Programmierkonzepte in der Physikalischen Chemie

16. Formatierung

Albert-Ludwigs-Universität Freiburg

Dr. Till Biskup

Institut für Physikalische Chemie Albert-Ludwigs-Universität Freiburg Wintersemester 2017/18

- Codeformatierung ist zu wichtig, um sie zu ignorieren oder darum zu eifern.
- 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

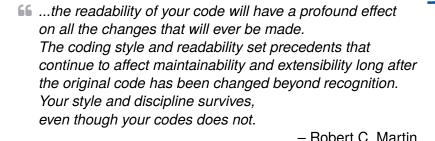
Horizontale Formatierung

Konsistenz: Konventionen und automatische Codeformatierung

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.
- Die eingesetzte Disziplin überlebt den eigentlichen Inhalt.
 - Gute Formatierung hat prägenden Vorbildcharakter.



- 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

- ▶ 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
 - Abstände zwischen Sinnabschnitten
 - keine Abstände innerhalb von Blöcken
- vertikaler Abstand
 - Konzeptionell aufeinander bezogene Codeteile sollten vertikal nahe beieinander stehen.

- 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.)

- 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



- 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

Ein reales Beispiel zur Verdeutlichung

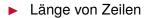
Listing 1: Unlesbarer Code: fehlende vertikale Formatierung

```
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 2: 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
```

- 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



- 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

- ► 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 aufbrechen



- ▶ 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

- ► 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 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 3: 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



- 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.



Idiom

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

- sprachliche Idiome f
 ür h
 äufige 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..."



Listing 4: for-Schleife über alle Elemente in C/C++

for (i = 0; i < n; i++)
 array[i] = 1.0;</pre>

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



- ▶ 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 schlechte Konventionen.
 - Alles, was sich nicht automatisieren/formalisieren lässt, ist nur schwer konsistent zu halten.

- Codeformatierung ist zu wichtig, um sie zu ignorieren oder darum zu eifern.
- 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.