

Einführung in die Informatik

05 Text

Prof. Dr. Carsten Link

Stand: 6. November 2024

Inhaltsverzeichnis

1	Kompetenzen und Lernegebnisse	1
2	Konzepte	2
2.1	Historische Entwicklung	2
2.2	Editieren von Texten	2
2.3	Manöver mit Maus und Tastatur	2
2.4	Differenzen zwischen Texten	3
2.5	Character Encodings	3
2.6	Fonts	5
2.7	Rendering von Texten	8
2.7.1	Markup	9
2.7.2	Markdown	10
2.7.3	WYSIWYG und WYSIWYM	10
3	Material zum aktiven Lernen	12
3.1	Zusammenfassender Lückentext	12
3.2	Vertiefungsaufgaben zu ‘Editieren von Texten’	12
3.3	Vertiefungsaufgaben zu ‘Character Encodings’	13
3.4	Vertiefungsaufgaben zu ‘Fonts’	13
3.5	Vertiefungsaufgaben zu ‘Rendering von Texten’	13
3.6	Verständnisfragen zu ‘Editieren von Texten’	13
3.7	Verständnisfragen zu ‘Character Encodings’	13
3.8	Verständnisfragen zu ‘Fonts’	14
3.9	Verständnisfragen zu ‘Rendering von Texten’	14
3.10	Testate zu ‘Unterschiede zwischen Texten’	14
3.11	Testate zu ‘Fonts’	14
4	Anhang: Literatur und weiterführendes Material	15

1 Kompetenzen und Lernegebnisse

Durch das Bearbeiten dieses Materialpaketes erwerben Sie diese Kompetenzen (Wissen, Fähigkeiten, Fertigkeiten zur Problemlösung):

Sie kennen einige unterschiedliche Kodierungen von Texten und können nachvollziehen, wie Texte vom Speicher auf den Bildschirm oder Papier gebracht werden und was es dabei zu beachten gilt.

Die oben genannten Kompetenzen erwerben Sie, indem Sie Lernziele erreichen, welche sich prüfen lassen. Lernegebnisse: Sie können nachweislich¹:

- gängige Texteditoren flüssig bedienen.
- Unterschiede zwischen verschiedenen Text Encodings erläutern.
- wichtige Steuerzeichen und deren Wirkung erklären.
- Eigenschaften von Schriftarten benennen.
- Schriftarten für verschiedene Zwecke auswählen.
- Bitmap- und Vektorschriftarten voneinander abgrenzen.
- erläutern, wie Texte auf den Bildschirm oder Papier gebracht werden.
- den Begriff Markup erläutern.
- Texte mit einfachen Markup-Anweisungen formatieren lassen.
- Texte mit einfachen Markdown-Markierungen formatieren lassen.

2 Konzepte

2.1 Historische Entwicklung

Nachdem Rechner sich für Berechnungen (in der Mathematik, Physik, Ingenieurwesen etc.) als sehr nützlich erwiesen haben, begann man Rechner auf für die Verarbeitung von Texten zu verwenden. Also sollte das Erstellen von Zeitungen, Büchern, Briefen, Werbeprospekten usw. vereinfacht werden. Unterstützung bieten Rechner vor allem beim Bearbeiten von Texten (schreiben, löschen, verschieben, suchen) und beim Textsatz (Typographie). Mit so genannten Desktop Publishing Systemen (DTP) sollte kleinen Firmen und Privatleuten möglich werden, was sonst nur Druckereien und Verlagen vorbehalten war.

2.2 Editieren von Texten

Bei der Programmierung oder der Administration von Rechnern wird häufig mit Textdateien gearbeitet. Diese lassen sich mit verschiedenen Texteditoren bearbeiten, die sich in ihrem Aussehen und ihrem Funktionsumfang unterscheiden. Probieren Sie die folgenden Texteditoren aus:

- Texteditoren mit GUI: pluma, gedit, featherpad, kate, geany, ...
- Texteditoren mit TUI (im Terminal): nano, joe, micro, tilde, ...

Viele Editoren helfen mit Automatisierung: Copy & Paste, Search, Search & Replace, Einrückung, Vervollständigung (completion), Rechtschreibprüfung.

¹Sie können das Erzielen der einzelnen Lernergebnisse beispielsweise bei einem Testat im Praktikum oder einer Aufgabe in der Modulprüfung nachweisen

2.3 Manöver mit Maus und Tastatur

Flüssiges Arbeiten wird durch Manöver mit Maus und Tastatur ermöglicht. Die wichtigsten Mausmanöver (teilweise auch per Touchpad nutzbar):

Kontext	Mausmanöver	Wirkung
Anything	Right Click	context menu
Anything	Left Click	select / set cursor
Text	Drag (left down & move)	mark character-wise
Text	Double Click	mark whole word
Text	Double Click & Drag	mark word-wise
Text	Triple Click	mark whole line
Text	Triple Click & Drag	mark line-wise

Die wichtigsten Tastenkürzel:

Kontext	Tastenkürzel	Wirkung
Editor	SHIFT – ARROW	mark
Editor	POS1	cursor → beginning
Editor	END	cursor → end
Editor	CTRL – ARROW	jump word-wise
pluma	CTRL – ALT – PAGEUP	cycle tabs
pluma	CTRL – ALT – PAGEDOWN	cycle tabs

Auf deutschen Tastaturen ist die CTRL-Taste mit „STRG“ beschriftet; auf Apple Rechnern mit „command“.

