen, so hat man Warnungen über uneindeutige Bookmarks. Ich habe keine Ahnung, wie man das vermeiden kann, aber das ist kein großes Problem. (Nur eine nervige Warnung halt.)

---

Die Umgebung `{print3}` ist ähnlich wie `{print2}`, nur dass sie selbstverständlich ihren Inhalt dreimal druckt. Es gibt auch eine Variante `{print3+}`, die danach auch den Text mit Lösung druckt. Auch hier kann man Aufgaben setzen:

● **Aufgabe 1 (Irgendwas)**

Text text



Die zwei horizontalen Linien sollen helfen, das Blatt zu schneiden.

Analog gibt es die Umgebungen `{print4}` und `{print4+}`: Sie machen genau, was ihr Name nahelegt.

Vorsicht: Verwendet man (wie hier) `\exercise` in einer `{printX}` Umgebung und wird `hyperref` geladen, so hat man Warnungen über uneindeutige Bookmarks. Ich habe keine Ahnung, wie man das vermeiden kann, aber das ist kein großes Problem. (Nur eine nervige Warnung halt.)



**Vorne 1**

**Vorne 2**

**Vorne 3**

**Vorne 4**

**Vorne 5**

**Vorne 6**

#### Hinten 2

Standardmäßig sind 2x3 Kärtchen auf einem Blatt.

#### Hinten 1.

Damit kann man Lösungskärtchen drucken. Das erste Argument von `\cluecard` wird vorne gedruckt (mit `\Large\bfseries\centering`; siehe unten, wie man das ändern kann). Das zweite Argument kommt hinten.

$$a = b$$



Gleichungen und Bilder sind unproblematisch.

#### Hinten 4

Die Standardgeometrie kann geändert werden: mit `\begin{cluecards}[N\times M]` werden  $N$  Spalten und  $M$  Zeilen erzeugt. Siehe das nächste Beispiel.

#### Hinten 3

Damit das mit vorne/hinten gut funktioniert, muss der Drucker natürlich mitmachen... das ist leider nicht unter meiner Kontrolle.

#### Hinten 6

Darüber hinaus kann man die Schrift ändern mit `front=...` (default `\Large\bfseries\centering`) und `back=...` (default leer).

#### Hinten 5

Auch hier kann man eine Key/Value-Syntax verwenden. `\begin{cluecards}[3\times 3]` ist dasselbe wie `\begin{cluecards}[layout=3\times 3]`.

*Kleiner Hinweis zu A1*

*Großer Hinweis zu A1*

*Lösung zu A1*

*Kleiner Hinweis zu A2*

*Großer Hinweis zu A2*

*Lösung zu A2*

*Kleiner Hinweis zu A3*

*Großer Hinweis zu A3*

*Lösung zu A3*

Lösung zu A1

Ein großer Hinweis für A2

Ein kleiner Hinweis für A1

Die Lösung für A2

Ein großer Hinweis für A2

Ein kleiner Hinweis für A2

Die Lösung für A3  
Man muss selbst zählen. Hat  
man zu viele \cluecards,  
so werden die Überzählige  
ignoriert.

Ein großer Hinweis für A3

Ein kleiner Hinweis für A3





Name:		
Punkte:	/6	Note:
		mündlicher Eindruck:

Eine Klassenarbeit/Test/Was auch immer wird in der Umgebung `{test}` gesetzt. Normalerweise müssen einige Keys gesetzt werden, zum Beispiel:

```
\begin{test}[number=2,  
            subject=Mathematik,  
            date=32. Oktober 2022,  
            version=A,  
            ptspre=>,  
            ptspost=<,  
            logo=example-image]
```

Hier eine kurze Beschreibung aller Keys:

**subject**=*<Fach>* legt das Fach fest. Es gibt auch die Optionen M, Ph und NwT, welche äquivalent sind zu `subject=Mathematik`, `subject=Physik` und `subject=NwT` sind.

