Ruby-Programmierung

de.wikibooks.org

21. November 2015

Inhaltsverzeichnis

1 Einleitung **3**
- 1.1 Ruby? . . . 3
- 1.2 Warum Ruby? . . . 3
- 1.3 Aufbau und nötiges Vorwissen . . . 3

2 Installation **5**
- 2.1 MRI . . . 5
- 2.2 Editor . . . 8

3 Hallo Welt! **9**
- 3.1 Hallo Welt . . . 9
- 3.2 Kommentare . . . 10
- 3.3 Shebang . . . 10

4 Rechnen **13**
- 4.1 Variablen . . . 13
- 4.2 Grundrechenarten . . . 14
- 4.3 Flächeninhalt eines Kreises . . . 14

5 Kontrollstrukturen **17**
- 5.1 Verzweigungen . . . 17
- 5.2 Schleifen . . . 18
- 5.3 Fakultät . . . 19
- 5.4 ? : . . . 20

6 Methoden **21**
- 6.1 Methoden . . . 21
- 6.2 Parameter . . . 21
- 6.3 Rückgabewerte . . . 22
- 6.4 Mehrere Quelltextdateien . . . 22
- 6.5 Rekursion und Sichtbarkeit von Variablen . . . 23
- 6.6 Rückblick: Fakultät . . . 24

7 Strings **25**
- 7.1 Anführungszeichen . . . 25
- 7.2 Einfügen von Variablen in Strings . . . 26
- 7.3 Teilstrings . . . 27
- 7.4 Symbole . . . 27

8 Zahlen **29**
8.1 Integer . . . 29
8.2 Floating Point Numbers . . . 30
8.3 BigDecimal . . . 31

9 Arrays **33**
9.1 Erzeugen und Zugriff . . . 33
9.2 Arrays verändern . . . 34
9.3 Mehrere Rückgabewerte einer Methode . . . 34
9.4 Kassenbon . . . 34

10 Hashes **37**
10.1 Erzeugen und Zugriff . . . 37
10.2 Rückblick: Kassenbon . . . 37
10.3 Sets . . . 38

11 Dateien **39**
11.1 Ausgabe des eigenen Quelltextes . . . 39
11.2 Rechte . . . 39
11.3 Auflistung eines Verzeichnis . . . 40
11.4 Verzeichnisse und Dateien . . . 40
11.5 DATA . . . 40

12 Regular Expressions **43**
12.1 Initialisierung und einfache Beispiele . . . 43
12.2 Zeichenklassen . . . 43
12.3 Sonderzeichen . . . 44
12.4 Die Variablen $1, $2, ... 44
12.5 Named Groups . . . 45
12.6 Längenkonverter . . . 45

13 Kommandozeilenintegration **47**
13.1 Kommandozeilenparameter . . . 47
13.2 Zur Kommandozeile zurück . . . 47
13.3 Fehlerbehandlung . . . 48

14 Klassen **51**
14.1 Motivation . . . 51
14.2 Klassen . . . 51
14.3 Instanzmethoden und -variablen . . . 52
14.4 Klassenmethoden und -variablen . . . 53

15 Module **55**
15.1 Module . . . 55
15.2 Namensraumoperator . . . 56

16 Vererbung **57**
16.1 Vererbung . . . 57

17 Rückblick: Grundlagen **59**
17.1 Alles, wirklich Alles . . . 59
17.2 Nocheinmal: Fakultät . . . 59

18 Funktionale Aspekte **61**
18.1 Blöcke . . . 61
18.2 Blöcke in eigenen Methoden . . . 62
18.3 Iteratoren . . . 62
18.4 Objektinstanzen von ausführbarem Code . . . 63

19 Threads **65**
19.1 Threads . . . 65
19.2 Prozesse unter Unix mit fork . . . 66

20 Exceptions **67**
20.1 Exceptions . . . 67
20.2 StandardError . . . 68
20.3 Rückblick: Threads . . . 69
20.4 Catch und Throw . . . 69

21 Debugging **71**
21.1 debug . . . 71
21.2 pry-debugger . . . 72

22 Testing **73**
22.1 Test Driven Development . . . 73
22.2 Notizbuch . . . 73

23 Netzwerkprogrammierung **75**
23.1 Sockets und Protokolle und 75
23.2 Chat . . . 76
23.3 HTTP . . . 78

24 Rubyerweiterungen mit C **79**
24.1 Beispielprogramm . . . 79
24.2 RubyInline . . . 80
24.3 C Erweiterung . . . 81
24.4 Zusammenfassung . . . 82

25 Rubygems **83**
25.1 Rubygems und Bundler . . . 83
25.2 Entwicklung eines Rubygems . . . 83
25.3 Gemfile . . . 85

26 Rake **87**
26.1 Syntax . . . 87

27 Aufgaben **89**
27.1 Passwortgenerator . . . 89

27.2 Primzahlprüfer 91
27.3 Konvertieren in das metrische System 91

28 Ich brauche Hilfe! 95
28.1 Ruby-Doc.org 95
28.2 Integrierte Dokumentation 95
28.3 Interactive Ruby Shell 97

29 Autoren 99

Abbildungsverzeichnis 101

30 Licenses 105
30.1 GNU GENERAL PUBLIC LICENSE 105
30.2 GNU Free Documentation License 106
30.3 GNU Lesser General Public License 107

1 Einleitung

1.1 Ruby?

Ruby ist eine Programmiersprache, die von Yukihiro "Matz" Matsumoto[1] entworfen wurde, der sie nach dem Rubin benannte. Vermutlich ist der Name auch als Anspielung auf Perl zu verstehen. Aufgrund des anfänglichen Mangels an englischsprachiger Dokumentation verbreitete sich die Sprache vor allem im japanischen Sprachraum, wo sie seither beliebter ist als ihr westliches Pendant Python. Erst mit dem Aufkommen des Webframeworks Ruby on Rails[2] und dem Bedarf an schneller Webentwicklung fand Ruby auch in der westlichen Welt Verbreitung.

1.2 Warum Ruby?

Bei der Entwicklung von Ruby stand die Einheitlichkeit und Lesbarkeit des Quelltextes im Vordergrund. Das betrifft insbesondere Konventionen bei der Bennenung von Variablen und Methoden sowie eine sehr einheitliche und konsistente Syntax. In Ruby werden die Vorteile von objektorientierter und funktionaler Programmierung verbunden; das ermöglicht, viele Probleme sehr einfach zu beschreiben und zu lösen.

Es existiert eine große Menge verfügbarer Bibliotheken, die in Verbindung mit RubyGems[3] durch einfache Verfügbarkeit es einem ermöglichen, verbreitete Probleme nicht selbst erneut lösen zu müssen. Dadurch können sehr viele Anwendungen vom Konsolenskript über grafische Anwendungen bis hin zu Netzwerk- und Internetapplikationen in Ruby entwickelt werden. An einigen Stellen des Buches wird auch auf Probleme bei der Verwendung von Ruby eingegangen, so kann es sich insbesondere als zu langsam herausstellen. Das sollte einen nicht davon abschrecken, sich Ruby anzuschauen oder mit Ruby programmieren zu lernen. Insbesondere soziale Konventionen in der Rubyentwicklergemeinde erlauben es sehr schnell, auch guten Code zu schreiben, und stellen einige Konzepte vor, die vielleicht selbst für erfahrene Programmierer neu sind.

1.3 Aufbau und nötiges Vorwissen

Die ersten beiden Teile des Buches führen in die Programmierung mit Ruby ein, die Probleme sind niederschwellig und es werden keine Voraussetzungen an Ihre Programmierkennt-

1 `https://de.wikipedia.org/wiki/Yukihiro%20Matsumoto`
2 `https://de.wikibooks.org/wiki/Ruby%20on%20Rails`
3 `https://de.wikipedia.org/wiki/RubyGems`

nisse gesetzt. Es findet jedoch keine Einführung in die allgemeine Programmierung statt, so werden algorithmisches und mathematisches Denken vorausgesetzt. Dabei müssen Sie keine Angst haben, denn die Probleme werden praxisnah erklärt und sind mit Codebeispielen illustriert. Beim Vertiefen des Stoffes hilft eine Aufgabensammlung am Ende des Buches.

Im weiterführenden Teil werden aufbauend auf die ersten beiden Teile fortgeschrittene Probleme und Lösungen präsentiert. Anfängern soll dieser Teil helfen, einen Übergang von den Grundlagen zu eigenen oder der Mitarbeit an anderen Projekten zu finden. Erfahrene Programmierer können diesen Teil als Einstieg in die Rubywelt nutzen, da hier interessante Konzepte wie *Test Driven Development* vorgestellt werden.

Wird auf Quelltext im Fließtext eingegangen, so ist er in `nichtproportionaler Schrift` gesetzt. Kommentare in längeren Quelltexten werden vor die Bezugszeile gesetzt, außer es wird auf die Ausgabe eines Befehls verwiesen. Spielt der Dateiname eines Skripts eine Rolle, so stellte er einen der ersten Kommentare da; sonst wird er weggelassen.

2 Installation

Für Ruby existieren zahlreiche Implementierungen, die ihre Vor- und Nachteile haben. Neben der sehr ausführlichen Liste in der Wikipedia[1] seien hier nur die wichtigsten erwähnt. Der MRI, Matz's Ruby Interpreter, wird in diesem Buch genutzt und gilt als Referenz für andere Implementierungen. Die aktuelle Version ist 2.2.0. Der Rubyinterpreter ist ein freies Programm und kann kostenlos auf der offiziellen Seite[2] heruntergeladen werden. Daneben sollten an dieser Stelle vor allem JRuby und MacRuby genannt werden. JRuby erzeugt aus Rubycode Java-Bytecode, der sich auf der JVM ausführen lässt, dadurch lassen sich alle Java-Bibliotheken nutzen. MacRuby nutzt die LLVM und hat den Ruf, die schnellste der verschiedenen Implementierungen zu sein.