2.4 Differenzen zwischen Texten

Oft haben Programmierer und Autoren verschiedene Versionsstände von Dokumenten, da diese weiterentwickelt werden. Insbesondere, wenn mehrere Personen an Dokumenten arbeiten, ist es nützlich, Unterschiede zwischen zwei Versionsständen anzeigen zu lassen. Neben dem klassischen Kommando `diff` gibt es einige mit GUI – beispielsweise `meld` oder `diffuse`. Letzteres ist in Abbildung 1 zu sehen. Dort sind diese Unterschiede hervorgehoben: a) Zeile 1 1 zu 2 geändert, b) Zeile 3 gelöscht, c) Zeile 6 `dogw` zu `dog` geändert. d) Zeile 9 eingefügt. Tools, mit denen Differenzen angezeigt werden können sind: `diff`, `meld`, `diffuse`

2.5 Character Encodings

Analogien: morse, rosetta stone, TBD

Eine Zeichenkodierung nummeriert Zeichen durch; bildet also eine Tabelle mit einer Zuordnung von Zahlen zu Schriftsymbolen. Hier das klassische ASCII-Encoding (hexadezimal, ohne führendes 0x):

man ascii, the hexadecimal set							
00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si

10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 \$	25 %	26 &	27 '
28 (29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c	7d }	7e ~	7f del

Wichtige Steuerzeichen des ASCII-Zeichensatzes:

- new line (`0x0a nl`): neue Zeile; in Programmiersprachen oft `\n` geschrieben
- carriage return (`0x0d cr`): Wagenrücklauf; in Programmiersprachen oft `\r` geschrieben
- horizontal tab (`0x09 ht`): Tabulator; in Programmiersprachen oft `\t` geschrieben
- null (`0x00 nul`): meist Stringende; in Programmiersprachen oft `\0` geschrieben
- bell (`0x07 bel`): lässt eine Glocke läuten.
- escape (`0x1b esc`): in Programmiersprachen oft `\033` geschrieben

Oft genügt ein `\n`, um ein Zeilenende mit Zeilenumbruch zu markieren. Bei manchen Betriebssystemen oder Netzwerkprotokollen muss jedoch `\r\n`, `\n\r` oder `\r` verwendet werden.

Da die Zeichen des ASCII-Zeichensatzes in anderen Ländern als den USA nicht ausreichend sind, wurde viele weitere Zeichensätze entwickelt. Die meisten dieser Zeichensätze sind mit ASCII in großen Teilen kompatibel.

Wichtige europäische Zeichensätze:

- ISO-8859-15: westeuropäisch
- Windows-1252: westeuropäisch, Microsoft Windows
- Unicode: Der Unicode Zeichensatz kodiert viele gesprochene und historische Sprachen sowie gebräuchliche Symbole (siehe List of blocks on en.wikipedia.org²). Da die Anzahl der Unicode Zeichen stetig wächst, kann keine einfache Tabelle bzw. Abbildung auf 16- oder 32-Bit Zahlen gegeben werden. Stattdessen hat sich das UTF8-Format durchgesetzt, welches die Speicherung und Verarbeitung von Unicode-Zeichen auf Bytebasis erlaubt.

²https://en.wikipedia.org/wiki/Unicode_block#List_of_blocks

Werden Texte mit einem anderen als zur Kodierung verwendeten Character Encodings dekodiert, entsteht Mojibake. Hier ist zu sehen, wie das Schwedische Wort „Smörgås“ durch falsche Dekodierung zu Mojibake wird: Sm rg s, SmÅ¶rgÅ¥s, ë_C¶ÊÄCvË, Sm/ rg/ •s, Sm^rgÂs (siehe Wikipedia – Mojibake³)

Besonders wichtig ist das Encoding in diesen Situationen:

- Beim Öffnen einer Textdatei mit einem Editor.
- Beim Lesen von Daten aus einer Textdatei oder aus einer Netzwerkverbindung durch ein Programm (Java, C++ etc.).
- Beim Schreiben oder Lesen von Daten in eine Textdatei oder in eine Netzwerkverbindung durch ein Programm (Web-Server, Web-Browser, eMail-Client, etc.).
- Beim Verarbeiten von Zeichen und Zeichenketten durch ein Programm.

