Schiffeversenken in Python - SS2021

Marc Ullmann, David Ruschmaritsch, Anne Lotte Müller-Kühlkamp

Frankfurt University of Applied Sciences
(1971-2021: Fachhochschule Frankfurt am Main)
Nibelungenplatz 1
60318 Frankfurt am Main
marc.ullmann@stud.fra-uas.de, david.ruschmaritsch@stud.fra-uas.de,
anne.mueller-kuehlkamp@stud.fra-uas.de

Zusammenfassung Hier kommt das Abstract hin. Das Abstract sollte in wenigen Sätzen (weniger als 10 Zeilen) den Inhalt des Dokuments beschreiben. Wikipedia schreibt: Ein Abstract ist eine prägnante Inhaltsangabe, ein Abriss ohne Interpretation und Wertung einer wissenschaftlichen Arbeit. Das Abstract soll so kurz wie möglich sein. Alle wesentlichen Sachverhalte sollen explizit enthalten sein. Das Abstract soll beim Leser Interesse für den Inhalt des Dokuments wecken.

Das Spiel Schiffeversenken, welches früher nur mit Stift und Papier gespielt wurde und sich seit dem in seiner Grundstruktur kaum verändert hat, bietet in der Implementierung die optimalen Grundlagen der objektorientierten Programmierung und grafischen Darstellung.

1 Grundprinzip Schiffeversenken

Für die Implementierung des Spiels wird ein Spielfeld benötigt, welches mit Kommandozeilenargumenten anpassbar sein soll. Die Beschränkung von einer Mindestgröße von 10x10 Feldern und Maximalgröße von 20x20 Feldern erwies sich als sinnvoll, da bei mehr als 20 Feldern die Standardgröße des Standard-Fensters überschritten wird und somit das Programm abstürzt. Bei weniger als zehn Feldern könnte es zu Platzproblemen der Schiffe mit dem zufälligen Platzier-Algorithmus geben und somit unerwünschte Ereignisse hervorrufen. Deshalb wird bei Eingabe von zwe Kommandozeilenargumenten die Begrenzung abgefragt, die bei Abweichung die Standardgröße von 10x10 wählt.

Für die Implementierung von Schiffen ist nun die Grundlage des Spielfelds gegeben. Ein Schiff ist als Objekt zu definieren. Die Schiffanzahl ist von grund auf festgelegt auf insgesamt zehn Schiffe (1x5 2x4 3x3 4x2).

2 Visuelle Darstellung

2.1 Wahl der Bibliothek

Zu Beginn kam es zur Wahl der Bibliothek für die visuelle Darstellung. Zuerst fiel die Wahl auf termbox [**?**]. Da diese jedoch nicht mehr unterstützt und weiterentwickelt wird, fiel die Entscheidung zuerst auf cursebox [**?**], für welche die einfachen Befehle und praktische Funktionen sprachen. Da auch diese nicht weiterhin unterstützt wird und wenig Dokumentation zu der Bibliothek vorliegt, wurde das Programm in curses [**?**] übersetzt.

2.2 Spielfeld

Das Spielfeld wurde mit Hilfe von der Extension numpy [**?**] entwickelt. Die Extension bietet eine einfache Erstellung mehrdimensionaler Arrays und liefert verschiedene Operationen diese zu konfigurieren. Ein zweidimensionales Array bietet die optimale Grundlage für das Spielfeld, da dies wie ein 2D-Koordinatensystem betrachtet werden kann. Doch ist es mit einem numpy Array nur möglich Zahlen zu verarbeiten. Dies hat zur Auswirkung, dass Felder die beispielsweise mit Schiffen belegt sind als „1“ definiert sind und Felder ohne Schiffe eine „0“. Weitergehend bedeutet dies, dass die grafische Darstellung des numpy Arrays nicht sonderlich ansehnlich erscheint, da typischerweise Schiffe mit einem „X“ oder anderen eindeutigeren Zeichen gekennzeichnet sind. Das Chararray, welches Zeichen speichern kann, bietet sich hier an. Jedoch wird bei der Darstellung des Arrays immer ein „b“ für Byte hinzugefügt, da es Bytestrings sind, die in dem Array gespeichert werden. Dies wurde umgangen indem der Datentyp des Arrays als Zeichenkette übersetzt und dann ausgegeben wird. Für die Darstellung eines Spielfeldes und nicht der Hinternanderreihung einzelner Objekte des Arrays, wird jede Reihe des Arrays einzeln in einer eigenen Zeile ausgegeben.