2.1 MRI

2.1.1 Installieren unter Windows

Unter Windows gibt es drei Möglichkeiten, den Interpreter zu installieren. Zunächst könnte man ihn selbst aus den Quelltexten mittels eines C-Compilers kompilieren; dann wäre es noch möglich, ihn in einer emulierten Linux-Umgebung wie Cygwin zu installieren. Am einfachsten ist es aber sicherlich, den Interpreter mittels des Windowsinstallers zu installieren. Dieses Verfahren wird nicht nur den meisten Windowsanwendern geläufig sein, zusätzlich werden auch noch eine Reihe oft verwendeter Bibliotheken sowie ein Editor installiert. Der Installer kann von rubyforge[3] heruntergeladen werden.

2.1.2 Installieren unter Linux/UNIX

Auch unter den meisten anderen Betriebsystemen ist es in der Regel nicht nötig, den Interpreter zu kompilieren. Bei vielen Linuxdistributionen ist er sogar bereits dabei, dies kann man einfach mittels des folgenden Befehls herausfinden:

```
$ ruby --version
```

Wenn Ruby gefunden wurde, liefert es die installierte Version zurück. Konnte Ruby nicht gefunden werden, ist es wahrscheinlich nicht installiert. Möglicherweise ist es aber auch nur an einem Ort installiert, der nicht in PATH angeben wurde (den Standardorten, in denen UNIX und Linux nach Programmen suchen). Letzte Klarheit bringt der folgende Befehl:

1 https://de.wikipedia.org/wiki/Ruby%20%28Programmiersprache%29%23Implementierungen
2 http://www.ruby-lang.org
3 http://rubyforge.org/projects/rubyinstaller/

```
$ su -c "updatedb"
$ locate ruby
```

Liefert dieser Befehl nichts oder keine ausführbaren Dateien zurück, dann ist Ruby nicht installiert und nachfolgend wird beschrieben, wie es auf Ihrem Betriebssystem installiert werden kann. Gleiches gilt natürlich auch, wenn eine aktuellere Version verfügbar ist und gewünscht wird.

Fedora Core

Unter Fedora Core und fedorabasierten Distributionen wie Redhat Enterprise und Yellowdog Linux installiert man Ruby einfach mittels des folgenden Befehls:

```
# yum install ruby ruby-devel
```

Debian & Ubuntu

Unter Debian und debianbasierten Distributionen wie z.⊠B. Ubuntu installiert man Ruby einfach mittels des folgenden Befehls:

```
# apt-get install ruby
```

Gentoo

Auch unter Gentoo kann man Ruby leicht installieren:

```
# emerge ruby
```

Arch Linux

Unter Arch Linux installiert man Ruby einfach durch Eingabe des Befehls:

```
# pacman -S ruby
```

FreeBSD

Und selbst unter FreeBSD ist es ein leichtes, Ruby zu installieren:

```
# cd /usr/ports/www/rubygem-rails
# make all install clean
```

OpenBSD

```
# pkg_add -r ruby
```

pkgsrc

Mit pkgsrc ist es ganz ähnlich:

```
# cd /usr/pkgsrc/lang/ruby/
# make install clean clean-depends
```

Selbst kompilieren

Zunächst benötigt man den Quelltext, diesen kann man u.☒a. von der offiziellen Seite[4] herunterladen. Nun öffnet man eine Konsole und wechselt in das Verzeichnis, in dem das heruntergeladene Tar-Archiv liegt. Der erste Schritt ist optional und dient nur der Authentifizierung, dass das Archiv vollständig heruntergeladen wurde und nicht verändert wurde.

```
$ md5sum ruby-<VERSION>.tar.gz
```

<VERSION> muss natürlich durch die Version ersetzt werden, die du herunter geladen hast.

Daraufhin erhält man eine 32 Zeichen lange Prüfsumme. Diese vergleicht man mit dem auf der offiziellen Seite angegebenen Code. Ist er gleich, können wir fortfahren, ansonsten ist die Datei beschädigt und muss erneut geladen werden.

```
$ tar xvzf ruby-<VERSION>.tar.gz
$ cd ruby-<VERSION>
```

Hiermit wird der Quellcode entpackt und in das Verzeichnis gewechselt. Nun müssen wir den Quelltext konfigurieren, also u.☒a. angeben, wohin er installiert werden soll. Dies geschieht mit dem Parameter *--prefix* . Ich empfehle, Ruby nach */usr/* zu installieren, der Interpreter wird dann nach */usr/bin/ruby* installiert. Ansonsten kann man natürlich ein anderes Ziel angeben. Mit dem darauffolgenden Befehl wird das Kompilieren gestartet.

```
$ ./configure --prefix=/usr
$ make
$ # Optionaler Test
$ make test
$ # Als root installieren
$ su -c "make install"
```

2.1.3 MacOS X

Unter MacOS X ist für die allerersten Gehversuche mit Ruby keine zusätzliche Installation notwendig: Ein Ruby-Interpreter wird mit dem Betriebssystem installiert.

Die mit MacOS X 10.4.x (Tiger-Release) ausgelieferte Ruby-Installation ist jedoch veraltet (ruby -v: 1.6.x) und z.☒T. fehlerhaft bzw. unzuverlässig. Eine aktuelle Ruby-Installation 1.8.x – parallele Installation und konfliktfreie Koexistenz zu der betriebssystemnahen – ist dringend zu empfehlen und für die Verwendung von Ruby-on-Rails sogar eine Notwendigkeit. Eine funktionsfähige Anleitung zum Kompilieren (XCode ist hierzu Voraussetzung) und Installieren von Ruby, Rails, Subversion, Mongrel und MySQL unter MacOS X findet sich unter hivelogic[5].

4 http://www.ruby-lang.org
5 http://hivelogic.com/narrative/articles/ruby-rails-mongrel-mysql-osx

2.2 Editor

Als Editor eignet sich im Grunde jeder Editor, der Textdateien im Klartext speichern kann. Trotzdem empfiehlt sich ein Editor, der den Programmierer bei seiner Aufgabe unterstützt. Der Editor sollte mindestens die Einrückung beibehalten und die Syntax farbig hervorheben können. Auch hier gibt es eine gute Auswahl an freien Editoren.

2.2.1 Cross-platform

- NetBeans IDE[6]
- JEdit[7]
- SciTE[8]
- Bluefish[9]

2.2.2 Linux/Unix

- gedit[10]
- Geany[11]

2.2.3 OSX

- Subethaedit[12]
- gedit[13]

2.2.4 Windows

- Notepad++[14]

6 https://de.wikipedia.org/wiki/NetBeans%20IDE
7 https://de.wikipedia.org/wiki/JEdit
8 https://de.wikipedia.org/wiki/SciTE
9 https://de.wikipedia.org/wiki/Bluefish
10 https://de.wikipedia.org/wiki/gedit
11 https://de.wikipedia.org/wiki/Geany
12 https://de.wikipedia.org/wiki/Subethaedit
13 https://de.wikipedia.org/wiki/gedit
14 https://de.wikipedia.org/wiki/Notepad%2B%2B

3 Hallo Welt!

In diesem Kapitel wollen wir die ersten Schritte in der Rubywelt unternehmen.

3.1 Hallo Welt

Das erste Programm in einer neuen Programmiersprache dient dazu, sich mit der Programmierumgebung vertraut zu machen. Es gilt zu lernen, wie man den Programmtext editiert und ihn schließlich zum Ausführen bringt. Es soll zunächst ein Programm geschrieben werden, das die Wörter "Hallo Welt" ausgibt, und sich anschließend wieder beendet. In Ruby sieht der Quelltext folgendermaßen aus:

```
puts "Hallo Welt!"
```

Dafür öffnet man den Editor der Wahl, tippt diese Zeile ein und speichert das Programm als *hello.rb* . Unter Windows kann man nun das Skript direkt starten, wie man auch ein normales Programm benutzen würde. Unter Linux muss man entweder die sogenannte Shebang[1]-Zeile benutzen (siehe unten) oder dem Ruby-Interpreter mitteilen, dass er das Skript ausführen soll. Dazu öffnet man das Terminal, wechselt mit `$ cd PATH` in den entsprechenden Ordner und startet den Ruby-Interpreter mit dem Skriptnamen als Parameter.

```
$ ruby hello.rb
```

In beiden Fällen passiert dasselbe. Die Methode `puts` wird aufgerufen. Sie dient dazu, Zeichen auf dem Bildschirm auszugeben. Wir wollen die Wörter "Hallo Welt!" ausgeben, also einen Text. Texte werden in Ruby als Zeichenkette (oder auch **String**) bezeichnet, genaugenommen handelt es sich um eine Instanz des Objekts String. Die Behandlung von Objekten ist an dieser Stelle jedoch nicht nötig, daher folgt die ausführliche Beschreibung von Objekten im Kapitel Objektorientierung. Zeichenketten werden durch einfache oder doppelte Anführungszeichen begrenzt. Alles zwischen zwei Anführungszeichen gehört zu einem String.