Ein Beispiel in Java, das eine Textdatei in UTF-8-Encoding öffnet (siehe Oracle Java Documentation – Character and Byte Streams⁴):

```
1 FileInputStream fis = new FileInputStream("test.txt");
2 InputStreamReader isr = new InputStreamReader(fis, "UTF8");
```

Ein weiteres Beispiel in Java, das eine String aus einem Byte-Array erzeugt: Java Tutorials Code Sample – StringConverter.java⁵

```
1 String original = new String("A" + "\u00ea" + "\u00f1" + "\u00fc" + "C");
2 byte[] utf8Bytes = original.getBytes("UTF8");
3 String roundTrip = new String(utf8Bytes, "UTF8");
```

Oben ist in der letzten Zeile zu sehen, das bei der Umwandlung roher Bytes in einen String ein Encoding angegeben werden muss.

2.6 Fonts

Eine Schriftart (engl.: font) bestimmt, wie die Zeichen auf Ausgabemedien (Papier, Bildschirme etc.) konkret aussehen. Für verschiedene Zwecke werden unterschiedliche Schriftarten verwendet. So werden auf Plakate, große auffällige Schriften verwendet; in Büchern gut lesbare und gefällige.

In diesem Dokument werden verschiedene Schriftarten verwendet: Überschriften, Verbatim, Fließtext. Im Fließtext werden bestimmte Zeichenketten typographisch hervorgehoben, um deren Typ kenntlich zu machen (Kommandos, Quelltexte, etc.). In Abbildung 2 ist ebenso zu sehen, wie typographische Hervorhebung durch Schriftarten, Farben von Text und Hintergrund etc. genutzt werden kann, Eigenschaften von Texten deutlich zu machen. So sind programmiersprachliche Konstrukte im Fließtext als solche erkennbar und Quelltext ist durch das Syntax Highlighting besser zu erfassen.

Für Informatiker ist die Auswahl von Schriftarten für Code, das Terminal und für Fließtext wichtig, um nicht durch ungeeignete Darstellung von Information beeinträchtigt zu werden.

Eigenschaften von Fonts für Code und Terminal:

³<https://en.wikipedia.org/wiki/Mojibake>

⁴<https://docs.oracle.com/javase/tutorial/i18n/text/stream.html>

⁵<https://docs.oracle.com/javase/tutorial/i18n/text/examples/StringConverter.java>

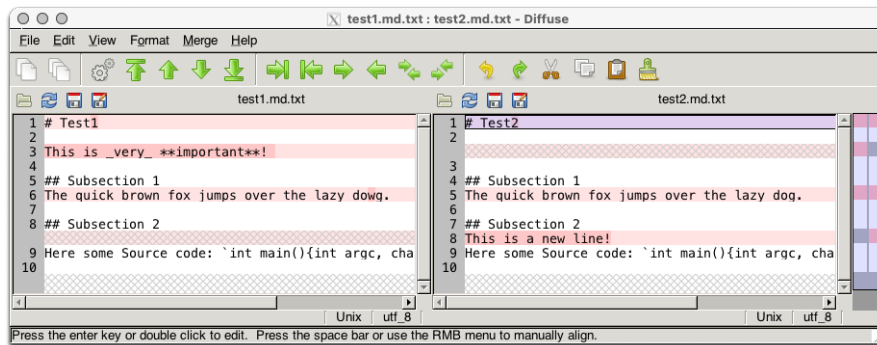


Abbildung 1: Unterschiede zwischen den Dateien `test1.md.txt` und `test2.md.txt` in dem Werkzeug `diffuse`.

2.6 Implizite Typumwandlungen zu bool

Werte der integralen Typen (`int`, `char`, etc.) können vom compiler implizit in `bool`-Werte umgewandelt werden. Hierbei entsprechen die Nullwerte `false` und alle anderen Werte werden in den `bool`-Wert `true` umgewandelt.

```

1 void implicit_bool(){
2     bool yes = true;
3     if (yes) {
4         println("yes is true");
5     }
6     char c = '\n';
7     if (c) {
8         println("c is truthy");
9     }

```

7

CPP-MP-03a-DATA-vars v1.9.1 ©Prof. C. Link 2022

Abbildung 2: Ausschnitt eines Vorlesungskriptes mit verschiedenen Schriftarten zur typographischen Hervorhebung.

- Monospaced (nichtproportional) für Einrückungen und Tabellen
- gut sichtbare Klammern
- Unterscheidbarkeit der Zeichen `ilL100` etc.

Wichtige Eigenschaften von Fonts für Fließtext:

- Gefälligkeit, gute Gestaltung
- Serifen: Helfen beim Lesen, da sie für das Auge eine Linie bilden („sans serif“ vs. „serif“).
- Ligaturen: Sonderzeichen für bestimmte Buchstabenkombinationen („fi“ vs. „fj“).
- Kerning: Abstand zwischen bestimmten Buchstabenpaaren.

Computerschriftarten lassen sich grob in Bitmap- und Vectorschriftarten einteilen. Hierbei ist die digitale Kodierung der einzelnen Zeichen gemeint. Bitmapschriftarten sind auf Rasterdisplays sehr scharf darstellbar und lassen sich sehr kompakt kodieren⁶, lassen sich jedoch nicht gut vergrößern oder verkleinern (skalieren). Vectorschriftarten skalieren sehr gut, benötigen aber ein Ausgabemedium mit hoher Auflösung (Papier, HiDPI-Displays).

