Programmierkonzepte in den Naturwissenschaften

16. Formatierung: Layout und Stil

PD Dr. Till Biskup
Physikalische Chemie und Didaktik der Chemie
Universität des Saarlandes
Sommersemester 2020

Zentrale Aspekte

- Codeformatierung ist zu wichtig, um sie zu ignorieren – oder sie dogmatisch zu behandeln.
- Formatierung offenbart die Sorgfalt und Professionalität, die ein Programmierer in seine Arbeit investiert hat.
- Formatierung erhöht die Les- und Wartbarkeit von Code. Die investierte Disziplin überlebt den eigentlichen Inhalt.
- Zusammenhänge und getrennte Konzepte sollten sich in horizontaler und vertikaler Formatierung widerspiegeln.
- Konsistenz ist wichtiger als der konkrete Inhalt. Regeln sollten vom *ganzen* Team akzeptiert werden.

Übersicht

Warum ist Code-Formatierung wichtig?

Vertikale Formatierung

Horizontale Formatierung

Konsistenz: Konventionen und automatische Codeformatierung

Warum ist Code-Formatierung wichtig?

Eine Reihe guter Gründe

> *Code formatting is about communication,*
> *and communication is the professional developer's*
> *first order of business.*
>
> – Robert C. Martin

- Formatierung offenbart die Sorgfalt und Professionalität.
  - Qualität und Formatierung sind selten wirklich trennbar.
- Formatierung erhöht die Lesbarkeit und Wartbarkeit.
  - Lesbarkeit ist eine Grundvoraussetzung für Wartbarkeit – und Wiederverwertbar-, Zuverlässig- und Überprüfbarkeit.
  - Formatierung hat direkte Auswirkung auf die *Wissenschaftlichkeit*.
- Die eingesetzte Disziplin überlebt den eigentlichen Inhalt.
  - Gute Formatierung hat prägenden Vorbildcharakter.

Robert C. Martin: Clean Code, Prentice Hall, Upper Saddle River, NJ, 2008, S. 76

Gute Formatierung schafft bleibende Werte

Von der Wirksamkeit guter Vorbilder

> *...the readability of your code will have a profound effect*
> *on all the changes that will ever be made.*
> *The coding style and readability set precedents*
> *that continue to affect maintainability and extensibility*
> *long after the original code has been changed beyond recognition.*
> *Your style and discipline survives,*
> *even though your codes does not.*
>
> – Robert C. Martin

- Guter Stil hat die Chance, bleibende Werte zu schaffen.
- Voraussetzung:
  Er muss Lesbarkeit und Wartbarkeit gewährleisten.