Wenn man unter Windows das Skript nicht via Kommandozeile ausführt, dann öffnet sich die Kommandozeile und wird direkt wieder geschlossen. Mit folgender Änderung kann man das umgehen:

```
puts "Hallo Welt"
gets
```

1 https://de.wikipedia.org/wiki/Shebang

Die Methode `gets` dient dazu, Eingaben von der Tastatur zu lesen. An dieser Stelle wird das Skript angehalten und erst dann fortgesetzt, wenn der Benutzer die [Eingabe]-Taste gedrückt hat, um seine Eingabe abzuschließen. Was der Benutzer tatsächlich eingegeben hat, interessiert an dieser Stelle nicht. Das Programm soll lediglich so lange warten, bis der Benutzer die Ausgabe gelesen hat und dies mit dem Drücken der [Eingabe]-Taste anzeigt.

3.2 Kommentare

In längeren Programmen kann es notwendig sein, die Funktion des Programms oder einzelner Methoden durch zusätzliche Informationen zu erklären. Dazu gibt es in allen Programmiersprachen **Kommentare** . Diese sind eigentlich keine Rubybefehle, sondern teilen dem Interpreter mit, dass alle Zeichen nach einer Raute # bis zum Zeilenende Kommentare sind und vom Rubyinterpreter ignoriert werden. Kommentare haben auf das Verhalten des Rubyskriptes keinen Einfluss.

```
# Gibt Hallo Welt! aus und wartet auf die Eingabe des Benutzers
puts "Hallo Welt!"
gets
```

Hierbei handelt es sich um einen Kommentar, da er mit einer Raute beginnt. Er erklärt dem Leser des Programms, was das Programm tut. Auf die Wirkung des Programms hat er keine Auswirkung. Ohne Kommentar stellt man keinen Unterschied in der Funktionsweise fest. Bei umfangreicheren Kommentaren, die sich über mehrere Zeilen erstrecken ist es möglich mehrere Zeilen mit einer Raute zu beginnen, oder auf folgende Syntax mit Gleichheitszeichen auszuweichen:

```
=begin
Das ist ein
mehrzeiliger Kommentar!
=end
```

3.3 Shebang

Die Shebang-Zeile ist eine spezielle Form des Kommentars. Sie ist auf Linuxsystemen nötig, um dem Betriebssystem mitzuteilen, welches Programm das folgende Skript interpretieren soll.

```
#!/usr/bin/env ruby
```

Der genaue Pfad kann sich dabei unterscheiden, je nachdem in welchem Verzeichnis der Interpreter installiert wurde. Danach muss man das Skript ausführbar machen. Dazu dient folgender Konsolenbefehl:

```
$ chmod u+x hello.rb
```

Dann ist es möglich, das Rubyskript auch unter Linux wie ein normales Programm zu benutzen. Nötig ist die Shebang-Zeile in zwei Fällen: Zum einen kann es sein, dass Sie Kon-

solenprogramme schreiben wollen und diese danach benutzen wollen, ohne daran zu denken, dass es sich um ein Ruby-Skript handelt, denn jetzt können Sie es wie jedes andere Programm auch mit `$ ./hello.rb` ausführen. Sie können sogar, anders als unter Windows, auf die Dateiendung verzichten, was die Nähe zu einem Programm zusätzlich erhöht. Ansonsten erfordert die Benutzung des Frameworks *Ruby on Rails* für Web-Entwicklungen die Shebang-Zeile.

4 Rechnen

Ein Computer als Rechenmaschine kann von sich aus nicht viel mehr als zu rechnen und alles, was Menschen heute mit Computern tun, ist eine Abstraktion von sehr einfachen Rechnungen. Diese grundlegende Nähe zur Mathematik zeigt sich in vielerlei Hinsicht auch in sowohl der Benennung, als auch der Denkweise von Programmiersprachen. Erst später in diesem Buch wird sich thematisch von der Mathematik immer weiter entfernt, indem weitere Abstraktionen eingeführt werden. Dieses Kapitel beschäftigt sich mit den Grundrechenarten in Ruby. Zunächst werden Variablen in Ruby dargestellt, danach folgt eine Einführung in Operatoren und zum Schluss wird mit diesem Wissen ein Programm implementiert, das den Flächeninhalt eines Kreises mit gegebenem Radius ermittelt.

4.1 Variablen

Variablen in Ruby sind Platzhalter. Sie speichern andere Daten und erlauben einen einfachen Zugriff auf diese Daten, die sonst nicht oder nur schlecht verständlich und lesbar sind. Betrachtet man das folgende einfache Skript, wird dies klarer.

```
a = 5
puts a
```

Dieses kleine Skript tut nichts weiter, als die Zahl 5 auf der Kommandozeile auszugeben. Dies ist ganz analog zu dem einführenden ”Hallo Welt”-Beispiel. Der einzige Unterschied besteht darin, dass die Daten vorher in der Variable `a` gespeichert werden und danach auf diese Daten zugegriffen wird. Die Verwendung von Variablen erlaubt zwei Vorteile gegenüber der direkten Verwendung von Daten: Zum einen erlauben Variablen eine Wiederverwendung der gleichen Daten und bei einem Fehler müssen die Daten nur an einer Stelle geändert werden. Zum anderen erlauben es Variablennamen, die Lesbarkeit deutlich zu verbessern.

Variablen in Ruby sind untypisiert. Versucht man also, der Variablen `a` nach der Zuweisung der Zahl 5 beispielsweise den String `"Hallo"` zuzuweisen, führt das nicht zu einem Fehler, sondern der ursprüngliche Wert wird überschrieben.

Relevant sind an dieser Stelle mögliche Namen von Variablen. Wie oben gesehen, ist `a` eine gültige Bezeichnung einer Variablen. Normale Variablen beginnen mit einem kleinen Buchstaben. Die weiteren Zeichen können frei aus großen und kleinen Buchstaben und dem Unterstrich gewählt werden. Es ist üblich, sprechende Namen zu verwenden, um ein weiteres Maß an Übersichtlichkeit hinzuzufügen. Statt `x = 3.14` sollte man also besser `pi = 3.14` verwenden.

4.1.1 Konstanten

Da bereits erwähnt wurde, dass Variablen mit einem kleinen Buchstaben beginnen müssen, stellt sich die Frage, was mit Bezeichnern wie `Test` oder `TEST` ist. Diese beiden bezeichnen keine Variablen. Im ersten Fall handelt es sich um eine Klassen- oder Modulbezeichnung, die im zweiten Teil des Buches über Objektorientierung behandelt wird. Bei Letzterem handelt es sich um eine **Konstante** . Betrachtet man die Ausführung des Skriptes:

```
TEST = 5
puts TEST
TEST = 4
puts TEST
```

Das Programm gibt, wie erwartet, erst 5 und danach 4 aus, dazwischen jedoch zeigt der Interpreter eine Warnung an, dass man gerade die Konstante überschrieben hat. Konstanten sind dann sinnvoll, wenn man ein unbeabsichtigtes späteres Überschreiben einer Variablen verhindern will.

4.1.2 Globale Variablen

Ein weiterer Typ von Variablen beginnt mit einem Dollarzeichen; das sind globale Variablen. Im Kapitel Methoden[1] wird die Sichtbarkeit von Variablen erklärt. Globale Variablen heißen so, weil sie aus jeder Methode heraus sichtbar sind. Sichtbar bedeutet in diesem Zusammenhang, dass auf sie in jeder Methode zugegriffen und sie verändern kann.

4.2 Grundrechenarten

Die Grundrechenarten Addition, Multiplikation usw. werden in Programmiersprachen durch **arithmetische Operatoren** ausgeführt. Operatoren arbeiten mit ein, zwei oder drei Variablen. Der folgende Code zeigt das:

```
a = b + c
```

In diesem Beispiel sind bereits zwei Operatoren dargestellt. Zunächst wird die Summe von `b` und `c` ermittelt und dann der Variablen `a` zugewiesen. Genauso funktionieren auch die anderen Grundrechenarten und die Potenz. Eine Besonderheit von Ruby ist es, dass auch alle Operatoren Methoden sind, die in einem Programm verändert werden können.

4.3 Flächeninhalt eines Kreises

Der Flächeneinhalt eines beliebigen Kreises ist das Produkt der Zahl Pi und des quadratischen Radius'. Mit den im vorherigen Kapitel vorgestellten Methoden `gets` und `puts` sowie Variablen und Operatoren ist man in der Lage ein Kommandozeilenprogramm zu schreiben, das den Flächeninhalt eines Kreises mit gegebenem Radius ermittelt.

1 Kapitel 6 auf Seite 21

```
#circle_area.rb
PI = 3.142
puts "Dieses kleine Programm berechnet den Flaecheninhalt eines Kreises!"
puts "Bitte nun den Radius eingeben:"
radius = gets.to_f
flaeche = radius**2 * PI
puts "Der Flaecheninhalt betraegt: " + flaeche.to_s
```

Nach dem Lesen der vorherigen beiden Seiten sollten Sie in der Lage sein dieses Programm grundsätzlich zu verstehen.

- Zeile 2: Eine Konstante mit dem Namen `PI` wird initialisiert und erhält den Wert 3.142
- Zeile 5: Mit der Methode `gets` liest das Programm die Benutzereingabe und mit dem Zusatz `.to_f` wird die Eingabe in eine Fließkommazahl umgewandelt. Dabei wird nicht geprüft, ob es sich um eine korrekte Eingabe handelt. Bei falscher Bedienung hat `radius` den Wert 0.0.
- Zeile 6: Der Radius wird quadriert und mit Pi multipliziert, danach steht der Flächeninhalt in der Variablen `flaeche` .
- Zeile 7: Die Methode `puts` wird aufgerufen und gibt den Flächeninhalt aus.