In Abbildung 3 ist ein Auszug der Schriftart des Hitachi HD44780U LCD-Controllers zu sehen, der aufgrund der geringen Pixelanzahl eine grobe Bitmapschriftart verwendet. Das Java-Projekt LCD-Simulator⁷ simuliert ein LCD-Display mit einem solchen Controller.











































































HD44780U																	
Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A02)																	
Lower 4 Bits	Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)																
xxxx0001	(2)																
xxxx0010	(3)																
xxxx0011	(4)																
xxxx0100	(5)																

Abbildung 3: Schriftart des Hitachi HD44780U LCD-Controllers.

Hier ist zu sehen, wie ein Zeichen einer Bitmapschriftart kodiert werden kann:

⁶Ein Buchstabe einer 10x8 Bitmapschriftart benötigt nur 10 Byte Speicherplatz, da pro Pixel nur ein Bit benötigt wird. Die Datei `cmtt10.t3` hat im Vergleich dazu ca. 1700 Byte pro Zeichen.

⁷https://www.technik-emden.de/~clink/projects/2016w-ProjGrp/05_Website_ohne_Quellcodes/11_Dokumente/13_Doxygen/doc/html/index.html

```

Zeichen „K“ als Bitmuster im Quelltext des LCD-Simulators
public static boolean[] zeichen_0x4B = { // K
    X, _, _, _, X,
    X, _, _, X, _,
    X, _, X, _, _,
    X, X, _, _, _,
    X, _, X, _, _,
    X, _, _, X, _,
    X, _, _, _, X,
    _, _, _, _, _};

```

Hier ist zu sehen, wie ein Zeichen einer Vectorschriftart kodiert werden kann:

```

Zeichen „o“ als Postscript-Funktion (Computer Modern cmtt12.t3)
0 0 0 setrgbcolor
newpath 255.49643 379.94165 moveto
218.78658 379.94165 184.78856 361.97224 160.93776 333.99045 curveto
134.69179 303.19861 121.77042 263.5237 121.77042 223.05397 curveto
121.77042 179.45609 134.09988 136.37297 160.93776 102.0211 curveto
184.11685 72.35242 217.97195 51.88782 255.49643 51.88782 curveto
293.0209 51.88782 326.876 72.35242 350.0551 102.0211 curveto
376.89297 136.37297 389.22244 179.45609 389.22244 223.05397 curveto
389.22244 263.5237 376.30107 303.19861 350.0551 333.99045 curveto
326.2043 361.97224 292.20628 379.94165 255.49643 379.94165 curveto
255.49643 437.59555 lineto
309.40608 437.59555 359.8672 412.57953 395.71442 372.2447 curveto
434.19424 328.9477 453.7944 272.4135 453.7944 214.47322 curveto
453.7944 157.15799 434.0553 101.32358 395.71442 58.74034 curveto
359.7181 18.76111 309.24893 -5.76608 255.49643 -5.76608 curveto
201.74393 -5.76608 151.27478 18.76111 115.27844 58.74034 curveto
76.93756 101.32358 57.19846 157.15799 57.19846 214.47322 curveto
57.19846 272.4135 76.79861 328.9477 115.27844 372.2447 curveto
151.12567 412.57953 201.58678 437.59555 255.49643 437.59555 curveto
closepath fill

```

In dem oben gezeigten Postscript-Code werden vor allem die Funktionen `moveto`, `curveto` und `lineto` verwendet, um den Buchstaben „o“ zu zeichnen. Das Ergebnis ist in Abbildung 4 zu sehen. Bei der starken Vergrößerung in der Abbildung wird deutlich, wie gut sich Vektorschriftarten skalieren lassen.

2.7 Rendering von Texten

to render: machen, wiedergeben, vortragen, verkünden, übergeben, übersetzen. Eine Form von Übersetzung; jedoch mit Interpretationsspielraum (im Gegensatz zum Compiler)

Genau genommen ist bereits ein Texteditor ein Renderer für Texte. Folgende Parameter bestimmen das Aussehen eines Textes im Editor: Schriftart, Zeilenabstand, Zeilenumbrüche, Wortumbrüche, Syntax Highlighting. So sieht ein einziger Text in verschiedenen Editoren leicht unterschiedlich aus.

Die Komponente, die das Rendering macht, lässt sich oft steuern, um das Aussehen der Texte beeinflussen zu können. Neben manueller Zuweisung in Wortprozessoren (Word, LibreOffice Writer etc.) existieren Textmarkierungen „Markup“ und „Markdown“ – beide nachfolgend vorgestellt.

2.7.1 Markup

Markup (engl. to mark up: auszeichnen) findet Verwendung bei HTML und dem Textsatzsystem \LaTeX , sowie in dem Dateiformat `RTF`. Letzteres soll nun als Beispiel für Markup-Auszeichnungen genommen werden.