**number**, **nr** legt die Nummer fest. Man kann auch nur 1 anstatt von `nr=1` schreiben, und das geht bis 4. Höhere Zahlen müssen mit `nr=` oder `number=` angegeben werden.

**date** Selbsterklärend.

**class** Auch selbsterklärend.

**schoolyear** Das aktuelle Schuljahr wird automatisch berechnet basierend auf dem aktuellen Monat. Alternativ kann man `schoolyear=...` explizit angeben.

**type** Art des Tests. Voreingestellt ist „KA“, aber mit `type=Test` kann man das ändern.

**version/variant/v** Will man verschiedene Varianten haben, so kann man e.g. `v=A` angeben.

**ptspre/prepts** Fügt Text vor der Punktzahl ein.

**ptspost/postpts** Fügt Text nach der Punktzahl ein.

**background/bg**=*<TikZ Optionen>* Fügt das Hintergrundgitter ein. Der Parameter wird weitergegeben, d.h. `bg=red` wird übersetzt in `\addbackgroundgrid[red]`. Gibt man nur `bg` ohne weitere Angabe, dann kriegt man das „normale“ graue Gitter, was von `\addbackgroundgrid` erzeugt wird.

**logo**=*<Dateiname>* Die angegebene Datei wird als Logo verwendet, das oben links in der Kopfzeile gedruckt wird. Wird kein Logo gewünscht, so einfach nichts angeben... Alternativ kann man in der Präambel (oder in der Konfigurationsdatei `drschool.cfg`, oder in einer `.sco` Konfigurationsdatei) `\SetLogo{Dateiname}` angeben.

Alle Befehle, die in einem Arbeitsblatt Verwendung finden, können auch hier verwendet werden. Es gibt einen Unterschied: das optionale Argument von `\exercise` ist nicht der Titel der Aufgabe, sondern die Punktzahl. Die einzelnen Punktzahlen werden in die `aux`-Datei geschrieben, so dass die Gesamtpunktzahl in der Tabelle automatisch errechnet wird.

Wie bei der Herstellung des Inhaltsverzeichnisses muss  $\text{\TeX}$  nach jeder Änderung einer Punktzahl mindestens *zweimal* laufen, damit die korrekte Gesamtzahl errechnet wird!

## ☒ 1. Teil — OHNE Taschenrechner

### ● Aufgabe 1 (>0,5< Punkte)

Man kann natürlich auch hier die Präfixe `\hard`, `\medium` und `\easy` verwenden.

Die komischen Zeichen „>“ und „<“ vor/nach der Punktzahl sind hier nur als Beispiel gegeben. Hier wurde z.B.

```
\begin{test}[...,ptspre=>,ptspost=<,...]
```

verwendet. Meistens nutze ich `ptspre={vor. }` („voraussichtlich“, es ist immer gut, sich Spielraum zu lassen...)

### Aufgabe 2 (>1< Punkt)

Die Gesamtpunktzahl wird automatisch gerechnet, stimmt aber erst ab der zweiten Kompilierung. Dies ist Aufgabe 2.

 2. Teil — mit Taschenrechner

### Aufgabe 3 (>2,5< Punkte)

Halbe Punkte können sowohl mit Komma als auch mit Punkt als Dezimaltrenner kodiert werden.  $\text{\TeX}$  ist da schlau genug, beides zu verstehen.

### Aufgabe 4\* (>3< Punkte)

Man kann auch hier Zusatzaufgaben angeben. Deren Punktzahl wird *nicht* zur Gesamtpunktzahl hinzugefügt.

### Aufgabe 5 (>2< Punkte)

Das Makro `\question` kann nur innerhalb der `{questions<*>}` oder der Single- bzw. Multiple-Choice-Umgebungen verwendet werden. (Außerhalb ergibt es einen Fehler.) Wie bereits gezeigt, druckt die Sternform den Inhalt des Makros `\starredquestionmark` links vom Buchstaben. Darüber hinaus kann man (mit oder ohne Stern) auch eine Teilpunktzahl in eckigen Klammern angeben. Das funktioniert in `{questions*}`

a. Normale Frage.