Robert C. Martin: Clean Code, Prentice Hall, Upper Saddle River, NJ, 2008, S. 76

Wie Code *nicht* formatiert sein sollte

Ein besonders attraktives Beispiel unleserlichen Codes

```
#!/usr/bin/perl -w                                          # camel code
use strict;

                                          $_='ev
                                        al("seek\040D
          ATA,0,                      0;");foreach(1..3)
        {<DATA>;}my                   @camel1hump;my$camel;
  my$Camel  ;while(                  <DATA>){$_=sprintf("%-6
9s",$_);my@dromedary                1=split(//);if(defined($
_=<DATA>)){@camel1hum              p=split(//);}while(@dromeda
 ry1){my$camel1hump=0             ;my$CAMEL=3;if(defined($_=shif
       t(@dromedary1            ))&&/\S/){$camel1hump+=1<<$CAMEL;}
      $CAMEL--;if(d           efined($_=shift(@dromedary1))&&/\S/){
     $camel1hump+=1          <<$CAMEL;}$CAMEL--;if(defined($_=shift(
    @camel1hump))&&/\S/){$camel1hump+=1<<$CAMEL;}$CAMEL--;if(
    defined($_=shift(@camel1hump))&&/\S/){$camel1hump+=1<<$CAME
    L;;}$camel.=(split(//,"\040..m`{/J\047\134}L^7FX"))[$camel1h
     ump];}$camel.="\n";}@camel1hump=split(/\n/,$camel);foreach(@
     camel1hump){chomp;$Camel=$_;y/LJF7\173\175`\047/\061\062\063\
     064\065\066\067\070/;y/12345678/JL7F\175\173\047`/;$_=reverse;
      print"$_\040$Camel\n";}foreach(@camel1hump){chomp;$Camel=$_;y
       /LJF7\173\175`\047/12345678/;y/12345678/JL7F\175\173\0 47`/;
        $_=reverse;print"\040$_$Camel\n";}';;s/\s*//g;;eval;   eval
          ("seek\040DATA,0,0;");undef$/;$_=<DATA>;s/\s*//g;(   );;s
            ;^.*_;;;map{eval"print\"$_\"";}/.{4}/g; __DATA__    \124
              \1   50\145\040\165\163\145\040\157\1 46\040\1  41\0
                   40\143\141  \155\145\1 54\040\1   51\155\  141
                   \147\145\0  40\151\156 \040\141    \163\16 3\
                    157\143\   151\141\16  4\151\1     57\156
                    \040\167  \151\164\1   50\040\      120\1
                    45\162\   154\040\15    1\163\      040\14
                    1\040\1   64\162\1       41\144       \145\
                    155\14    1\162\        153\04        0\157
                     \146\     040\11      7\047\         122\1
                     45\15      1\154\1  54\171           \040
                     \046\         012\101\16             3\16
                     3\15           7\143\15              1\14
                     1\16            4\145\163            \054
                    \040             \111\156\14          3\056
                   \040\           125\163\145\14         4\040\
                   167\1           51\164\1  50\0         40\160\
                 145\162                                 \155\151
               \163\163                                   \151\1
             57\156\056
```

https://www.perlmonks.org/?node_id=45213 **(Zugriff am 11.06.2020)**

Übersicht

Aspekte vertikaler Code-Formatierung

Eine Übersicht

- Länge von Dateien
  - Dateien sind i.d.R. länger als Funktionen.
  - Zu lange Dateien werden unübersichtlich.
- die Zeitungs-Metapher
  - von einfach nach komplex, abstrakt nach konkret
- vertikale Offenheit und Dichte
  - Leerraum (Leerzeilen) zwischen Sinnabschnitten
  - kein Leerraum innerhalb von Blöcken
- vertikaler Abstand
  - Konzeptionell aufeinander bezogene Codeteile sollten vertikal nahe beieinander stehen.

Länge von Dateien

Abhängig vom Inhalt einer Datei – und dem Programmierstil

- Kürze ist kein Selbstzweck.
  - Die Lesbarkeit sollte nicht leiden.
  - Explizite Formulierung über mehrere Zeilen ist oft lesbarer als „clevere“ Einzeiler.
- Die Länge ist abhängig vom Inhalt einer Datei.
  - Länge von Funktionen und Dateien oft unabhängig
  - häufig eine Klasse in einer Datei
- Kurze Dateien haben Vorteile.
  - übersichtlicher
  - einfacher wiederverwendbar
  - zu lange Dateien meist Hinweis auf zu große Einheiten (wie Klassen etc.)

☛ Gute Editoren erleichtern das Navigieren im Code.

Die Zeitungs-Metapher

Von einfach nach komplex, abstrakt nach konkret

- Aspekte eines Zeitungsartikels
  - Überschrift
  - Zusammenfassung
  - eher kurz gefasst und fokussiert
- Abbildung auf Code
  - Überschrift: Funktions- oder Klassenname
  - Zusammenfassung: oberste Abstraktionsebene
  - Abfolge aufeinander aufbauender kleiner Funktionen
- Zielstellung
  - wichtige Informationen möglichst weit oben
  - Leser/Nutzer soll schnell wissen, ob er „richtig“ ist

Die Zeitungs-Metapher

Von einfach nach komplex, abstrakt nach konkret

Listing 1: Quellcode im Stil der Zeitungs-Metapher

```
def analyse_my_data(source):
    data = read_data(filename=source)
    data = preprocess_data(data)
    result = analyse_data(data)
    present_result_of_analysis(result)
    save_result_of_analysis(result)

def read_data(filename):
    data['data'] = import_data(filename)
    data['header'] = get_header()
    data['metadata'] = import_metadata()
    return data

# ...

def import_data(filename):
    with open(filename) as file:
        content = file.read()
    raw_data = np.genfromtxt(io.BytesIO(content.replace(
        ',', '.').encode()), skip_header=2)
    return raw_data[:, 1]

# ...
```

Vertikale Offenheit und Dichte

Vertikale Abstände gliedern Sinnabschnitte.