In Abbildung 5 sind vier verschiedene Schriftarten in einem einzigen Rich Text-Dokument (`RTF`-Datei) zu sehen. Dabei ist zu sehen, wie der `RTF`-Editor einiges je nach Schriftart unterschiedlich behandelt. So werden bei den Proportionalschriften die Zeichen unterschiedlich breit gesetzt und es werden Ligaturen verwendet. Da unsichtbare Tabulatorzeichen verwendet werden, ist es dennoch möglich, die Texte an einer nicht sichtbaren senkrechten Linie auszurichten. Es fällt auf, dass die Zeichen von Monospace Schriftarten alle in gleicher Breite gesetzt werden und dass Zeichen nicht leicht miteinander verwechselt werden können.

Im Toolbar des Editorfensters sind Eigenschaften von Zeichen und Text zu sehen, die das Rendering des Textes beeinflussen:

- Schriftart und Variante (Palatino, regular)
- Schriftgröße und Farbe von Schrift und Hintergrund (schwarz, keine)
- Absatzausrichtung (linksbündig)
- Zeilenabstand (1,3-fach)

Im `RTF`-Quelltext der Datei `fonts4.rtf` ist zu sehen, dass der sichtbare Text von Markierungen umgeben ist. Diese steuern die zur Ausgabe verwendeten Schriftarten (`f1`, `f2`, `f3`, `f4`; jeweils definiert in der zweiten Zeile unter `fonttbl`). Solche Markierungen werden Markup genannt. Weiterhin sind Steueranweisungen enthalten, die beispielsweise die Tabulatorenpositionen steuern (`tx`) oder Angaben zum Papierformat machen (`paperw1900`, `paperh16840`, `margl1440`, `margr1440`). Die Einzelheiten sind hier nicht wichtig. Es soll dargestellt werden, dass der sichtbare Text vom `RTF`-Editor aufgrund der Markierungen und Steuerangaben im `RTF`-Quelltext so dargestellt wird, wie er auf dem Bildschirm oder Papier aussehen soll. Bei HTML verhält es sich genauso (Das `M` in HTML steht für Markup).

Auszug aus der Datei `fonts4.rtf`

```
{\rtf1\ansi\ansicpg1252
{\fonttbl{\f0\froman\fcharset0 Palatino-Roman;
\f1\fswiss\fcharset0 Helvetica;\f2\froman\fcharset0 TimesNewRomanPSMT;
\f3\fnil\fcharset0 IBM Plex Mono;\f4\fnil\fcharset0 MonospaceXenon-Regular;}}
{\colortbl;\red255\green255\blue255;}
\paperw1900\paperh16840\margl1440\margr1440\vieww10480\viewh4860\viewkind0
\pard\tx566\tx1133\tx1700\tx2267\tx2834\tx3401\tx3968\tx4535\tx5102\tx5669\tx6236\tx6803\sl312\slmult1\pardir
t\

\f0\fs18 \cf0 \
Font: Palatino          0123456789   I1l00   floppy fl ff fi\
The quick brown fox jumps over the lazy dog\

\f2 Font: Times New Roman  0123456789   I1l00   floppy fl ff fi\
The quick brown fox jumps over the lazy dog

\f3 Font: IBM Plex Mono    0123456789   I1l00   floppy fl ff fi\
The quick brown fox jumps over the lazy dog\
```

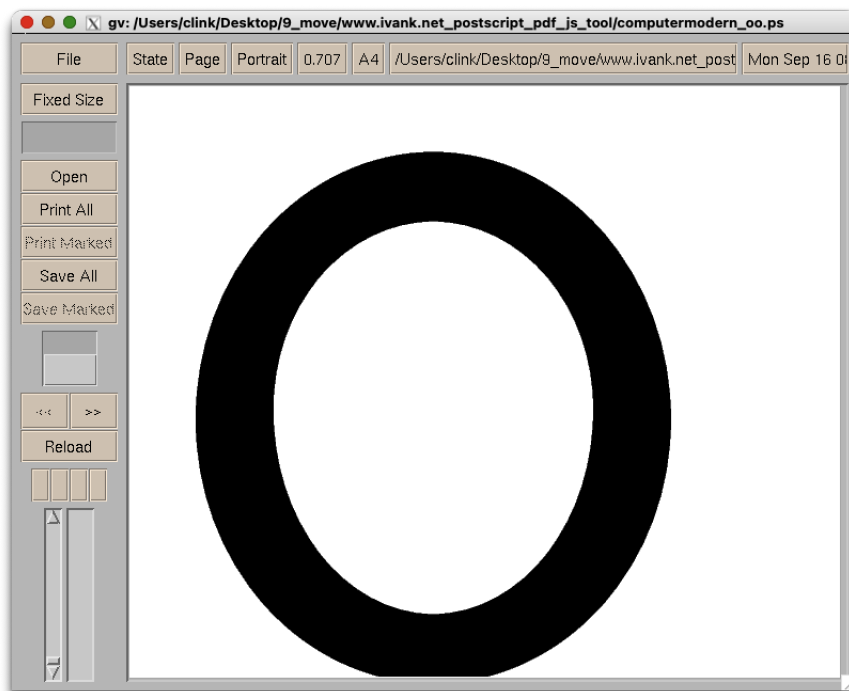


Abbildung 4: Der Glyph des Buchstaben „o“ in Ghostview (gv) gerendert.