\*b. Extra Frage.

c. (1 Punkt) Frage.

in der „normalen“ `{questions}`

d. (0,5 Punkte) Frage.

e. (2 Punkte) Frage. Wenn man jetzt `\renewcommand{\starredquestionmark}{+}` schreibt, bekommt man beim nächsten `\question*`

+f. (1,5 Punkte) Extra Frage.

sowie z.B. in `{multiresponse<*>}` und `{multichoice<*>}`

\*g. (1 Punkt) Frage A

☐ Antwort 1

☐ Antwort 2

h. Frage B

☐ Antwort 1

☐ Antwort 2



Name:		
Punkte:	/6	Note:
		mündlicher Eindruck:

Eine Klassenarbeit/Test/Was auch immer wird in der Umgebung `{test}` gesetzt. Normalerweise müssen einige Keys gesetzt werden, zum Beispiel:

```
\begin{test}[number=2,
             subject=Mathematik,
             date=32. Oktober 2022,
             version=A,
             ptspre=>,
             ptspost=<,
             logo=example-image]
```

Hier eine kurze Beschreibung aller Keys:

**subject**=*<Fach>* legt das Fach fest. Es gibt auch die Optionen M, Ph und NwT, welche äquivalent sind zu `subject=Mathematik`, `subject=Physik` und `subject=NwT` sind.

**number**, **nr** legt die Nummer fest. Man kann auch nur 1 anstatt von `nr=1` schreiben, und das geht bis 4. Höhere Zahlen müssen mit `nr=` oder `number=` angegeben werden.

**date** Selbsterklärend.

**class** Auch selbsterklärend.

**schoolyear** Das aktuelle Schuljahr wird automatisch berechnet basierend auf dem aktuellen Monat. Alternativ kann man `schoolyear=...` explizit angeben.

**type** Art des Tests. Voreingestellt ist „KA“, aber mit `type=Test` kann man das ändern.

**version/variant/v** Will man verschiedene Varianten haben, so kann man e.g. `v=A` angeben.

**ptspre/prepts** Fügt Text vor der Punktzahl ein.

**ptspost/postpts** Fügt Text nach der Punktzahl ein.

**background/bg**=*<TikZ Optionen>* Fügt das Hintergrundgitter ein. Der Parameter wird weitergegeben, d.h. `bg=red` wird übersetzt in `\addbackgroundgrid[red]`. Gibt man nur `bg` ohne weitere Angabe, dann kriegt man das „normale“ graue Gitter, was von `\addbackgroundgrid` erzeugt wird.

**logo**=*<Dateiname>* Die angegebene Datei wird als Logo verwendet, das oben links in der Kopfzeile gedruckt wird. Wird kein Logo gewünscht, so einfach nichts angeben... Alternativ kann man in der Präambel (oder in der Konfigurationsdatei `drschool.cfg`, oder in einer `.sco` Konfigurationsdatei) `\SetLogo{Dateiname}` angeben.

Alle Befehle, die in einem Arbeitsblatt Verwendung finden, können auch hier verwendet werden. Es gibt einen Unterschied: das optionale Argument von `\exercise` ist nicht der Titel der Aufgabe, sondern die Punktzahl. Die einzelnen Punktzahlen werden in die `aux`-Datei geschrieben, so dass die Gesamtpunktzahl in der Tabelle automatisch errechnet wird.

Wie bei der Herstellung des Inhaltsverzeichnisses muss  $\text{\TeX}$  nach jeder Änderung einer Punktzahl mindestens *zweimal* laufen, damit die korrekte Gesamtzahl errechnet wird!