Um die Vorteile beider Arrays zu benutzen, müssen verschiedene Arrays erstellt werden. Insgesamt gibt es fünf Arrays: Zwei Charrays für die grafische Darstellung und drei Null-Arrays für die logische Verarbeitung.

2.3 Spielfeld: Schiffplatzierung

Das erste logische Array speichert die temporären Daten, welche für die Berechnungen von Spielfeldbegrenzung dienen. Das zweite speichert tempöräre Daten, wenn diese die Regeln nicht verletzen. Für die Platzierung der Schiffe wird nun in dem tempörären Array bestimmt, ob die Schiffsgröße die Spielfeldgröße überschreitet. Wenn schon Schiffe platziert wurden gleicht es zudem mit dem zweiten logischen Array die Position dieser ab und berechnet so die Möglichkeit der

Platzierung. Bei der Platzierung werden die umliegenden Felder reserviert, da die Schiffe sich nicht direkt berühren dürfen. Das dritte logische Array speichert hingegen nur die Schiffspositionen ab, um diese später zu übergeben.

In diesem Fall wird nur das eine Charray benutzt, welches die ersten beiden logischen Arrays visuell übersetzt. Dafür wird ein „X“

2.4 Spielfeld: Beschuss

3 Problem: Mausimplementation

Grundsätzlich soll das Spiel mit der Tastatur spielbar sein, jedoch gab es in den Anforderungen auch die Wahl eine Maussteuerung einzubinden. Die benutzte Programmbibliothek „curses“, welche zur Darstellung zeichenorientierter Benutzerschnittstellen unabhängig vom darstellenden Textterminal ist, bietet hierzu die Funktion „getMouse()“ an. Diese Funktion liefert ein 5-stelliges Tupel. Darin enthalten sind unter Anderem die x- und y-Koordinaten der Maus, welche mit einem Tastenereignis (hier ein Mausklick) abzufragen sind. Nun gilt es die in dem curses-Terminal dargestellten Texte, welche als Knöpfe dienen sollen, per Mausklick zu aktivieren. Dazu müssen die Koordinaten der Maus mit denen der dargestellten Texte übereinstimmen. Dies erwies sich als umständlich und komplex für die weitere Implementierung des Spiels, da die dargestellten Strings keine Objekte sind und damit keine Funktionen zur Abfrage und dem damit verbundenen Vergleich der Mausposition hatten und man diese aufwendig herausfinden müsste.

4 Benutzereingaben

Für die Entgegennahme der Eingaben des Benutzers wurde die Funktion „userinput()“ entwickelt.

5 Die Klasse Ship

Alle Schiffe sind als Objekte definiert. Objekte bieten hier einen erheblichen Vorteil durch deren Attribute. In jedem Schiff werden die Größe, Rotation, Startposition und Koordinaten gespeichert. Die Koordinaten berechnen sich durch die Objektmethode „ship$_c ords_1, welche jenach Ausrichtung(horizontal, vertikal) eine Schleife bis zur Grö„edes Schiffs durchläuft

5.1 Unformatierter Text

Die Fähigkeit von LaTeX, unformatierten Text mit fester Zeichenbreite darzustellen, ist besonders bei der Einbindung von Quellcode hilfreich.

Zum setzen von unformatiertem Text gibt es die Umgebung `verbatim`. Den Inhalt der Umgebung setzt LaTeX in der Schrift `TypeWriter` (Schreibmaschinenschrift) und interpretiert ihn nicht. Er kann also auch Sonderzeichen von LaTeX enthalten. Leerzeichen und Zeilenumbrüche werden einfach übernommen und gedruckt. Es können keine Befehle mehr ausgeführt werden bis zum `\end{verbatim}`. Die Umgebungen `verbatim` sorgt zudem dafür, dass der unformatierte Text abgesetzt dargestellt ist. Die folgenden Zeilen sind ein Beispiel für

```
Das ist ein Test mit der Umgebung verbatim.
Hier kann man einfach Sonderzeichen eingeben.
§ $ % & / | \ ~ * # - -- ---
```

Es existiert eine Möglichkeit, maximal eine Zeile langen Text, unformatiert auszugeben. Dabei handelt es sich um den Befehl `\verb<Zeichen>Text<Zeichen>`.