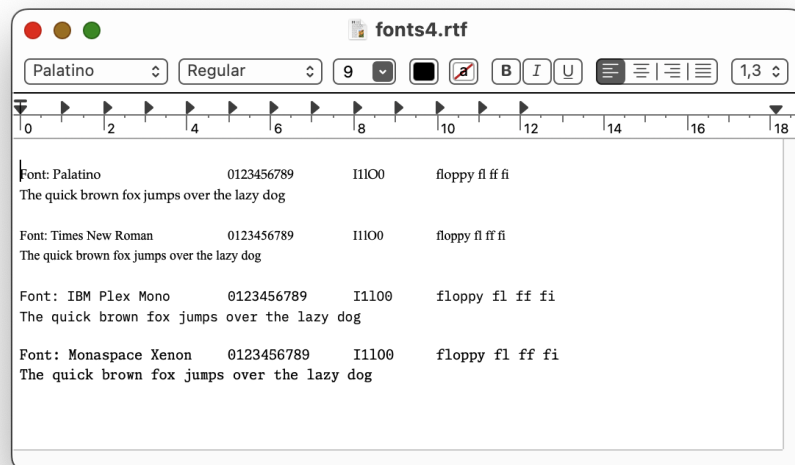


Abbildung 5: Vier verschiedene Schriftarten in einem einzigen Rich Text-Dokument.

```
\f4 Font: Monospace Xenon 0123456789 I1l00 floppy fl ff fi\
The quick brown fox jumps over the lazy dog\
}
```

2.7.2 Markdown

Der Begriff „Markdown“ ist ein Wortspiel auf „Markup“. Oft beschäftigt sich Markup mit typographischen Details, die vor allem dem Autor eines Textes beim Schreiben zunächst nicht wichtig sind. Markdown-Auszeichnungen sind daher eher strukturell oder geben Bedeutung. Beispiel: `\textit{kursiv}` (L^AT_EX) vs. `_kursiv_` (pandoc-Markdown).

2.7.3 WYSIWYG und WYSIWYM

Das Konzept „What you see is what you mean“ (WYSIWYM) liegt einerseits zwischen „What you see is what you get“ (WYSIWYG) Wortprozessoren und Markup bzw. Markdown-Auszeichnungssprachen: andererseits hat es zwei Auslegungen:

1. Semantische Auszeichnungen statt typographischer (beispielsweise `\code{int x=0;}` statt `\texttt{int x=0;}` – also „Code“ statt „typewriter font“ in L^AT_EX oder in HTML `<code>int x=0;</code>`)
2. Ansehnliche Darstellung im Editor für den Autor.

WYSIWYG ist bzgl. 2) zwar sehr gut; hat jedoch viele Unannehmlichkeiten für den Autor. Markup bzw. Markdown-Quelltexte sehen aus wie Kraut und Rüben, was das konzentrierte Arbeiten am Dokument erschwert. Hier ist das WYSIWYM-Konzept hilfreich. Es belastet den Autor nicht mit typographischen Details, umherspringenden Wörtern und Zeilen und verfügt über sehr gute Lesbarkeit (siehe Abbildung 6). Beispiele für WYSIWYM-Editoren sind a) LyX – a document processor that encourages an approach to writing based on the structure of your documents (WYSIWYM) and not simply their appearance (WYSIWYG)⁸ und b) TeXmacs – a free scientific editing platform⁹. Ein gut konfigurierter Editor mit gut eingestelltem Syntax Highlighting kommt WYSIWYM sehr nahe.

3 Material zum aktiven Lernen

3.1 Zusammenfassender Lückentext

1. Zur Programmierung oder Administration von Computern werden vielerlei Textdateien verwendet. Bei Textdateien wird zwischen, und unterschieden.
2. Textverarbeitungsprogramme verwenden (engl.: word prozessor) -Formate, bei denen der Texte manuell typographisch gestaltet werden können.

⁸<https://www.lyx.org/>

⁹<https://www.texmacs.org/>

3. Beispiele für plain text sind:
4. Beispiele für rich text sind:
5. Beispiele für Quelltext sind:
6. Beispiele für reine Anzeigeformate sind:
7. Beispiele für Anzeigeformate, die tatsächlich Textformate sind:
8. Rendering von Buchstaben 1) Bitmaps enthalten für jeden Buchstaben einer Schriftart
9. Rendering von Buchstaben 2) Für jeden Buchstaben einer Schriftart sind angegeben, mit denen der entsprechende Glyph werden kann (von: Grafikbibliothek des Betriebssystems, GhostView, Drucker, etc.).
10. Beim Rendering von Texten muss ein Plain Text-Editor beachten und den Text Hierbei ist es oft üblich, Teile des Textes automatisch
11. Textsatzsysteme (LaTeX, Markdown) erzeugen aus Quelltexten (.tex, .md) Dateien in Anzeigeformaten wie PDF oder HTML. Dabei werden die Dokumente anhand typographischer Regeln ansehnlich gestaltet. Es werden nicht nur einzelne Zeichen, sondern auch ganze Sätze, Abschnitte und Kapitel angeordnet (gerendert).
12. Die Formate HTML, PostScript, RTF werden beim Doppelklick (oder xdg-open) mit einem Betrachtungsprogramm ansehnlich angezeigt. Tatsächlich sind es jedoch, die mit einem Plain Text Editor bearbeitet werden können.