## ☒ 1. Teil — OHNE Taschenrechner

### ● Aufgabe 1 (>0,5< Punkte)

Man kann natürlich auch hier die Präfixe `\hard`, `\medium` und `\easy` verwenden.

Die komischen Zeichen „>“ und „<“ vor/nach der Punktzahl sind hier nur als Beispiel gegeben. Hier wurde z.B.

```
\begin{test}[...,ptspre=>,ptspost=<,...]
```



- c. Es gibt **absolut keine Kontrolle**, dass die Summe aller Teilpunkte der Gesamtpunktzahl entspricht.
- d. Das ist eigentlich ein „Relikt“ meiner Klasse `unituennf`, das einfach copy/pasted wurde.
- e. Es fehlen Antworten, aber keine Warnung, weil ich hier `\NoCheck` verwendet habe.
- +f. Was *möglich* ist, ist aber auch nicht unbedingt *sinnvoll*. Ich finde horizontale Auflistungen mit Punktzahl etwas zu „überladen“, aber das ist Geschmackssache.



## 6 „Utilities“

Dieser Abschnitt beschreibt einige Makros, welche die Klasse zur Verfügung stellt. Da sie nicht notwendigerweise mit Arbeitsblättern zu tun haben bzw. auch außerhalb der `{worksheet}` Umgebung funktionieren, werden sie hier vorgestellt.

### 6.1 Farbige `{minipage}`

Die Klasse definiert eine Umgebung `{colorminipage}`, die sich wie `{minipage}` verhält, aber ihren Inhalt in einer farbigen Box setzt. Es handelt sich im Grunde um eine Kombination aus `\colorbox` und `{minipage}`, aber die Textbreite wird so gerechnet, dass die angegebene Breite auch den Rand berücksichtigt. Die Randgröße ist ganz normal der Wert von `\fboxsep`. Alle optionalen Argumente zu `{minipage}` können wie gewohnt verwendet werden:

```
\begin{colorminipage}{\langle Farbe \rangle}[\langle Ausrichtung \rangle][\langle Höhe \rangle][\langle innere Ausrichtung \rangle]{\langle Breite \rangle}
```

Beispiel: mit `\begin{colorminipage}{teal!30!white}[t]{3cm}` hat man:

Hello world, and  
such, for 'tis bet-  
ter to suffer, and  
yet, nothing was  
meant by this text.

Die Umgebung `{graybox}` ist ein `{colorminipage}`, bei der die Farbe auf `lightgray` festgelegt wurde, d.h.

```
\begin{graybox}[\langle Ausr. \rangle][\langle Höhe \rangle][\langle inn. Ausr. \rangle]{\langle Breite \rangle}
```


ist eine Abkürzung für

```
\begin{colorminipage}{lightgray}[\langle Ausr. \rangle][\langle Höhe \rangle][\langle inn. Ausr. \rangle]{\langle Breite \rangle}
```

### 6.2 Unsichtbares `\put`




Das LaTeX Makro `\put(\langle x-Shift \rangle, \langle y-Shift \rangle){...}` platziert ihren Inhalt verschoben um die Werte `\langle x-Shift \rangle` und `\langle y-Shift \rangle`. Die daraus resultierende Box hat keine Breite, aber sehr wohl Höhe und Tiefe. Dagegen erzeugt `\drcput` eine Box, die keinen Platz in Anspruch nimmt, ansonsten aber die gleiche Syntax wie `\put` hat. Mit einem optionalen Parameter entscheidet man, ob der Inhalt rechts steht (default, wie bei `\put`), oder links, oder zentriert. Ich mache ein Beispiel und zeichne zuerst einen Pfeil 2 cm nach rechts und 2 ex nach oben als Referenzpunkt. Mit dem Code

```
\rlap{\tikz{\draw[->](0,0)--(2cm,3ex);}}%  
\drcput(2cm,3ex){\textcolor{red}{X}}% same as \drcput[r](2cm,3ex)  
\drcput[c](2cm,3ex){\textcolor{blue}{X}}%  
\drcput[l](2cm,3ex)[l]{\textcolor{green}{X}}%
```

kriegt man:  Wozu? Na ja, ich benutze es, um z.B. die Lösungen auf gescannten bzw. importierten Arbeitsblätter drauf zu schreiben.