`puts` erwartet eine Zeichenkette. An dieser Stelle zeigt sich eine weitere Stärke von Ruby: die Polymorphie. Das bedeutet, dass dieselbe Bezeichnung für Funktionen in Abhängigkeit von den übergebenen Typen verschiedenen Nutzen haben kann. An späterer Stelle werden wir feststellen, dass es sich bei den oben vorgestellten Operatoren auch um Methoden handelt. Im Falle zweier Zahlen erfüllt die Methode `+` ihren offensichtlichen Zweck. Steht aber vor und nach dem `+` je ein String, dann erfüllt es einen anderen Zweck: Es bildet einen neuen String, wobei der Zweite an den Ersten angehängt wird.

5 Kontrollstrukturen

Diese Seite beschäftigt sich mit **Kontrollstrukturen** . Um etwas auf die Frage einzugehen, was eine Programmiersprache ausmacht und warum andere Sprachen keine Programmiersprachen sind. Die wichtigsten beiden Eigenschaften einer Programmiersprache sind, Quellcode wiederholt und optional ausführen zu können. Für verschiedene Sprachen existieren unterschiedliche Implementierungen dieser beiden Grundlagen. Ruby kennt Verzweigungen, zur optionalen Ausführung von Programmtext. Es lassen sich innerhalb des Programmflusses Entscheidungen treffen und auf sie reagieren. Zum Beispiel, wenn eine falsche Benutzereingabe behandelt werden muss.

Für die wiederholte Ausführung von Programmteilen gibt es in Ruby drei wesentliche Möglichkeiten, mit aufsteigender Präferenz: Schleifen, Rekursionen und Iterationen. An dieser Stelle wird nur auf Schleifen eingegangen. Rekursionen sind Teil des Kapitels Methoden[1], während Iterationen im zweiten Teil des Buches thematisiert werden.

5.1 Verzweigungen

Für die Behandlung von **Verzweigungen** ist es zunächst nötig sich mit Vergleichs- und Logikoperatoren zu beschäftigen. Es geht darum Daten auf etwas hin zu prüfen, zum Beispiel, ob eine Zahl größer ist als ein erwarteter Wert und die Ergebnisse verschiedener Prüfungen logisch zu kombinieren. Die folgende Tabelle gibt eine Übersicht über die vorhandenen Vergleichs- und Logikoperatoren, wobei die einzelne Funktionalität in Abhängigkeit von den übergebenen Typen variieren kann.

Operator	**Erklärung**
`==`	Prüft auf Gleichheit
`<`	Kleiner als
`>`	Größer als
`<=`	Kleiner gleich
`>=`	Größer gleich
`!`	logisches nicht
`&&`	logisches und
`||`	logisches oder

Die eigentliche Verzweigung im Programmtext erfolgt über die Schlüsselwörter `if` , `elsif` , `else` und `unless` , denen ein Wahrheitswert übergeben wird, der typischerweise durch oben betrachtete Operatoren ermittelt wird. Das folgende Skript gibt dabei Beispiele für verschiedene Anwendungsfälle. Es gilt zu beachten, dass `elsif` ohne 'e' geschrieben wird.

1 Kapitel 6 auf Seite 21

```
a = 2
b = 3
if a > b
  puts "a ist groesser"
elsif b == 10
  puts "b ist zehn!"
else
  puts "Standard"
end
```

Jede Kontrollanweisung leitet dabei einen Block ein, in dem sie gilt. Beendet wird ein Block mit dem Schlüsselwort `end` . Beim Ausführen des Skripts wird der Text ”Standard” ausgegeben, da weder `a` größer als `b` , noch `b` gleich zehn ist. Die Anweisung `unless STATEMENT` ist äuqivalent zur Anweisung `if !STATEMENT` , der Programmblock wird also nur Ausgeführt, wenn die Bedingung nicht erfüllt wird. In Ruby sind nur die Werte `nil` und `false` logisch falsch. Insbesondere Programmierer, die bereits C oder eine ähnliche Sprache können, müssen an dieser Stelle aufpassen, dass eine 0 logisch wahr ist.

Eine Möglichkeit seinen Programmtext stilistisch zu verbessern ist es, eine Kontrollabfrage hinter den eigentlichen Kontrollblock zu schreiben. Dieser Kontrollblock besteht dann lediglich aus einer Zeile, nämlich allem, was vor dem entsprechenden Schlüsselwort steht. Das Skript von oben lässt sich also auch in kürzerer Form schreiben. Dabei gilt zu beachten, dass inbesondere die fünfte Zeile deutlich weniger lesbar ist, als die entsprechende Zeile im obigen Skript.

```
a = 2
b = 3
puts "a ist groesser" if a > b
puts "b ist zehn!" if b == 10
puts "Standard" unless a > b || b == 10
```

Eine weitere Möglichkeit ist das Wählen der Verzweigung mit `case ... when` . Mit `case` wählt man aus, was man überprüfen möchte und mit `when` wechselt man die Fälle. Wie bei `if` -Anweisungen ist es Möglich mit `else` einen Default-Fall zu erzeugen, falls sonst nichts zutrifft. Die gleiche Behandlung mehrerer Fälle erfolgt durch Trennung mit einem Komma. Zum Beispiel:

```
a = 2

case a
when 1
  puts "Eins!"
when 2, 3
  puts "Zwei oder Drei!"
else
  puts "Irgendetwas!"
end
```

5.2 Schleifen

Die einfachste Form einer **Schleife** ist die Endlosschleife. Sie entsteht oftmals als Folge von Programmierfehlern, kann aber unter Umständen auch gewünscht sein, wenn sie zum Beispiel in Folge einer Benutzereingabe abgebrochen wird. Somit handelt es sich nicht um eine echte Endlosschleife.

```
loop do
  input = gets
  break if input == "exit\n"
  puts "Ihre Eingabe lautete: " + input
  next
  puts "Diese Zeile wird nie ausgegeben"
end
```

Wenn Sie dieses Skript starten und ein paar Eingaben tätigen, werden Sie feststellen, dass Sie beliebig oft Eingaben tätigen können und diese ausgegeben werden. In dem Skript finden sich die Schlüsselwörter `next` und `break` . Mit `break` verlässt man die aktuelle Schleife, in diesem Fall ist also `if eingabe == "exit\n"` die Abbruchbedingung für die Pseudoendlosschleife. Es sei noch einmal betont, dass eine wirkliche Endlosschleife selten erwünscht ist, da man das Programm nur noch extern beenden kann. Während man mit `break` also hinter die Schleife springt, springt `next` an den Anfang der Schleife, es steht einfach für den nächsten Schleifendurchlauf. Alle Formen von Schleifen sind ineinander umformbar, doch ist es offensichtlich, dass ein stetiges Prüfen der Abbruchbedingung am Anfang einer Endlosschleife syntaktisch nicht schön ist, deswegen gibt es zwei weitere Schleifenschlüsselwörter. Im folgenden Skript ist eine `while` - bzw. `until` -Schleife implementiert, deren Funktion dem eben dargestellten Skript gleicht. Die Schlüsselwörter `while` und `until` verhalten sich zueinander wie `if` und `unless` .

```
input = ""
until input == "exit\n"
  input = gets
  puts input
end
```

Eine weitere Möglichkeit eine Schleife zu bilden ist `for` . Diese Schleife kommt dabei nicht zu derselben Bedeutung wie ihr Synonym in C oder Java, sondern wird weitaus weniger eingesetzt. Sie wird insbesondere durch die Iterator-Methoden[2] verdrängt, die eine weitaus mächtigere Möglichkeit bieten, Arrays und andere Datenstrukturen zu behandeln. Trotzdem sei eine `for` -Schleife der Vollständigkeit halber hier erwähnt, denn sie kann notwendig werden, wenn man eigene Iteratoren definieren möchte.

```
for i in (0..10)
  puts i
end
```

Bei dem Konstrukt `(0..10)` handelt es sich um eine Range, also alle Zahlen zwischen 0 und 10. Ranges gibt es in einer inklusiven Variante mit zwei Punkten und einer exklusiven Variante mit drei Punkten.

5.3 Fakultät

Die Fakultät ist eine mathematische Funktion, die natürliche Zahlen verarbeiten kann. Das Implementieren der Fakultät ist eine Standardaufgabe beim Lernen einer Programmiersprache, da man dafür die auf dieser Seite gelernten Kontrollstrukturen benutzen muss und somit schnell vertraute Strukturen aus anderen Programmiersprachen wiedererkennt, oder

2 Kapitel 18 auf Seite 61

auf die Unterschiede stößt. Die Fakultät ist wie folgt definiert: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = \prod_{k=1}^{n} k$, außerdem gilt $0! = 1$.

```
#fakultaet.rb
#ermittelt die fakultaet einer vom Benutzer eingegebenen Zahl

eingabe = ""
until eingabe == "exit\n"
  eingabe = gets
  n = eingabe.to_i
  if n < 0
    puts "Fehlerhafte Eingabe!"
    next
  elsif n == 0
    ergebnis = 1
  else
    ergebnis = 1
    until n == 1
      ergebnis *= n
      n -= 1
    end
  end
  puts "Die berechnete Fakultaet ist: " + ergebnis.to_s  unless eingabe ==
 "exit\n"
end
```

Wieder sollte Ihnen vieles vertraut vorkommen. Zur Erklärung des Algorithmus' gibt es lediglich zu sagen, dass `ergebnis` mit 1 initialisiert wird und solange mit dem herunterzählenden `n` multipliziert wird, bis dieses den Wert 1 hat und die Fakultät fertig berechnet ist.