3.2 Vertiefungsaufgaben zu ‘Editieren von Texten’

- a) Konfigurieren Sie ihren GUI-Editor so, dass dieser flüssiges Arbeiten ermöglicht: Schriftart, Schriftgröße, Zeilennummern, Theme, Light Mode, Tabs/Spaces, etc.
- b) Bearbeiten Sie eine Textdatei in ihrem GUI-Editor und verwenden dabei Tastenkürzel zur Manipulation: Wörter und Textblöcke löschen, verschieben und verdoppeln.
- c) Machen Sie sich in ihrem Editor vertraut mit den Möglichkeiten bzgl. Copy & Paste, Search & Replace, Makros, Einrückung, Vervollständigung, Rechtschreibprüfung.
- d) Kopieren Sie eine Textdatei und verändern die Kopie. Lassen Sie die Differenzen zwischen den Texten anzeigen.

3.3 Vertiefungsaufgaben zu ‘Character Encodings’

- a) Wie viele Bytes werden mindestens gebraucht, um diese Textdatei (zwischen ---) zu speichern? Begründen Sie!

```
---  
Hello  
World  
---
```

- b) Was ändert sich bei vorheriger Aufgabe, wenn beide o durch ö ersetzt werden?
- c) Erstellen Sie mit `iconv` mehrere Varianten der Datei `hellöworld.utf8.txt` und öffnen diese in einem Texteditor. Schauen Sie, wie das Kommando `file *.txt` die Dateien einordnet.
- d) Stöbern Sie in der Datei `3rdParty/unifont-15.1.05.bmp`.

3.4 Vertiefungsaufgaben zu ‘Fonts’

- a) Finden Sie eine Schriftart, die Sie gerne in Zukunft für die Programmierung verwenden möchten.

3.5 Vertiefungsaufgaben zu ‘Rendering von Texten’

- a) Lassen Sie die Datei `test1.md.txt` von Pandoc in RTF und HTML umwandeln (mit `pandoc -s -o 1.rtf test1.md.txt` und `pandoc -o 2.html test1.md.txt`). Verändern Sie dann das Markup in `test1.md.txt` und zeigen die Unterschiede in der Ausgabe (auch im Quelltext). Hinweise: 1) kopieren Sie sich `1.rtf` und `2.html` bevor Sie Pandoc neu rendern lassen. 2) RTF-Dateien lassen sich mit Wordpad, TextEdit.app oder abword anzeigen. 3) Unterschiede zwischen Textdateien lassen sich mit `diff`, `meld` oder `diffuse` aufspüren.

3.6 Verständnisfragen zu ‘Editieren von Texten’

- a) Warum springt oft der Textcursor horizontal umher, wenn dieser vertikal mit den Pfeiltasten bewegt wird?

3.7 Verständnisfragen zu ‘Character Encodings’

- a) Welche Angabe (bzw. Annahme) wird benötigt, um eine Datei mit der Endung `.txt` korrekt zu interpretieren?
- b) Wie können Sie sicherstellen, dass ein menschlicher Empfänger einer Textdatei den Inhalt `111|00` korrekt interpretiert?
- c) Was ist mit dem Begriff „Binärdatei“ gemeint?

3.8 Verständnisfragen zu ‘Fonts’

- a) Über welche Eigenschaften sollte eine Schriftart verfügen, die für die Programmierung verwendet werden?
- b) Warum gibt es so viele verschiedene Schriftarten?
- c) Welche Schriftarten kommen in Frage, um für alphanumerische Freischaltcodes verwendet zu werden?
- d) Warum sind Bitmapschriftarten für moderne Bildschirme wenig geeignet?
- e) Warum sind Vektorschriftarten für kleine LC-Displays wenig geeignet?
- f) Warum ist es schwierig oder unmöglich, in `PostScript`- oder `PDF`-Dateien nach Wörtern zu suchen?
- g) Wie ist es möglich, dass moderne Smartphones Texte in Photos erkennen können?

3.9 Verständnisfragen zu ‘Rendering von Texten’

- a) Zwei Dateien, welche C++-Quellcode zeigen, sehen auf dem Bildschirm gleich aus. Eine heißt `sort.cpp` und eine `sort.ttf`. Worin besteht der Unterschied in der Art und Weise, wie die syntaktischen Hervorhebungen (syntax highlighting) realisiert sind?
- b) Diskutieren Sie den Unterschied zwischen den Markup-Anweisungen für Überschriften in `RTF` und `Markdown`.
- c) Warum ist es schwierig, fehlerbehaftet oder unmöglich, Texte in `PDF`, `Postscript` oder Bildern mit der Maus zu markieren und zu kopieren?