### 6.3 Rechte Winkel

Mit der TikZ-Library `angles` (die standardmäßig geladen wird) kann man einen Winkel zwischen drei Koordinaten plotten. Beispiel: Man definiere drei Koordinaten  $A(0.5|1)$ ,  $B(0|0)$  und  $C(1|0)$  und man vergleiche

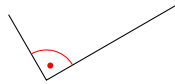
`\pic[draw]{angle=C--B--A};` →  `\pic[draw]{right angle=C--B--A};` →   
`\pic[draw]{rightangle=C--B--A};` → 

Die ersten zwei Bilder zeigen das Ergebnis von `angle` und `right angle`, welche in TikZ vordefiniert sind. Das dritte Bild dagegen kommt aus `rightangle` (zusammengeschrieben!). Man beachte, dass man den Punkt in der Mitte des Winkels immer bekommt, unabhängig davon, ob der Winkel wirklich ein rechter Winkel ist. (Ich habe absichtlich auf eine entsprechende Kontrolle und Warnung/Fehlermeldung verzichtet: Manchmal will man einen rechten Winkel in einer 3D-Darstellung markieren, wo der tatsächlich gezeichnete Winkel nicht wirklich recht ist.)

Als ältere Variante existiert noch das Makro

```
\rightangle[⟨TikZ Optionen⟩](⟨coord⟩){⟨start angle⟩}[⟨radius⟩]
```

plottet in einem TikZ-Bild einen rechten Winkel (und zwar immer einen rechten Winkel).



## 6.4 Einfache Formen von `{wrapfigure}`

Das Makro `\wrap[⟨l oder r⟩]{⟨Inhalt⟩}` ist eine sehr abgespeckte Version der vom Paket `wrapfig` definierten Umgebung `{wrapfigure}`. Standardmäßig ist das Bild auf der linken Seite: Dies wurde hier mittels `\wrap[r]{...}` geändert. Das Makro `\wrap` ist dazu gedacht, zu Beginn eines Abschnitts verwendet zu werden, und daher startet sie immer auch einen neuen Abschnitt. Wer eine feinere Kontrolle und bessere Ergebnisse möchte, sollte natürlich auf `{wrapfigure}` zugreifen.



Ähnlich zu `\wrap` existiert auch

```
\wraptikz[⟨TikZ Optionen⟩][⟨l oder r⟩]{⟨TikZ Code⟩}
```

das im Grunde dasselbe wie `\wrap` macht aber den Inhalt gleich in einer `{tikzpicture}` setzt.

Die Syntax ist leicht anders, denn es gibt nun *zwei* optionale Argumente: erst die TikZ-Optionen und dann `l` (default) oder `r`.

Vorsicht! Bei `\wrap` und `\wraptikz` soll beachtet werden, dass sie nicht gut reagieren, wenn in ihrer Nähe ein neuer Abschnitt begonnen wird oder gar ein Seitenumbruch stattfindet. Dann wird das Ergebnis äußerst enttäuschend sein...

## 7 Ausfüllbare PDFs (NEU! v1.0.0)

Lädt man die Klasse mit der Option `hyperworksheet`, so steht auch eine gleichnamige Umgebung zur Verfügung (sowie eine Sternform dazu). Diese verhält sich im großen und ganzen wie die normale `{worksheet}`, erzeugt aber eine *ausfüllbare* PDF-Datei, d.h. mit Feldern, die mit einem geeigneten PDF-Viewer ausgefüllt werden können.