Der betreffende Text ist dabei von einem (fast) beliebigen Zeichen umschlossen. Es kann sich zum Beispiel um ein `!`, `§`, `|` oder `+` handeln. Das Zeichen muss direkt nach dem `\verb`-Befehl angegeben sein. Beim ersten Auftreten beginnt die `verbatim`-Umgebung und beim nächsten Auftreten des Zeichens endet die Umgebung. Darum darf das Zeichen sich auch nicht innerhalb der Umgebung (des darzustellenden Textes) befinden.

5.2 Sonderzeichen und Fortsetzungspunkte

Ein paar Sonderzeichen: \, $, &, €, %, #, _, ~, ^, |, {, }

Weitere Sonderzeichen: ©, ®, ™, §, ¶, £, †, ‡, •

Fortsetzungspunkte macht das Kommando `\dots`. Ergebnis: ...

5.3 Trennstrich, Gedankenstrich, längerer Gedankenstrich

LaTeX unterscheidet drei verschiedene Arten von Strichen in gedrucktem Text.

Es gibt den *Trennstrich*, der als Trennzeichen beim Trennen von Wörtern, oder als *Bindestrich* in zusammengesetzten Begriffen benutzt wird. Diesen schreibt man wie einen ganz gewöhnlichen Strich (-).

Die zweite Variante ist der *Gedankenstrich* –. Diesen verwendet man häufig bei Strecken- oder Zeitangaben. Die Realisierung geschieht mit zwei Strichen (`--`). Vor und hinter diesem Gedankenstrich sollten sich Leerzeichen befinden.

Im englischen Sprachraum existiert der *längere Gedankenstrich* —. Die Realisierung geschieht mit drei Strichen (`---`). Er ist für gewöhnlich ohne Leerzeichen mit den vorhergehenden und folgenden Textstellen verbunden. Im deutschen Sprachraum ist dieses Stilmittel eher unüblich.

5.4 Quellcode

Zum Setzen von Quellcode empfiehlt sich die Umgebung `lstlisting`. Zuvor definiert der Autor mit dem Kommando `\lstset`, um was für einen Quellcode es sich handelt und wie dieser formatiert sein soll. Listing **??** enthält ein Shell-Skript ein Beispiel für einen Quellcode. Jeder Quellcode sollte eine Unterschrift (Beschreibung) haben. Damit die Syntaxhervorhebung korrekt funktioniert, muss der Autor mit dem Schlüsselwort `language` die Programmiersprache angeben.

```
1  #!/bin/bash
2  #
3  # Skript: operanden2.bat
4  #
5  # Funktionsbibliothek einbinden
6  . funktionen.bib
7
8  echo "Bitte geben Sie den gewünschten Operator ein."
9  echo "Mögliche Eingaben sind: + - * /"
10 read OPERATOR
11 echo "Bitte geben Sie den ersten Operanden ein:"
12 read OPERAND1
13 echo "Bitte geben Sie den zweiten Operanden ein:"
14 read OPERAND2
15
16 # Eingabe verarbeiten
17 case $OPERATOR in
18   +)  add $OPERAND1 $OPERAND2 ;;
19   -)  sub $OPERAND1 $OPERAND2 ;;
20   \*) mul $OPERAND1 $OPERAND2 ;;
21   /)  div $OPERAND1 $OPERAND2 ;;
22   *)  echo "Falsche Eingabe: $OPERATOR" >&2
23       exit 1
24       ;;
25 esac
26
27 # Ergebnis ausgeben
28 echo "$OPERAND1 $OPERATOR $OPERAND2 = $ERGEBNIS"
```

Listing 1.1. Das ist die Unterschrift des Quellcodes

5.5 Auflistungen und Aufzählungen

- Auflistungen sind häufig sehr hilfreich, um Text zu strukturieren.
- Mit Auflistungen kann man Text auch zusammenfassen.
- Auflistungen macht man mit der `itemize`-Umgebung.
  - Verschachtelte Auflistungen sind auch kein Problem.
    - Man kann bis zu vier Ebenen verschachteln.
      - Mehr als zwei Ebenen sieht aber fast nie gut aus.

1. Es gibt natürlich auch Aufzählungen in LaTeX.
2. Aufzählungen macht man mit der `enumerate`-Umgebung.
   (a) Diese kann man auch nach belieben verschachteln.
3. Man kann natürlich auch...
   - Auflistungen und
   - Aufzählungen nach Belieben verschachteln.