- Grundsätzlicher Aufbau von Quellcode
  - Leserichtung von links nach rechts und oben nach unten
  - eine Zeile für jeden Ausdruck
  - eine Gruppe zusammenhängender Zeilen pro Gedanke
- vertikale Offenheit (Leerzeilen)
  - Jeder Gedanke ist durch eine Leerzeile getrennt.
  - großer Einfluss auf Übersichtlichkeit und Lesbarkeit
- vertikale Dichte (keine Leerzeilen)
  - Die Teile (Zeilen) eines Konzepts gehören zusammen.
  - Unnötige Kommentare zerstören diesen Zusammenhang.

☛ Simple Regel mit großer Auswirkung auf die Lesbarkeit.

☛ universell einsetzbar – nicht nur bei Quellcode

Vertikale Offenheit und Dichte

Ein reales Beispiel zur Verdeutlichung

Listing 2: Unlesbarer Code: fehlende vertikale Formatierung

```
tic; t=0:0.01:2*pi; x=4*cos(3*t); y=4*cos(4*t+pi/2);
figure;hold on; for k=1:length(x) plot(x(k),y(k),'r.');pause(0.001);end toc
```

Listing 3: Lesbarer Code: saubere vertikale Formatierung

```
tic;

t = 0:0.01:2*pi;
x = 4 * cos(3*t);
y = 4 * cos(4*t+pi/2);

figure;
hold on;
for k = 1:length(x)
    plot(x(k),y(k),'r.');
    pause(0.001);
end

toc
```

Listing 4: LaTeX-Quellcode mit übersichtlicher vertikaler Formatierung

```
\section{Wie alles begann}
\label{sec:beginn}

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin semper,
purus eu feugiat ullamcorper, lorem nulla pellentesque ligula, pretium
aliquam ex nulla eget diam. Ut quis quam nunc.

\begin{itemize}
\item Integer eu leo quis est rutrum convallis a in nibh.

\item Nulla quis magna vitae lectus pharetra consequat.
\end{itemize}

Nunc pretium, lectus sit amet mollis tempus, diam magna cursus orci,
vel rhoncus nisi purus sit amet turpis.


\subsection{Als ich noch klein und unwissend war}

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per
inceptos himenaeos. Curabitur pharetra vitae enim sed lobortis. Morbi
tempus scelerisque nulla, vitae hendrerit massa maximus sit amet.
```

Vertikaler Abstand

Aufeinander bezogene Codeteile so nah beieinander wie möglich

- Variablendeklarationen
  - so nah wie möglich am Code, der sie verwendet
  - kurze Funktionen, deshalb Variablen an deren Beginn
- Instanzvariablen (von Klassen/Objekten)
  - an einem zentralen Ort
  - meist ganz am Anfang
- abhängige Funktionen
  - aufgerufene nahe bei den aufrufenden Funktionen
  - aufgerufene unterhalb der aufrufenden Funktionen
- konzeptionelle Affinität
  - Bezug konzeptionell, nicht notwendig durch Aufrufe
  - Beispiel: Funktionen mit ähnlichen Aufgaben

Übersicht

Aspekte horizontaler Code-Formatierung

Eine Übersicht

- Länge von Zeilen
  - Zu lange Zeilen werden unübersichtlich.
- horizontale Offenheit und Dichte
  - Horizontale Abstände verdeutlichen Zusammenhänge.
- horizontale Ausrichtungen
  - meist inkompatibel zu automatischer Formatierung
- Einrückung
  - entscheidend für die Erfassung von Zusammenhängen
  - erhöhen die Lesbarkeit sehr stark
- leere Schleifenkörper
  - immer mit eigener Zeile explizit machen

Länge von Zeilen

Zu lange Zeilen werden unübersichtlich.