Damit dies erfolgen kann, muss natürlich das Paket `hyperref` geladen werden. Da dies einen ziemlich großen Einschnitt in die Funktion verschiedener Makros und Umgebungen darstellt, ist dies keine Defaulteinstellung sondern muss eben extra mit Option deklariert werden.

Wenn jemand unbedingt denkt, das *muss* ihr/sein Standard sein, und keine Lust hat, die Option `hyperworksheet` jedes Mal zu schreiben, dann kann man

```
\Hyperworksheet
```

in der Hauptkonfigurationsdatei `drschool.cfg` schreiben.

**Bemerkung:** Es gibt einige wenige Pakete, die *nach* `hyperref` geladen werden müssen: diese können getrost in der Präambel geladen werden. Wenn jemand allerdings ein Paket will, das vor `hyperref` geladen werden muss, so soll dies entweder noch vor `\documentclass` erfolgen, oder in einer Hauptkonfigurationsdatei.

Das folgende Beispiel beschreibt, was darin möglich ist:



Datum:

## MIT VORSICHT GENIESSEN!

Ob und wie genau PDF-Forms funktionieren (oder nicht), hängt *sehr* vom verwendeten PDF-Viewer ab! Wenn etwas nicht läuft, dann nicht sofort mit den Schülern/Studenten schimpfen! Vielleicht trifft sie doch ausnahmsweise keine Schuld...

Zuerst muss im Code vor der Umgebung `{hyperworksheet}` das Makro `\Form` aufgerufen werden. Alternativ, kann man das Arbeitsblatt zwischen `\begin{Form}` und `\end{Form}` setzen. Pro Datei kann man nur *einmal* `\Form` (bzw. eine `{Form}` Umgebung) verwenden. (Für Details sei auf die Dokumentation von `hyperref` hingewiesen.)

Die `{hyperworksheet}` Umgebung akzeptiert die gleichen Optionen von `{worksheet}`. Einige Makros/Umgebungen bekommen eine besondere Bedeutung:

### Aufgabe 1 (Lückentexte)

Lückentexte in der Umgebung `{cloze}` funktionieren genau so: Das Makro `\fillhere` gibt wie gewohnt ihren unterstrichenen in der Variante Lösung, erzeugt aber eine

Box in der Variante für Schüler. Das optionale Argument funktioniert normal:

siehe , , .

Was NICHT funktioniert ist die Sternform, bzw. sie funktioniert, macht aber das gleiche (also keine Linie bis zum ).

### Aufgabe 2 (Wahr/falsch Tabelle)

Die Umgebung `{TF}` kann auch in der PDF angekreuzt werden.

	wahr	falsch
Aussage 1		
Aussage 2, die sehr sehr lang ist, so dass der Text mehr als eine Zeile braucht, um gesetzt zu werden		
Aussage 3		

### Aufgabe 3 (Grids)

Das Makro `\grid` ergibt in der Lösung das übliche Ergebnis, erzeugt aber in der Schülervariante eine mehrzeilig auffüllbare Box.

Ein Wort der Vorsicht: Die Boxen in den beiden Fällen sind nicht *genau* gleich groß, so leider werden sich (vor Allem vertikale) Abstände (und schlimmstenfalls auch Seitenumbrüche) i.A. unterscheiden in den Varianten mit/ohne Lösung.

### ○ Aufgabe 4 (Multiple Response)

Die Umgebung `{multresponse}` funktioniert genauso wie im normalen Fall.

a. Bei Phänomen A gilt...

bar bar bar bar bar bar bar bar bar bar bar bar bar bar bar bar  
baz baz baz baz baz baz baz baz baz baz baz baz baz baz baz baz

b. Bei Phänomen B gilt...

test test test test test test test test test test test test test test  
yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn

### ● Aufgabe 5 (Multiple Choice)

(Die Schwierigkeit bezieht sich auf  $\TeX$ , nicht auf den Inhalt...)