5.6 Farben in Texten

Farben sollte man hier nicht oder nur im geeigneten (seltenen!) Fall einsetzen. Die Gründe sind:

- Farbige Texte erschweren das Lesen erheblich.
- Ein Farbdrucker ist nicht überall vorausgesetzt vorhanden.
- Nicht alle Farbdrucker erzeugen das gleiche Ergebnis.

5.7 Zitieren

Zitate können hilfreich sein. Wenn Sie wortwörtlich Texte übernehmen, machen Sie dies mit Einrückungen deutlich und geben Sie die Quelle an. Für Zitate gibt es u.a. die Umgebung `quotation`. Passend hierzu ein Zitat von Donald E. Knuth, dem Entwickler von TeX:

> Science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it. [**?**]

In naturwissenschaftlichen Abschlussarbeiten sollte der Bedarf an solchen Zitaten aber eher gering sein, da in der Regel der Entwurf und die anschließende Implementierung einer Lösung der Kern der Arbeit ist. Auch wenn Sie Bilder oder Tabellen übernehmen, müssen Sie in der Bildunterschrift bzw. Tabellenüberschrift die Quelle angeben. Blöd ist, wenn man am Ende Quellenangaben vergessen hat. Darum empfiehlt es sich, dass Sie zur Sicherheit ihre eigene Arbeit mit Werkzeugen wie PlagScan[1] untersuchen.

[1] http://www.plagscan.com

5.8 Fußnoten

Fußnoten können hilfreich sein, um Zusatzinformationen im Dokument unter zu bringen. Generell sollte man Fußnoten aber auch eher sparsam verwenden.

In LaTeX werden Fußnoten[2] mit dem Befehl `\footnote{Text der Fußnote}` erzeugt. Man braucht sich weder um die Nummerierung oder Positionierung der Fußnoten Gedanken zu machen.[3]

5.9 Bilder

Abbildungen fügen Sie, wenn möglich, als Vektorgrafiken ein. Besonders bei selbst erstellten Diagrammen ist das problemlos möglich. Im Gegensatz zu Rastergrafiken ist das skalieren von Vektorgrafiken ohne Qualitätsverlust stufenlos und verlustfrei möglich. Dateiformate für Vektorgrafiken sind u.a. `eps`, `ps`, `pdf` und `svg`. Dateiformate für Rastergrafiken sind u.a. `bmp`, `gif`, `jpg` und `png`.

Jede Abbildung benötigt eine Bildunterschrift mit eindeutiger Nummer und muss im Text referenziert sein (siehe Abbildung **??**.

Abbildung 1. Bildunterschrift von Abbildung **??**

Das Drehen von Bildern ist mit dem Parameter `angle=<Winkel>` einfach möglich. Abbildung **??** wurde um 270 Grad gedreht.

Abbildung 2. Bildunterschrift von Abbildung **??**

Soll ein Bild genauso breit sein wie das Textfeld, geht dieses einfach mit `width=\textwidth`.

5.10 Tabellen

LaTeX bietet viele Umgebungen, um Tabellen zu erzeugen. Eine einfache Umgebung ist `tabular`. Springer empfiehlt, bei Tabellen die Tabellenüberschrift über

[2] Eine Fußnote
[3] Eine weitere Fußnote

die Tabelle zu schreiben und keine senkrechten Trennlinien zu verwenden. Eine weitere Empfehlung von Springer ist, nur zur Begrenzung der eigentlichen Tabelle, sowie des Tabellenkopfs, horizontale Trennlinien zu ziehen. Ein Beispiel für eine solche Tabelle ist Tabelle **??**.

Jede Tabelle benötigt eine Tabellenüberschrift mit eindeutiger Nummer und muss im Text referenziert sein.

Tabelle 1. Eine einfache Tabelle

Zeile	Linksbündig	Zentriert	Rechtsbündig
1	Zeile 1	Zeile 1	Zeile 1
2	Zeile 2	Zeile 2	Zeile 2
3	Zeile 3	Zeile 3	Zeile 3

Klassischerweise sehen Tabellen eher aus, wie Tabelle **??**, mit senkrechten und horizontalen Trennlinien an allen Feldgrenzen. Das Ergebnis ist natürlich korrekt, sieht aber nicht so elegant aus, wie Tabelle **??**.