3.10 Testate zu ‘Unterschiede zwischen Texten’

- a) Finden Sie die Unterschiede zwischen den Dateien `test1.md.txt` und `test3.md.txt`.

3.11 Testate zu ‘Fonts’

- a) Laden Sie die Datei `pretzel2.ps` in einen Texteditor und in Ghostview (`gv`) oder auf `PDFi.js`¹⁰. Experimentieren Sie mit dem Quellcode. Verändern Sie Zahlen (beispielsweise vor `setlinewidth`) und Befehle (beispielsweise `curveto` ändern in `lineto`) und beobachten, was nach erneutem laden in Ghostview passiert.
- b) Die Datei `unifont-15.1.05-ASCII.png` enthält Pixelmuster für alle 128 ASCII-Zeichen. Die Zeichen sind alle gleich groß und in der Datei direkt nebeneinander angeordnet. Ermitteln Sie mit dem `file`-Kommando die Breite und Höhe des Gesamtbildes. Ermitteln Sie daraus die Breite und Höhe eines einzelnen Zeichens in diesem Bild.

¹⁰<https://www.ivank.net/veci/pdfi/>

- c) Installieren Sie GraphicsMagick (`sudo apt install graphicsmagick`). Berechnen Sie das nötige x-Offset (Abstand zum linken Rand des Bildes) anhand der Position des Zeichens 0 in der ASCII-Tabelle, um aus der Datei `unifont-15.1.05-ASCII.png` den Glyph für das Zeichen 0 zu ermitteln. Extrahieren Sie nun mit dem Kommando `gm convert unifont-15.1.05-ASCII.png -crop 16x16+XOFFSET+0 glyph0.png` den Glyph und betrachten die dabei entstandene Datei `glyph0.png` (ersetzen Sie `XOFFSET` durch den eben berechneten Wert).

4 Anhang: Literatur und weiterführendes Material

Bücher und Papers:

- Hunt, James W.; McIlroy, M. Douglas (June 1976). „An Algorithm for Differential File Comparison“. Computing Science Technical Report. 41. Bell Laboratories.
- TBD
- Matthew Butterick – Practical Typography¹¹
- ROBERT BRINGHURST – ELEMENTS OF TYPOGRAPHIC STYLE

nützliche URLs:

- The Tilde Text Editor¹²
- Programming Fonts – The most complete resource for the best monospace coding fonts.¹³
- CodingFont – a gamified experience to help you find your true love of coding fonts!¹⁴
- Atkinson Hyperlegible – greater legibility and readability for low vision readers¹⁵
- OPENDYSLEXIC – A TYPEFACE FOR DYSLEXIA¹⁶
- FIGlet – a program for making large letters out of ordinary text¹⁷
- banner – classic-style banner program similar to the one found in Solaris or AIX in the late 1990s. It prints a short string to the console in very large letters.¹⁸

Fundgrube:

¹¹<https://practicaltypography.com>

¹²<https://os.ghalkes.nl/tilde/index.html>

¹³<https://www.programmingfonts.org/>

¹⁴<https://www.codingfont.com>

¹⁵<https://brailleinstitute.org/freefont>

¹⁶<https://opendyslexic.org/>

¹⁷<http://www.figlet.org/>

¹⁸<https://github.com/pronovic/banner/>

- Difftastic – a structural diff tool that compares files based on their syntax.¹⁹
- xkcd 1015 – Kerning²⁰
- Sans Bullshit Sans – Leveraging the synergy of ligatures²¹

¹⁹<https://github.com/Wilfred/difftastic/>

²⁰<https://www.explainxkcd.com/wiki/index.php/1015>

²¹<https://www.sansbullshitsans.com>

5.1.6 Operators with Limits Math ! Sums Math ! Integral... subsec:Operators-with-Limits

Sum (\sum) and integral (\int) operators are very often decorated with limits. These limits can be entered in LyX by entering them as you would enter a super- or subscript, directly after the symbol. The sum operator will automatically place its “limits” over and under the symbol in displayed formulas, and to the side in inline formulas, as in $\sum_{n=0}^{\infty} \frac{1}{n!} = e$, versus

$$\sum_{n=0}^{\infty} \frac{1}{n!} = e$$

5.1.6. Operators with Limits

Sum (\sum) and integral (\int) operators are very often decorated with limits. These limits can be entered in LyX by entering them as you would enter a super- or subscript, directly after the symbol. The sum operator will automatically place its “limits” over and under the symbol in displayed formulas, and to the side in inline formulas, as in $\sum_{n=0}^{\infty} \frac{1}{n!} = e$, versus

$$\sum_{n=0}^{\infty} \frac{1}{n!} = e$$

Abbildung 6: WYSIWYM-Editor Lyx: Oben Ansicht des Autors; unten das gerenderte Endprodukt.