Die Umgebung `{multchoice}` funktioniert mehr oder weniger auch, aber nicht jeder PDF-Viewer schafft es, sie richtig darzustellen.

a. Frage A

foo

bar

baz

b. Frage B

foo

bar

baz

c. Frage C

foo

bar

baz

Die Idee einer `{multchoice}` Umgebung ist, dass nur eine Antwort richtig ist. Wenn man dann einen anderen Feld anklickt, so sollte ein zuerst angeklickter Radiobutton (aus derselben Frage) „weggehen“. Nun, das funktioniert nicht mit allen Viewern: Auf meinem Windows 10 Rechner hat es funktioniert nur mit den eingebauten Viewern von Chrome, Edge und Thunderbird. Es hat nicht funktioniert mit PDF-XChange Viewer und Okular. Adobe Acrobat Reader habe ich nur auf Handy und Tablet (beides Android) und er hat dort versagt. (Fairerweise muss ich sagen, dass *alle* PDF-Readers auf meinem Handy und Tablet versagt haben.)

Eine Wahr-Falsch-Tabelle ist eigentlich also eine sinnvolle (und rechtlich sicherere) Alternative. Wer sich mit `hyperref` auskennt, kann natürlich mit `\ChoiceMenu` spielen.

### Aufgabe 6 (Lückentexte als graue Boxen)

Das Makro `\fillbox` funktioniert auch: Im Textmodus ist es natürlich genau dasselbe wie `\fillhere`. Im Mathe-Modus skaliert er auch wie gewohnt, aber das Ergebnis ist noch nicht gaaaanz schön, aber es geht:

$$3^2 = 9, \quad 7^{1/2} = 7.$$

### ● Aufgabe 7 (Schlussbemerkung)

Ich habe gerade keinen Ansatz für `{matching}`. Dafür kann man im Grunde eine Tabelle mit `\fillhere` oder `\fillbox` aufsetzen.

Wenn man das Blatt online verteilen will, ist es natürlich wesentlich sinnvoller, die Sternform der Umgebung, `{hyperworksheet}`, zu verwenden. Nutzt man die normale Variante, so muss man danach noch die Seite(n) ohne Lösung extrahieren.

MIT VORSICHT GENIESSSEN!

Ob und wie genau PDF-Forms funktionieren (oder nicht), hängt *sehr* vom verwendeten PDF-Viewer ab! Wenn etwas nicht läuft, dann nicht sofort mit den Schülern/Studenten schimpfen! Vielleicht trifft sie doch ausnahmsweise keine Schuld...

Zuerst muss im Code vor der Umgebung `{hyperworksheet}` das Makro `\Form` aufgerufen werden. Alternativ, kann man das Arbeitsblatt zwischen `\begin{Form}` und `\end{Form}` setzen. Pro Datei kann man nur *einmal* `\Form` (bzw. eine `{Form}` Umgebung) verwenden. (Für Details sei auf die Dokumentation von `hyperref` hingewiesen.)

Die `{hyperworksheet}` Umgebung akzeptiert die gleichen Optionen von `{worksheet}`. Einige Makros/Umgebungen bekommen eine besondere Bedeutung:

### Aufgabe 1 (Lückentexte)

Lückentexte in der Umgebung `{cloze}` funktionieren genau so: Das Makro `\fillhere` gibt wie gewohnt ihren unterstrichenen Inhalt in der Variante mit Lösung, erzeugt aber eine ausfüllbare Box in der Variante für Schüler. Das optionale Argument funktioniert normal: siehe hier , hier , hier .

Was NICHT funktioniert ist die Sternform, bzw. sie funktioniert, macht aber das gleiche (also keine Linie bis zum Zeilenende).