- Zeile 07: `.to_i` dient der Typumwandlung zu einer natürlichen Zahl.
- Zeile 08: Abfangen einer fehlerhaften, negativen Eingabe.
- Zeile 16: `ergebnis *= n` ist äquivalent zu `ergebnis = ergebnis * n`

5.4 ? :

Für das häufig vorkommende `if` und `else` gibt es eine Kurzschreibweise mit `? :` . Dieser kryptische Operator stammt aus der Programmiersprache C, wo er hinzugefügt wurde, um Parameter für Funktionen auszuwählen. Er ist mit Vorsicht anzuwenden, da er den Lesefluss des Programms deutlich einschränken kann, was der Philosophie eines leicht lesbaren Programmtextes in Ruby zuwider ist. Er kann jedoch insbesondere bei kurzen Verzweigungen vorteilhaft sein. Folgende beiden Programmabschnitte sind dabei äquivalent:

```
# 1. Variante mit if und else
if 1 == 2
  puts "Etwas ist seltsam."
else
  puts "Hier ist alles korrekt."
end

# 2. Variante mit ? :
1 == 2 ? puts("Eins ist gleich zwei!") : puts("Nein, alles ist richtig.")
```

6 Methoden

Der Sinn einer Programmiersprache besteht darin die Möglichkeiten, aber auch Begrenzungen, des Computers möglichst weit zu abstrahieren. So wird es dem Entwickler möglich seinen Quelltext nah am Problem zu gestallten, statt sich mit der Technik und Funktionsweise des Computers beschäftigen zu müssen. Eine erste Form der Abstraktion sind Methoden. Betrachtet man zum Beispiel die Funktion `puts` so wissen wir, dass sie auf der Kommandozeile eine Ausgabe tätigt. Es ist nicht wichtig, *wie* die Methode eine Funktion bereit stellt, sondern nur *was* sie ermöglicht. In diesem Kapitel geht es um die Definition eigener Methoden.

6.1 Methoden

```
def hallo
  puts "Hallo Welt!"
end

hallo
```

Methodendefinitionen werden mit dem Schlüsselwort `def` eingeleitet und der Block wie im Falle der Kontrollstrukturen mit `end` beendet, alles was zwischen diesen beiden Zeilen steht wird beim Aufrufen der Funktion ausgeführt, es ist also ohne Probleme möglich mehrzeilige Methoden zu schreiben. Durch einfaches Schreiben des Methodennamens in einem Programmtext, wird diese an der Stelle aufgerufen und ausgeführt.

Bei Methodennamen gelten ähnliche Regeln wie bei Variablennamen. Zusätzlich ist die Konvention der Methodennamen möglichst nah an der Funktion zu wählen und ggf. eine Warnung mitzugeben. Eine Warnung ist beispielsweise dann nötig, wenn die Funktion die übergebenen Parameter verändert, oder Daten auf der Festplatte verändert. Dieses Verhalten wird auch als Seiteneffekte bezeichnet und wird mit einem Ausrufezeichen am Ende des Methodennamens angezeigt. Eine Prüfung endet mit einem Fragezeichen.

```
def is_two? (zahl)
  puts "Ja!" if zahl == 2
  puts "Nein!" unless zahl == 2
end
```

6.2 Parameter

Bleiben wir bei der Betrachtung der Methode `is_two?` , so haben wir bereits beim ersten Programm festgestellt, dass sie einen **Parameter** erhält, nämlich die Zahl, die auf

die Gleichheit mit 2 geprüft werden soll. Methoden können entsprechend der übergebenen Parameter auch unterschiedliche Funktionen enthalten (Zum Beispiel: `+`).

```
def hallo(ort)
  puts "Hallo " + ort
end
hallo "Welt"
```

Hier wird der Methode ein String mit dem Wert "Welt" übergeben. Die Methode gibt daraufhin den Text "Hallo Welt" aus. Die Verwendung von Klammern bei Parametern einer Methode ist in den meisten Fällen optional. In diesem Buch werden bei Methodendefinitionen Klammern verwendet. Beim Aufruf der Methode werden sie je nach Anzahl der Parameter weggelassen.

Es ist auch möglich, den Parametern von Methoden Standardwerte zuzuteilen. Im nachfolgenden Beispiel wird der Variablen `ort` der Wert "Welt" zugeteilt. Wenn man die Funktion mit Parameter aufruft, dann überschreibt der übergebene Parameter den Standardwert. Im nachfolgenden Skript wird der Parameter `ort` mit `"Erde"` initialisiert, dadurch ist es möglich die Methode ohne Parameter aufzrufen, wodurch `"Hallo Erde"` ausgegeben wird, oder diesen Parameter wie oben manuell zu setzen.

```
def hallo(ort = "Erde")
  puts "Hallo " + ort
end
```

6.3 Rückgabewerte

Wenn man sich klar macht, dass Programmiersprachen in den Anfängen vor allem von Mathematikern entwickeln wurden, dann wird deutlich warum Methoden (in anderen Programmiersprachen auch Funktionen) Gemeinsamkeiten mit den mathematischen Funktionen haben. Eine mathematische Funktion erhält Parameter in Form von Variablen und gibt einen Wert (oder mehrere Werte) zurück. Auch Methoden müssen also Rückgabewerte ermöglichen. In Ruby gibt es zwei Möglichkeiten, dass Methoden Werte zurückgeben. Im ersten und einfachsten Fall ist der Rückgabewert der letzte errechnete Wert vor dem verlassen der Funktion. Im zweiten Fall kann man mit dem Schlüsselwort `return` anzeigen, dass man sowohl die Methoden verlässt und der Wert nach `return` zurückgegeben werden soll.

```
def addiere(a, b)
  a + b
end

#ist das gleiche wie:
def addiere(a, b)
  return a + b
end
```

6.4 Mehrere Quelltextdateien

Bei längeren Programmtexten wird es auch in einer gut lesbaren Sprache wie Ruby notwendig, seinen Quelltext zu strukturieren. Daher ist es oftmals sinnvoll, logisch zusam-

menhängende Bereiche eines Programms in einer eigenen Quelltextdatei zu schreiben. Das Einbinden erfolgt dann mit dem Schlüsselwort `require` . Standardmäßig sucht `require` in dem Ordner, indem auch Rubygems seine Erweiterungen speichert, dies kann jedoch durch angabe eines konkreten Pfades zur Datei geändert werden. Will man stattdessen einen Pfad relativ zur aufrufenden Datei angeben so verwendet man `require_relative` , das mit Ruby 1.9 eingeführt wurde. Im Gegensatz zu `require` ist es deutlich flexibler. Im unten stehenden Beispiel führt zwar ein `require './hallo2.rb'` zu einem funktionierenden Programm, doch nur, wenn das Skript aus dem selben Verzeichnis aufgerufen wird, da der Punkt in der Pfadangabe nicht relativ zum Skript ausgewertet wird, sondern für das aktuelle Arbeitsverzeichnis steht. Außer zur Verwendung von Gems sollte man auf `require` verzichten.

```
#hallo1.rb
require_relative 'hallo2.rb'
hallo

#hallo2.rb
def hallo
  puts 'Hallo Welt!'
end
```

Speichert man beide Skripte im gleichen Verzeichnis ab und startet `hallo1.rb` , dann führt es zur Ausgabe von `Hallo Welt!` . Das Skript importiert zunächst alle Definitionen aus dem Skript `hallo2.rb` , in diesem Fall also die Definition einer Methode `hallo` , die nun in `hallo1.rb` genauso zur Verfügung steht, als wäre sie dort definiert.

6.5 Rekursion und Sichtbarkeit von Variablen

Rekursion bezeichnet einen Vorgang, dass sich innerhalb einer Methode ein weiterer Methodenaufruf der selben Methoden befindet. Mit dieser insbesondere in funktionalen Programmiersprachen beliebten Möglichkeit lassen sich einige Probleme eleganter formulieren, als es mit Schleifen möglich wäre. Bevor wir jedoch eine einfache rekursive Methode betrachten, ist es nötig, sich die Sichtbarkeit von Variablen anzuschauen.

```
def n_wird_zwei(n)
  n = 2
end
n = 1
$n = 3
n_wird_zwei n
n_wird_zwei $n
puts n
puts $n
```

Entgegen der ersten Annahme gibt dieses Programm erst 1, dann jedoch 2 aus. Das hat zum einen etwas mit der **Sichtbarkeit** von Variablen zu tun. Beim Aufrufen der Funktion wird nicht die tatsächliche Variable übergeben, sondern erst kopiert und dann übergeben. Von dieser Regel ausgenommen sind globale Variablen, daher verändert die Methode nicht die lokale Variable `n` , sondern die globale Variable `$n` . Das ist der Grund, warum die oben erwähnte Namenskonvention erst bei der späteren, genaueren Betrachtung von Methoden wichtig wird. Die Methode `n_wird_zwei` verändert also nicht die übergebene Variable, sondern gibt stattdessen den Wert 2 zurück. Erinnern wir uns an die eingangs erwähnte Rekur-

sion, dann ist dieses Verhalten von übergebenen Parametern sehr wünschenswert, denn wir können innerhalb einer Methode diese wieder (und auch andere Methoden) aufrufen ohne erst unsere Variablen in temporären Variablen zu sichern. Folgende sehr einfache Rekursion zeigt das Herunterzählen von 10 bis 0.

```
def decrease(n)
  return if n < 0
  puts n
  decrease n-1
end

decrease 10
```

Dieses Beispiel hätte man natürlich auch mit einer Schleife lösen können. Das ist ein weit verbreitetes Problem beim Lehren von Rekursionen, denn die einfachen Bespiele sind viel einfacher durch Schleifen darstellbar und wirken daher überflüssig. Erinnert man sich jedoch an den einleitenden Text dieser Seite, so geht es bei der Frage, ob man Rekursion oder Schleifen benutzt lediglich darum, welche Implementierung näher am Problem ist.