- Harte Limits sind (heute) schwer zu rechtfertigen.
  - Moderne Programmiersprachen haben keine Limitierungen.
  - Moderne Monitore sind viel größer als früher.
  - Der Textsatz seit Gutenberg gibt Hinweise...
- allgemeine Aspekte
  - Horizontales Scrollen sollte immer vermieden werden.
  - Lange Zeilen werden unübersichtlich/schwer lesbar.
  - 120 Zeichen ist ein sinnvolles oberes Limit.
  - Druckbarkeit erfordert deutlich kürzere Zeilen.
- Hilfsmittel und Strategien
  - gewähltes Längenlimit im Editor anzeigen lassen
  - lange Zeilen umbrechen (abhängig von der Sprache)
  - lange Statements auf mehrere Zeilen aufteilen

Horizontale Offenheit und Dichte

Horizontale Abstände verdeutlichen Zusammenhänge.

- Generelles Prinzip
  - Zusammengehöriges ohne Zwischenraum schreiben, nicht Zusammenhängendes durch Leerzeichen trennen
- häufige Anwendungsfälle
  - Zuweisungsoperator von Leerzeichen umgeben
  - keine Leerzeichen zwischen Funktionsname und Klammern
- Beispiel: Verdeutlichung der Operatorrangfolge
  - kann die intuitive Erfassbarkeit von Quellcode erhöhen
  - fällt häufig der automatischen Formatierung zum Opfer
  - Formatierung ist keine Garantie für die reale Rangfolge.
  - im Zweifelsfall immer Klammern setzen

Listing 5: Horizontale Offenheit und Dichte in Python (PEP 8)

```
def _change_keys_in_dict_recursively(self, dict_=None):
    tmp_dict = collections.OrderedDict()
```

Einrückung

Entscheidend für die Erfassung von Zusammenhängen

- Einrückung verdeutlicht die Hierarchieebenen.
  - Hierarchieebenen stellen den jeweiligen Kontext dar.
  - Kontext ist essentiell für die Lesbarkeit und das Verständnis von Code.
  - Ohne Einrückung ist Verständnis fast unmöglich.
- Einrückung sollte nie aufgebrochen werden.
  - Kurze Schleifen oder Bedingungen passen auf eine Zeile.
  - Die Lesbarkeit geht aber weitestgehend verloren.
- Die Art der Einrückung ist oft festgelegt.
  - Möglichkeiten: Leerzeichen oder Tabulator
- Einrückung kann *nicht optional* sein.
  - Python verzichtet auf Klammern und nutzt Einrückung.

Leere Schleifenkörper

Möglichst vermeiden – sonst explizit kenntlich machen

- Leere Schleifenkörper sind schwer zu lesen.
  - sollten nach Möglichkeit vermieden werden
  - wenn unvermeidbar, dann explizit kenntlich machen

Listing 6: Leerer Schleifenkörper in C

```
while ((c = getchar()) == ' ' || c == '\n' || c == '\t')
    ;
```

- Voraussetzung für leere Schleifenkörper
  - Zuweisung in Bedingung einer Schleife möglich
- Funktion des Listings
  - entfernt alle Leerzeichen, Zeilenumbrüche, Tabulatoren

Übersicht

Ein paar Anmerkungen zu Konsistenz

Konsistenz ist zu wichtig, um sie dogmatisch zu behandeln.

> *A foolish consistency is the hobgoblin of little minds*
>
> – Ralph Waldo Emerson

> *It is an old observation that the best writers*
> *sometimes disregard the rules of rhetoric.*
> *When they do so, however, the reader will usually find*
> *in the sentence some compensating merit,*
> *attained at the cost of the violation.*
> *Unless he is certain of doing as well,*
> *he will probably do best to follow the rules.*
>
> – William Strunk

`https://en.wikisource.org/wiki/Essays:_First_Series/Self-Reliance` **(Zugriff am 11.06.2020)**

In: Strunk und White, The Elements of Style, Fourth Ed., Longman, 2000, S. xvii f.

Ein paar Anmerkungen zu Konsistenz

Konsistenz ist zu wichtig, um sie dogmatisch zu behandeln.

> *A style guide is about consistency.*
> *Consistency with this style guide is important.*
> *Consistency within a project is more important.*
> *Consistency within one module or function is the most important.*
>
> *However, know when to be inconsistent –*
> *sometimes style guide recommendations just aren't applicable.*
> *When in doubt, use your best judgment.*
> *Look at other examples and decide what looks best.*
>
> – PEP 8

- ☛ Souveräner Umgang mit Konventionen
  erfordert Bewusstsein und informierte Entscheidungen im Einzelfall.