### Aufgabe 2 (Wahr/falsch Tabelle)

Die Umgebung {TF} kann auch in der PDF angekreuzt werden.

	wahr	falsch
Aussage 1	<input checked="" type="radio"/>	<input type="radio"/>
Aussage 2, die sehr sehr lang ist, so dass der Text mehr als eine Zeile braucht, um gesetzt zu werden	<input type="radio"/>	<input checked="" type="radio"/>
Aussage 3	<input type="radio"/>	<input checked="" type="radio"/>

### Aufgabe 3 (Grids)

Das Makro \grid ergibt in der Lösung das übliche Ergebnis, erzeugt aber in der Schülervariante eine mehrzeilig auffüllbare Box.

Denken ist überbewertet, daher versuche ich, es zu unterlassen. Ach ja, in der füllbaren Variante gibt es keinen Unterschied zwischen \* und + als erstes Argument.

Ein Wort der Vorsicht: Die Boxen in den beiden Fällen sind nicht *genau* gleich groß, so leider werden sich (vor Allem vertikale) Abstände (und schlimmstenfalls auch Seitenumbrüche) i.A. unterscheiden in den Varianten mit/ohne Lösung.

○ **Aufgabe 4 (Multiple Response)**

Die Umgebung {multresponse} funktioniert genauso wie im normalen Fall.

- a. Bei Phänomen A gilt...  
☒ bar bar bar bar bar bar bar bar bar bar bar bar bar bar bar bar  
☒ baz baz baz baz baz baz baz baz baz baz baz baz baz baz baz baz
- b. Bei Phänomen B gilt...  
☒ test test test test test test test test test test test test test test  
☐ yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn yawn

### ● Aufgabe 5 (Multiple Choice)

(Die Schwierigkeit bezieht sich auf  $\TeX$ , nicht auf den Inhalt...)

Die Umgebung `{multchoice}` funktioniert mehr oder weniger auch, aber nicht jeder PDF-Viewer schafft es, sie richtig darzustellen.

a. Frage A

- ☐ foo
- ☒ bar
- ☐ baz

b. Frage B

- ☐ foo
- ☐ bar
- ☒ baz

c. Frage C

- ☒ foo
- ☐ bar
- ☐ baz

Die Idee einer `{multchoice}` Umgebung ist, dass nur eine Antwort richtig ist. Wenn man dann einen anderen Feld anklickt, so sollte ein zuerst angeklickter Radiobutton (aus derselben Frage) „weggehen“. Nun, das funktioniert nicht mit allen Viewern: Auf meinem Windows 10 Rechner hat es funktioniert nur mit den eingebauten Viewern von Chrome, Edge und Thunderbird. Es hat nicht funktioniert mit PDF-XChange Viewer und Okular. Adobe Acrobat Reader habe ich nur auf Handy und Tablet (beides Android) und er hat dort versagt. (Fairerweise muss ich sagen, dass *alle* PDF-Readers auf meinem Handy und Tablet versagt haben.)

Eine Wahr-Falsch-Tabelle ist eigentlich also eine sinnvolle (und rechtlich sicherere) Alternative. Wer sich mit `hyperref` auskennt, kann natürlich mit `\ChoiceMenu` spielen.

### Aufgabe 6 (Lückentexte als graue Boxen)

Das Makro `\fillbox` funktioniert auch: Im Textmodus ist es natürlich genau dasselbe wie `\fillhere`. Im Mathe-Modus skaliert er auch wie gewohnt, aber das Ergebnis ist noch nicht gaaanz schön, aber es geht:

$$3^2 = 9, \quad 49^{1/2} = 7.$$

### ● Aufgabe 7 (Schlussbemerkung)

Ich habe gerade keinen Ansatz für `{matching}`. Dafür kann man im Grunde eine Tabelle mit `\fillhere` oder `\fillbox` aufsetzen.

Wenn man das Blatt online verteilen will, ist es natürlich wesentlich sinnvoller, die Sternform der Umgebung, `{hyperworksheets}`, zu verwenden. Nutzt man die normale Variante, so muss man danach noch die Seite(n) ohne Lösung extrahieren.