6.6 Rückblick: Fakultät

Mit diesem Kapitel ist es möglich die auf der letzten Seite vorgestellte Fakultätsfunktion sehr viel prägnanter zu schreiben. Die Fakultät lässt sich auch rekursiv definieren, denn $0! = 1$ und $1! = 1$ sowie $n! = n(n-1)!$. So lässt sich die Fakultät wie folgt rekursiv abbilden:

```
#fakul.rb
#Berechnet die Fakultaet

def fakul(n)
  if n == 0
    1               # 0! = 1
  else
    n * fakul n-1   # n! = n*(n-1)!
  end
end

puts fakul(5)
```

In diesem Beispiel wird die Frage, ob eine Schleife oder Rekursion sinnvoller ist, schon deutlicher. Formuliert man das Problem rekursiv, dann ist selbstverständlich eine rekursive Abbildung sinnvoller und umgekehrt. Im Normalfall ist die Ausführung der äquivalenten Schleife schneller als Rekursionen, jedoch gibt es automatisierte Konvertierungstechniken zwischen beiden Varianten. Es mag also einen Geschwindigkeitsvorteil geben, doch sollten Sie zunächst auf die Lesbarkeit des Codes achten.

7 Strings

String ist der erste Datentyp in diesem Buch, der Ihnen explizit und ausführlich vorgestellt wird. Dabei ist String die Bezeichnung für eine Zeichenkette. Sie haben Strings bereits ganz zu Beginn des Buches kennengelernt, in Form von `puts "Hallo Welt!"` und auch schon einige Methoden im Umgang mit Strings. Diese Seite soll nun Strings genauer beleuchten und einige Tücken im Umgang mit ihnen klären.

Für Programmierer anderer Sprachen, insbesondere C-Ähnlichen, ist zu sagen, dass Strings kein Array aus Zeichen sind, sondern ein eigener Datentyp. Der String ersetzt dabei einzelne Zeichen komplett und ein einzelnes Zeichen ist einfach ein String der Länge eins.

7.1 Anführungszeichen

Strings können auf mehrere Arten begrenzt werden. Es gibt die Möglichkeit, sie durch einfache und doppelte Anführungszeichen zu kennzeichnen. Der Unterschied besteht darin, ob innerhalb des Strings bestimmte Zeichenketten gesondert ausgewertet werden. Wenn man die Ausgabe des folgenden Skripts betrachtet, stellt man genau diesen Unterschied fest.

Für Strings existiert die Möglichkeit, sie mit folgender Syntax zu definieren `%(String)` . Die Wahl von Begrenzern ist dabei relativ frei, so können Sie statt `%()` auch unter anderem eckige und geschweifte Klammern verwenden. Das ist insbesondere von Vorteil, wenn Sie mit Strings umgehen müssen, die viele Anführungszeichen enthalten.

```
a = 5
puts '#{ a }'
puts "#{ a }"
puts %(#{ a })
```

Man nennt den String mit doppelten Anführungszeichen Formatstring. In einem Formatstring stehen wie oben erwähnt bestimmte Escape-Sequenzen zur Verfügung, die das Layout der Ausgabe ändern.

- Einfache Anführungszeichen
  - `\'` −einfache Anführungszeichen
  - `\\` −single backslash
- Doppelte Anführungszeichen
  - `\"` −doppelte Anführungszeichen
  - `\\` −backslash
  - `\a` −bell/alert[1]

1 `https://de.wikipedia.org/wiki/bell%20character`

- `\b` −backspace[2]
- `\r` −carriage return[3]
- `\n` −newline[4]
- `\s` −space[5]
- `\t` −tab[6]

7.2 Einfügen von Variablen in Strings

Auf der Seite Rechnen[7] wurde bereits eine Variable an einen String angefügt. Dies geschah durch die Methode + und dem Umformen der Zahl zu einem String. Dieses Verfahren ist natürlich auch an dieser Stelle möglich, aber man kann sich leicht vorstellen, dass das Einfügen von mehreren Zahlen auf diese Weise sehr umständlich ist. Dafür gibt es zwei Möglichkeiten. Das folgende Listing stellt einige Möglichkeiten gegenüber:

```
a = 5

puts "a: #{ a }"
puts "a: " << a.to_s
puts "a: " + a.to_s
```

Da die erste Variante auch bei vielen Variablen und einem großen String übersichtlich bleibt, ist das die konventionelle Vorgehensweise beim erstellen einen Strings mit Daten des Programms. Liegen die Strings nicht als Konstante vor, sondern nur als Variable kann es notwending sein eine der beiden anderen Methoden zu verwenden. Man sollte dann die Lesbarkeit von `string << data.to_s` mit `"#{ string }#{ data }"` zu vergleichen. Der wesentliche Unterschied zwischen + und << ist, dass + ein neues Objekt erzeugt und dadurch wesentlich langsamer ist als << . Die Verwendung von << kann dahingegen zu unerwünschten Nebeneffekten führen:

```
def append_world_to(string)
  string << "World"
end

s = "Hello "
result = append_world_to(s)
puts result
puts s
```

In diesem Fall hat die Methode `append_world_to` den (möglichweise unerwünschten) Nebeneffekt, dass es nicht nur ein neues Ergebniss produziert, sondern auch das Original manipuliert.

2 https://de.wikipedia.org/wiki/backspace
3 https://de.wikipedia.org/wiki/carriage%20return
4 https://de.wikipedia.org/wiki/newline
5 https://de.wikipedia.org/wiki/space%20character
6 https://de.wikipedia.org/wiki/tab%20key
7 Kapitel 4 auf Seite 13

7.3 Teilstrings

Nicht nur kann es nötig sein Daten in Strings einzufügen, sondern sie auch wieder zu extrahieren bzw. Teilstrings zu erzeugen.

```
s = "Hallo Wikipedia!"
puts s[0]
puts s[0,5]
puts s["Wikipedia"]
puts s["Welt"]
puts s[/\w+/]
```

Der Zugriff mit einem String gibt den Text aus, wenn der entsprechende String vorkommt. Die anderen Methoden sind angelehnt an den Zugriff auf Arrays oder nutzen Reguläre Ausdrücke, daher sollten Sie eventuell die entsprechenden Kapitel dieses Buches zunächst aufsuchen. Der Zugriff mit Zahlen bezieht sich auf die Position im String und bei der Verwendung von Regulären Ausdrücken wird der erste Treffer ausgewählt.

7.4 Symbole

Symbole sind spezialisierte Strings, die vor allem zur internen Darstellung von Zeichenketten in der Software dienen. Im Gegensatz zu Strings, die bei jedem Auftreffen einer Konstante neu erzeugt werden müssen werden Symbole nur einmal erzeugt und danach behalten, dadurch erlauben sich in bestimmten Fällen Geschwindigkeitsvorteile. Symbole werden auch häufig verwendet, wenn man Meta-Programmiert. Die Initialisierung erfolgt durch vorranstellen eines Doppelpunktes vor einen String, falls der String aus nur einem Wort besteht, kann auf die Anführungszeichen auch ganz verzichtet werden.

```
puts :hello
puts :"Hello World"
puts :'Hello Reader'
```

8 Zahlen

Diese Seite behandelt Zahlen, dabei ist es wichtig zunächst zwischen verschiedenen Arten von Zahlen zu unterscheiden. Es gibt ganze Zahlen (Integer) oder gebrochene Zahlen (Floating Point Numbers). Diese beiden Grundtypen sind bei einem Computer die wichtigsten und finden häufig Anwendung. Ein Computer kann keine reellen Zahlen verarbeiten, da er dafür unendlich viel Speicherplatz bräuchte. Man kann also beispielsweise Pi zwar beliebig genau bestimmen (Speicherplatz- und Zeitbegrenzt), wird aber nie einen Wert für Pi errechnen. Da Sie das grundsätzliche Rechnen mit Zahlen bereits kennengelernt haben, wird in diesem Kapitel lediglich auf einige Tücken eingegangen, die im Zusammenhang mit Zahlen auftreten können.

8.1 Integer

Bei der Arithmetik mit ganzen Zahlen findet keine automatische Typanpassung statt. Das kann insbesondere beim dividieren von ganzen zahlen zu seltsamen Fehlern kommen:

```
puts 5 / 2
puts 5 % 2
puts 5 / 2.0
```

Naiv nehmen wir an, dass `puts 5 / 2` als Ergebniss `2.5` ausgibt. Das ist aber nicht der Fall, da es sich um eine ganzzahlige Division handelt, die auch als Ergebnis nur ganzzahlige Werte liefert. Es handelt sich im wesentlichen um Division in der Grundschule vor dem Lernen von Brüchen, dann gilt: `5 / 2 = 2 Rest 1` . Daher gibt es für ganzzahlige Arithmetik auch zwei Divisionsoperatoren: `/` für das Ergebniss der eigentlichen Division und `%` für den Rest der Division. Dieser Fallstrick ist insbesondere zu beachten, wenn Ihnen der Typ der Werte nicht bekannt ist, da Sie zum beispiel in Variablen gespeichert sind. Dann müssen Sie bei jeder Division aufpassen, dass Sie entweder mit ganzen zahlen rechnen, oder eine manuelle Typkonvertierung vornehmen. Denn das Ergebnis ist dann ebenfalls eine gebrochene Zahl, wie im Beispiel `puts 5 / 2.0` . Eine Methode mit Typkonvertierungen kann dann zum Beispiel so aussehen:

```
def div(a, b)
  a / b.to_f
end

def div2(a, b)
  a / Float(b)
end
```