`https://www.python.org/dev/peps/pep-0008/` **(Zugriff am 08.06.2020)**

Konventionen sind inhaltlich verhandelbar

Wichtig ist ihre konsistente Anwendung und Befolgung.

- Konventionen dienen der Lesbarkeit.
  - Alles, was wesentlich zur Lesbarkeit beiträgt, ist gut.
  - Einzelne Aspekte sind oft eine Geschmacksfrage.
- Vereinbarung aller Programmierer eines Projekts
  - Alle initial Beteiligten sollten an Konventionen mitwirken.
  - Zentrale Konventionen für eine Programmiersprache sollten nur mit gutem Grund gebrochen werden.
- Alle sollten die Konventionen mittragen können.
  - Einzelpersonen werden sich selten komplett durchsetzen.
  - Jeder Projektbeteiligte folgt den gleichen Konventionen.

☛ Alle zu beteiligen schafft die notwendige Akzeptanz, die (selbstgegebenen) Konventionen auch zu befolgen.

Sprachliche Idiome

Etablierte Muster mit hohem Wiedererkennungseffekt

Idiom

Linguistik: Spracheigentümlichkeit;
Softwaretechnik: Umsetzung (Implementierung) abstrakter Muster bzw. Lösung einfacher Aufgaben (auf niedrigster Abstraktionsstufe) in einer konkreten Programmiersprache

- sprachliche Idiome für häufige Konstrukte
  - eine anerkannte Variante, ein Problem zu lösen
  - Muster auf fundamentalster Ebene einer Sprache
- Beispiel aus der Linguistik
  - „Es war einmal . . . “ – „Once upon a time . . . “ – „C’era una volta . . . “ – „Il était une fois . . . “

Sprachliche Idiome

Programmiersprachen haben Idiome für häufige Konstrukte.

Listing 7: for-Schleife über alle Elemente in C/C++

```
for (i = 0; i < n; i++)
    array[i] = 1.0;
```

- Verwendung ist *nicht optional*
  - Verwendung ist ein Zeichen der Sprachbeherrschung.
  - Sprachbeherrschung ist eine wesentliche Voraussetzung für qualitativ hochwertigen Code.
  - Idiome sind allgemein akzeptierte Konventionen.
- Gründe für Idiome
  - erhöhen die Wiedererkennbarkeit
  - sorgen für korrektes Verhalten und weniger Fehler
  - sind unabhängig vom konkreten Team

Automatische Codeformatierung

Automatisierung sorgt für Konsistenz.

- Grundsatz: automatisieren, was sich automatisieren lässt
  - Formatierung ist zu wichtig, um sie zu vernachlässigen.
  - Automatische Formatierung nimmt viel Arbeit ab
    und sorgt für konsistente Ergebnisse.
- Gute Editoren unterstützen nutzerspezifische Regeln.
  - Nutzerspezifische Regeln sind essentiell,
    da nicht die Werkzeuge Konventionen festlegen sollten.
  - Ggf. übernehmen externe Werkzeuge diesen Part.
- Konsistenz ist wichtiger als Konventionen.
  - Konventionen, die sich nicht automatisch umsetzen lassen,
    sind (meist) schlechte Konventionen.
  - Alles, was sich nicht automatisieren/formalisieren lässt,
    ist nur schwer konsistent zu halten.

Zentrale Aspekte

- Codeformatierung ist zu wichtig, um sie zu ignorieren – oder sie dogmatisch zu behandeln.
- Formatierung offenbart die Sorgfalt und Professionalität, die ein Programmierer in seine Arbeit investiert hat.
- Formatierung erhöht die Les- und Wartbarkeit von Code. Die investierte Disziplin überlebt den eigentlichen Inhalt.
- Zusammenhänge und getrennte Konzepte sollten sich in horizontaler und vertikaler Formatierung widerspiegeln.
- Konsistenz ist wichtiger als der konkrete Inhalt. Regeln sollten vom *ganzen* Team akzeptiert werden.