Tabelle 2. Eine Tabelle im klassischen Layout

Zeile	Linksbündig	Zentriert	Rechtsbündig
1	Zeile 1	Zeile 1	Zeile 1
2	Zeile 2	Zeile 2	Zeile 2
3	Zeile 3	Zeile 3	Zeile 3

5.11 Korrektes Arbeiten mit Zahlen und Währungen

Vermeiden Sie Bewertungen, die sie nicht belegen und in Zahlen festmachen können. Formulierungen wie „einfach“, „schwierig“, „langsam“, „schnell“, „günstiger Preis“ oder „hoher Datendurchsatz“ sind nicht eindeutig und haben in einer wissenschaftlichen Dokumentation nichts zu suchen. Fragen Sie sich selbst: Wie viel ist viel? Wie viel Geld ist günstig? Wie viele MB/s sind schnell? Solche Formulierungen sind immer vom Kontext abhängig und nicht eindeutig. Wenn Sie etwas bewerten, muss es auf Basis von Quellen oder eigenen Messergebnissen geschehen.

In deutschsprachigen Dokumenten schreibt man Währungssymbole nach dem Betrag (Beispiel: 10 €) mit einem geschützten Leerzeichen zwischen Währungssymbol und Betrag. Ein solches Leerzeichen darf nicht umbrochen werden. In englischsprachigen Dokumenten werden Währungssymbole vor den Betrag ge-

setzt (Beispiel: $10). Dabei ist kein Leerzeichen zwischen Währungssymbol und Betrag gesetzt.

Das Dezimaltrennzeichen markiert die Grenze zwischen dem ganzzahligen Teil und dem gebrochenen Teil einer Zahl. In deutschsprachigen Dokumenten verwendet man ein Komma als Dezimaltrennzeichen (Dezimalkomma). In englischsprachigen Dokumenten verwendet man einen Punkt als Dezimaltrennzeichen (Dezimalpunkt).

Bei großen Zahlen empfiehlt es sich zur besseren Lesbarkeit Tausendertrennzeichen zu verwenden. In deutschsprachigen Dokumenten verwendet man als Tausendertrennzeichen einen Punkt (Beispiel: 10.000 €) oder ein geschütztes Leerzeichen (Beispiel: 10 000 €). In englischsprachigen Dokumenten verwendet man als Tausendertrennzeichen ein Komma (Beispiel: $10,000).

5.12 Qualität der Sprache

Qualität kommt von Qual und nicht der Leser soll sich quälen, sondern der Autor!

Formulieren Sie sachlich und präzise. Ihr Text sollte frei von weitschweifigen, prosaischen und umständlichen Formulierungen sein.

Vermeiden Sie zu lange Sätze, die über mehrere Zeilen gehen. Der Text soll flüssig lesbar sein.

Formulieren Sie aktiv. Das erreichen Sie, indem Sie Wörter wie „wird“, „wurde“ und „werden“ vermeiden.

Vermeiden Sie in einer deutschsprachigen Abschlussarbeit unnötige Anglizismen. Wörter wie Webseite, E-Mail, Server und Browser stehen im Duden und sind quasi eingedeutscht. Für zahlreiche andere Wörter in der Informatik existieren seit Jahrzehnten etablierte deutsche Begriffe. Es bietet sich an, diese auch zu verwenden.

Schreiben in der dritten Person. Das heißt Sie vermeiden die Ich-Form (z.B. „Ich/Wir habe(n) den Datendurchsatz mit xyz gemessen...“) und das direkte Ansprechen des Lesers (z.B. „in meiner/unserer Arbeit sehen Sie...“). Wahren Sie Distanz zum Leser (z.B. „Die Messung des Datendurchsatzes mit xyz ergab...“).

Verfallen Sie im Überschwang nicht in Werbesprache („BlaBla-Sprache“), indem Sie bestimmte Produkte oder Unternehmen euphorisch anpreisen. Wenn Sie Abschnitte mit Werbetexten von amerikanischen Produktseiten übersetzen und einbauen, merkt der Leser das.

Bei eigenen Werken wird man als Autor mit der Zeit betriebsblind und ließt über schlechte Formulierungen hinweg. Wenn Sie mit der Arbeit inhaltlich fertig sind, lesen Sie sich die Arbeit von Anfang bis Ende einmal selbst laut vor. Schlechte Formulierungen und Wortwiederholungen erkennen Sie so sofort.

Vermeiden Sie den Deppenapostroph[4].