8.2 Floating Point Numbers

Eine Gleitkommazahl wird auf Computern mit einer begrenzten Anzahl an Speicher binär dargestellt. In Ruby sind das typischerweise 64 Bit. Für die Ausführung des Programms ist es also notwendig, dass Ruby die Werte, die wir geschrieben haben (typischerweise im Dezimalsystem) in das Binärsystem umrechnet. Bei diesem Vorgang kann es zu Rundungsfehlern kommen, sodass Gleitkommazahlen nicht beliebig genau sind. Dadurch kann es beim Prüfen auf Gleichheit zu Problemen kommen:

```
if 2.2 - 1.2 == 1.0
  puts "Gleich."
else
  puts "Ungleich."
end
```

Man erwartet die Ausgabe "Gleich.", doch leider wird man enttäuscht. Ändert man dagegen den Vergleich zu `if 2.0 - 1.0 == 1.0` , funktioniert alles. Dieser Verlust an Genauigkeit hat mehrere Implikationen. Zum einen begründet sich nun, warum es überhaupt die ganzzahlige Division gibt, denn das Ergebnis der ganzzahligen Division ist exakt, wenn man bedenkt, den Rest zu beachten. Bei Gleitkommazahlen kann es hingegen zu Problemen kommen. Zweitens ist es nicht sinnvoll und manchmal grob fahrlässig, Gleitkommazahlen zu verwenden, wenn es auf Genauigkeit ankommt. Ein typisches Beispiel ist das Rechnen mit Geldbeträgen, denn obiges Skript könnte auch zur Berechnung des Kontostandes dienen, aber das Ergebnis wäre nicht, dass der Kunde noch einen Euro hat, sondern eben nur ungefähr einen Euro, bei einem kleinen Fehler. Bei Geldbeträgen eignet es sich daher viel mehr, mit ganzen Zahlen und der jeweils kleinsten Geldeinheit zu rechnen. Drittens ist der Vergleich von zwei Gleitkommazahlen auf Gleichheit offensichtlich nicht sinnvoll, da er sich je nach interner Darstellung der Zahlen (die wir nicht kennen und die sich ändern kann) zu anderen Ergebnissen kommt. Lediglich der Vergleich, ob zwei Gleitkommazahlen innerhalb eines bestimmten Bereiches dicht aneinander liegen ist möglich:

```
def equal_float?(a, b, accuracy=0.01)
  avg = (a + b) / 2.0
  dif = a - b
  #Absolutwert von dif benutzen
  dif = 0 - dif if dif < 0
  dif / avg < accuracy
end

puts equal_float?(2.0,2.1)              #=> False
puts equal_float?(100.0,100.1)          #=> True
puts equal_float?(100.0,100.1,0.00001)  #=> False
```

Dieses Skript vergleicht zwei Gleitkommazahlen darauf, ob der Quotient von Differenz und Mittelwert unterhalb einer Schranke liegt. Durch Einfügen der Zeile `puts fp_gleichheit?(2.2-1.2,1.0,0.00001)` erkennt man, dass der Rundungsfehler sehr klein ist und die Methode das gewünschte Ergebnis liefert. Eine Alternative bietet da die Klasse BigDecimal, die das exakte Rechnen im Dezimalsystem ermöglicht.

8.3 BigDecimal

Das äquivalente Skript mit der Klasse BigDecimal sieht wie folgt aus:

```
require 'bigdecimal'

if (BigDecimal.new("2.2") - BigDecimal("1.2")) == BigDecimal("1.0")
  puts "Gleich."
else
  puts "Ungleich."
end
```

Mit BigDecimal opfert man Kompaktheit des Codes gegen genaue Berechnungen, man sollte es also überall dort anwenden, wo man die Genauigkeit benötigt und ansonsten Gleitkommazahlen oder ganze Zahlen verwenden. Auch ist das Rechnen mit BigDecimal wesentlich langsamer als die Alternativen.

9 Arrays

Oftmals ist es nötig eine Menge gleichartiger Elemente strukturiert abrufen zu können. Stellen Sie sich zum Beispiel einen Kassenbon vor: Die Elemente haben immer die gleichen Eigenschaften, doch weiß man nicht wie viele Elemente es insgesamt gibt. Es genügt also nicht genügend viele Variablen zu erstellen, sondern greift dann auf **Arrays** zurück, die genau dieses Problem lösen. Der Zugriff auf einzelne Elemente erfolgt mit einem Index. Typische Beispiele für Arrays sind Anwendungsfälle wie: Alle Eingaben des Users bisher oder eingeloggte User einer Webseite oder alle Daten einer physikalischen Messung.

9.1 Erzeugen und Zugriff

Es ist möglich entweder ein leeres Array zu erzeugen oder es mit Werten zu initialisieren. Auch wenn eingangs erwähnt wurde, dass es sich um eine Menge gleichartiger Elemente handelt, wird dies nicht durch Ruby erzwungen. Zwar gibt es Anwendungsfälle, wo es sinnvoll ist verschiedene Typen zusammenzufassen, so werden größtenteils Arrays dann verwendet, um mehrere gleichartige Daten zu verarbeiten.

```
array = []
array = ["Batman", "Robin", "Catwoman"]
array = ["A", 1, []]
```

Der Zugriff erfolgt über einen Index, wobei die Zählung mit 0 beginnt. Das erste Element hat also den Index 0 und das letzte Element eines Arrays mit zehn Elementen den Index 9. Es existieren außerdem verschiedene Abkürzungen und Schlüsselwörter für den Zugriff.

```
array = ["Batman", "Robin", "Catwoman"]
puts array[0]
puts array.first
puts array.last
puts array[array.length-1]
puts array[-1]
```

Die Methode `liste.length` gibt die Position nach dem letzten Element des Arrays zurück. Sind also vorher alle Positionen belegt, so gibt die Methode die Anzahl der gespeicherten Elemente aus. Wie gesehen, ist es möglich Berechnungen auszuführen und das Ergebnis als Index zu verwenden. Die letzte Zeile ist die abkürzende Schreibweise für die Zeile davor.

Es ist Möglich einen Teilbereich des Arrays zu adressieren, wobei der Rückgabewert dieser Methode wieder ein Array ist.

```
puts array[1,2]
```

9.2 Arrays verändern

Nicht nur das Zugreifen auf ein Element des Arrays erfolgt mit Indizes, sondern auch die Manipulation einzelner Elemente. Das Anfügen eines Elements kann auf zwei Arten gelöst werden, zum einen durch die Adressierung des nächsten Elementes oder durch die Methode << .

```
array[0] = "Superman"
puts array[0]
array[array.length] = "Superman"
array << "Hulk"
```

9.3 Mehrere Rückgabewerte einer Methode

Mit Arrays ist es Möglich das eine Methode scheinbar mehrere Rückgabewerte hat. Sie hat natürlich trotzdem nur einen Rückgabewerte, nämlich das Array, aber Ruby erlaubt es durch paralelle Zuweisung Array quasi unsichtbar auf Variablen aufzuteilen. Das ist natürlich auch bei Arrays möglich, die nicht aus Methoden stammen.

```
def whole_number_division(a, b)
  [a / b, a % b]
end

_, rest = whole_number_division(5, 2)
puts rest
```

9.4 Kassenbon

Das Eingangs erwähnte Kassenbonbeispiel soll nun in vereinfachter Form gelöst werden. Der Kassierer soll die Möglichkeit haben die Preise der einzelnen Waren einzutippen und zum Schluss soll die Summe angezeigt werden.

```
# kasse.rb

#Endlosschleife erzeugen
loop do
  puts "Neuer Bon:"

  #Kassenbon anlegen
  prices = []
  input = ""
  #Preiseingabe beenden durch leeren Preis
  until input == "\n"
    input = gets
    prices << input.to_f unless input == "\n"
  end

  #Summe errechnen
  sum = 0.0
  for x in prices
    sum += x
  end

  #Wenn der Kassenbon leer war, Programm beenden
```

```
  puts "Ihr Einkauf kostet:\t #{ sum.to_s } EUR" unless summe == 0.0
  break if summe == 0.0
end
```

10 Hashes

Hashes sind den vorher besprochenen Arrays sehr ähnlich. Sie dienen ebenfalls dazu auf verschiedene Daten strukturiert zugreifen zu können. Unterschiede zeigen sich in Ordnung und Indizierung. Die Ordnung eines Hashes entsteht durch die Reihenfolge des Hinzufügens, während es bei einem Array kein Problem ist das vierte vor dem dritten Element einzufügen. Arrays werden außerdem über Zahlen Indiziert, Hashes über beliebige Objekte. Das ist insbesondere von Vorteil, wenn die Daten nicht gleichwertig in ihrer Bedeutung sind.