Beachten Sie die korrekte Verwendung des Bindestrichs. Merke: Wenn Sie ein englisches Wort mit einem deutschen Wort verbinden, brauchen Sie zwingend einen Bindestrich. Beispiele sind „Cloud-Dienst“, „Grid-Ressource“ und „Client-Server-Anwendung“.

5.13 Layout

Unter jede Überschrift gehört ein Text. Es sollte zum Beispiel nicht direkt unter einer Kapitelüberschrift die Überschrift eines Abschnitts folgen.

Absätze sollten nicht zu lang sein, aber nach Möglichkeit auch nicht nur aus einem einzelnen Satz bestehen.

Achten Sie auf einen ausreichend großen Rand. Dieser erleichtert die Korrektur der Arbeit und wertet diese optisch auf. Es ist kein Ziel Ihrer Arbeit, möglichst viel Text auf eine Seite zu quetschen. LaTeX und die `llncs`-Vorlage machen das automatisch.

Verwenden Sie Blocksatz und keinen Flattersatz. LaTeX und die `llncs`-Vorlage machen das automatisch.

5.14 Literaturangaben

Die Literaturangaben (und der Bezug auf diese!) sind von entscheidender Wichtigkeit, denn sie zeigen, dass der Autor belesen und in der Materie drin ist. Das beziehen auf anerkannte Quellen, ist ein Eckpfeiler wissenschaftlicher Arbeit. Bei großen Dokumenten empfiehlt sich der Einsatz von BibTeX. Dabei sind die Literaturquellen in einer `.bib`-Datei zentral gesammelt, die einmal im Dokument importiert wird. Es erscheinen nur die Literaturquellen im fertigen Dokument, auf die referenziert wurde und das Layout der Quellen steuert BibTeX.

Für kleinere Dokumente ist BibTeX häufig nicht notwendig. Hier genügt die Umgebung `thebibliography`. Jeder neue Eintrag beginnt mit dem Befehl `\bibitem{Marke}`. Im Fließtext ist es möglich via `\ref{Marke}` auf diese Marke zu verweisen. Ein Vorteil von `thebibliography` ist, dass die Anwendung sehr einfach ist. Der Nachteil ist, dass alle Änderungen manuell erfolgen.

[4] http://www.deppenapostroph.info

6 Schlusswort

Das Schlusswort enthält eine Zusammenfassung des Dokuments. Eine Art Fazit. Hier sind die wichtigsten Erkenntnisse bzw. Ergebnisse noch einmal knapp und präzise in wenigen Sätzen zusammengefasst. Zahlreiche Autoren empfinden die Arbeit am Schlusswort und Abstract als lästige Arbeit. Dennoch sollte man sich als Autor hier besonders viel Mühe geben, denn zahlreiche Leser entscheiden nur anhand von Inhalt und Qualität dieser beiden Teile, ob sie das ganze Dokument lesen möchten bzw. in Zukunft darauf verweisen werden.

Literatur

1. „nsf“, Termbox, Github. 2020
   https://github.com/nsf/termbox
2. „nsf“, Termbox, Github. 2020
   https://github.com/nsf/termbox
3. A.M. Kuchling, Eric S. Raymond, Cursebox, python.org. 2021
   https://docs.python.org/3/howto/curses.html
4. Numpy, numpy.org. 2021
   https://numpy.org/
5. A.M. Kuchling, Eric S. Raymond, Cursebox, python.org. 2021
   https://docs.python.org/3/howto/curses.html
6. Knuth, Donald E. *Computer Programming as an Art*. Communications of the ACM 17 (12). December 1974. S.667-673
   http://fresh.homeunix.net/ luke/misc/knuth-turingaward.pdf
7. LaTeX – A document preparation system. http://www.latex-project.org
8. Kopka, Helmut. *LaTeX, Band 1: Einführung*. Pearson. 2005
9. Kopka, Helmut. *LaTeX, Band 2: Ergänzungen*. Pearson. 2002
10. Kopka, Helmut. *LaTeX, Band 3: Erweiterungen*. Pearson. 2002
11. Mittelbach, Frank und Goossens, Michel. *Der LaTeX-Begleiter*. Pearson. 2005
12. Schlosser, Joachim. *Wissenschaftliche Arbeiten schreiben mit LaTeX: Leitfaden für Einsteiger*. Mitp-Verlag. 2008
13. Willms, Roland. *LaTeX: Für Schnelleinsteiger*. Franzis. 2006