10.1 Erzeugen und Zugriff

Die häufigste Benutzung von Hashes ist die Indizierung mit Symbolen. Die Syntax ist denen von Arrays sehr ähnlich, lediglich die Erstellung mit Werten unterscheidet sich deutlich, da man gleichzeitig Index und Wert angeben muss. Das folgende Beispiel verdeutlicht den Nutzen der Indizierung mit Symbolen.

```
person = {}
person2 = {:name => "Klaus Müller", :age => 21 }
person[:name] = "Hans Schuster"
person[:age] = 20
puts "#{ person[:name] } ist #{ person[:age] } Jahre alt."
```

In Anlehnung an die Schreibweise von JSON gibt es in Ruby auch die Möglichkeit einen Hash mit Symbolen so zu erzeugen: `person2 = { name: "Klaus Müller", age: 21 }` , das ist zu der oben dargestellten Schreibweise äuqivalent. Falls Sie jedoch andere Objekte als Index verwenden müssen Sie dies wie oben gezeigt tun, z.B. `ages = { "Klaus Müller" => 21, "Hans Schuster" => 20 }`

Ein Element des Hash' bezeichnet man als Key-Value-Paar. Implementiert man die gleiche Funktionalität mit einem Array, dann würde der Zugriff wohl über `person[0]` und `person[1]` erfolgen, das ist nicht nur sehr viel schlechter lesbar, es hat auch den Nachteil deutlich weniger flexibel zu sein. Möchte man später neben Namen und Alter auch noch den Geburtsnamen speichern, dann müsste man entweder sich merken, dass der Name an Stelle 0 und der Geburtsname an der Stelle 2 gespeichert ist, oder alles um eins verschieben, wodurch jedoch der sämtliche Programmcode angepasst werden muss. Bei Hashes fügt man einen neuen Index an und der restliche Code bleibt unberührt.

10.2 Rückblick: Kassenbon

Das Kassenbonprogramm aus dem letzten Kapitel hatte einen großen Nachteil: eine Produktangabe fehlte. Mit einem Hash sind wir in der Lage dieses Manko zu beseitigen.

```
# kasse.rb

# Endlosschleife erzeugen
loop do
  puts "Neuer Bon:"

  #Kassenbon anlegen
  prices = {}
  product = ""
  price = ""
  #Preiseingabe beenden durch leeres Produkt
  until product == "\n"
    print "Produkt:\t"
    product = gets
    print "Preis:\t"
    price = gets.to_f
    prices[product.chomp] = price unless product == "\n"
  end

  #Summe errechnen und Bon ausgeben
  puts "Ihr Einkauf:"
  sum = 0.0
  for name, price in prices
    sum += price
    puts "#{ name }:\t#{ price } EUR"
  end

  #Wenn der Kassenbon leer war, Programm beenden
  puts "Ihr Einkauf kostet:\t#{ sum } EUR" unless sum == 0.0
  break if sum == 0.0
end
```

Das Programm erwartet die abwechselnde Eingabe von Produktnamen und -preis und wird ähnlich wie im vorhergehenden Beispiel durch leere Eingaben beendet.

- Zeile 17 `.chomp` ist eine Methode die aus einem String das letzte Zeichen entfernt, in diesem Fall ist es das Newlinezeichen `\n` . Dies erfolgt, damit der Bon korrekt ausgeben wird.

10.3 Sets

Eine weitere ähnliche Form der Aufbereitung von Daten sind **Sets** . Sie füllen gewissermaßen eine Lücke zwischen Hashes und Arrays. Sets sind ungeordnet wie Hashes und ermöglichen überhaupt keinen Zugriff auf ein einzelnes Element. Sie sind dabei einer Menge im mathematischen Sinne am ähnlichsten.

```
require 'set'
s = Set.new
s << 1
s << 2
s.inspect
s << 2
s.inspect
```

Die Funktion `.inspect` ist eine spezielle Methode, die auf allen Standardobjekten definiert ist. Die einzelne Funktionalität kann dabei variieren, sie soll jedoch detaillierte Informationen über das vorhergehende Objekt ausgeben. In diesem Fall wird ausgegeben, dass es sich um ein Set handelt und welche Elemente es enthält.

11 Dateien

Beim Zugriff auf Dateien stellt Ruby einen manchmal vor Probleme. Zum einen liefert es ein sehr elegantes Interface für das einfache Lesen und Schreiben, bei spezielleren Anwendungen befindet man sich näher an den zugrunde liegenden C-Bibliotheken, die nicht mehr den Konventionen von Ruby folgen. Außerdem ist es an dieser Stelle nicht mehr möglich, ohne die Vorwegnahme von Teilen der Objektorientierung auszukommen.

11.1 Ausgabe des eigenen Quelltextes

Eine einfache Aufgabe eines Programms im Umgang mit Dateien ist es, den eigenen Quelltext auszugeben. Es muss sich also selber öffnen, lesen, ausgeben und wieder schließen. Die besondere globale Variable `$0` oder `$PROGRAM_NAME` enthält den Skriptnamen.

```
source = open($PROGRAM_NAME, "r")
puts source.read
source.close
```

Dateien muss man sich anders vorstellen, als bisherige Datentypen. Sie sind sogenannte Streams, das heißt beim Öffnen einer Datei wird ein Stream angelegt, auf den danach zugegriffen werden kann. Erst beim Lesen mit `read` oder `gets` wird der Stream in einen String umgewandelt. Diese Eigenschaft von Streams zeigt sich insbesondere, wenn man mehrmalig aus einer Datei lesen möchte. Ergänzen Sie das Skript um eine zweite Zeile `puts source.read` , dann werden Sie feststellen, dass die neue Zeile zwar in der Ausgabe vorkommt, das gesamte Skript jedoch nur einmal ausgegeben wird. Der Stream wird nämlich ausgelesen und ist danach leer. Vor dem zweiten Auslesen muss der Stream zurückgesetzt werden, dies erfolgt mit der Methode `source.rewind` .

Schließlich soll noch die folgende Syntax kurz erläutert werden.

```
source.read
```

Die Methode `read` wirkt dabei auf die davor stehen Variable und liest sie in dem Fall aus. Diese Form von Methoden traten bereits früher in Form von `Hash#each` und `String#+` auf. Das ist ein Vorgriff auf den folgenden Teil über Objektorientierung. Man kann sich vorstellen, dass es sich um eine Methode handelt, deren erstes Argument die Variable `source` ist.

11.2 Rechte

Der zweite Parameter der Methode `open` legt die Zugriffsrechte auf den Stream fest.

Parameter	Erklärung
`"r"`	Default: Erlaubt das Lesen des Streams.
`"w"`	Erlaubt das Schreiben des Streams. Beim Schreiben wird der alte Inhalt überschrieben.
`"a"`	Erlaubt ebenfalls das Schreiben. Der Inhalt wird jedoch angehängt.

Das Schreiben in eine Datei erfolgt mit `File#puts` .

11.3 Auflistung eines Verzeichnis

```
dir = Dir.open("./")
dir.each do |file_or_directory|
  puts file_or_directory
end

dir.close
```

Jedes Verzeichnis enthält die Verzeichnisse `.` und `..` . Sie stehen für das aktuelle Verzeichnis und das übergeordnete Verzeichnis. Beim Öffnen eines Verzeichnisses wird im Gegensatz zu einer Datei ein Array aus Dateinamen erzeugt. Daher ist es eventuell nötig vor dem Öffnen zu prüfen, ob eine Datei existiert und ob es sich nicht um ein Verzeichnis handelt.

11.4 Verzeichnisse und Dateien

Bei Verzeichnispfaden verwendet Ruby als Separator zwischen Verzeichnis und Unterverzeichnis den Schrägstrich `/` unabhängig vom eingesetzten Separator des Betriebssystems.

Zum Überprüfen, ob eine Datei vorhanden ist, dient die Methode `File#exist?` . Dabei wird nicht zwischen Verzeichnissen und Dateien unterschieden, dafür dient die Methode `File#file?` .

11.5 DATA

Ein Quelltext in Ruby wird begrenzt durch das Ende der Datei oder der Zeile `__END__` . Es ist möglich danach Daten zu hinterlegen. In der Konstante `DATA` ist eine Datei hinterlegt, sodass der Lesezeiger auf die erste Zeile nach `__END__` steht. Dadurch ist es möglich in kleinen Skripten Daten direkt in den Quelltext einzubauen und sehr einfach zu lesen.

```
puts DATA.read

__END__
Hello World!
```

Dadurch, dass es sich um eine ganz normale Datei handelt, der eigene Quelltext, ist es auch möglich die üblichen Dateioperationen darauf auszuführen. Dadurch ist es wie folgt Möglich den Quelltext auf etwas kryptische Art und Weise auszugeben.

```
DATA.rewind
puts DATA.read
__END__
```

Die Konstante `DATA` wird nur beim Vorhandensein von `__END__` definiert, so ist die letzte Zeile im obigen Skript zwingend erforderlich.

12 Regular Expressions

Regular Expressions, oder reguläre Ausdrücke, sind eine Beschreibungssprache für Strings. Sie erlauben bestimmte Muster in Zeichenketten zu finden, zu ersetzen oder einen String auf seine Gültigkeit zu überprüfen. Eine komplette Beschreibung regulärer Ausdrücke füllt eigene Bücher, daher soll es in diesem Kapitel um einführende Beispiele und einen groben Überblick gehen.

12.1 Initialisierung und einfache Beispiele

Regular Expressions werden zwischen zwei `/` definiert. Möchte man ein Slash innerhalb des regulären Ausdrucks verwenden, so ist es nötig diesen durch eine Escapesequenz (zum Beispiel `/\//`) darzustellen. Dabei wird das `/` durch ein vorangestelltes `\` maskiert. Es stehen einige Vergleichsoperatoren zur Verfügung, die in ihrer Bedeutung an denen von Zahlen angelehnt sind.

```
s = "Hallo Welt!"
if s =~ /Hallo/
  puts s + " enthält Hallo"
end
if s !~ /hallo/
  puts s + " enthält kein hallo"
end
```

Regular Expressions unterscheiden Groß- und Kleinschreibung und es gibt einen Unterschied zwischen den Rückgabewerten der beiden Operatoren `=~` und `!~` , der im obigen Skript nicht deutlich wird. `!~` prüft ob der reguläre Ausdruck zu dem String passt oder nicht und gibt entsprechend `true` oder `false` zurück. `=~` prüft, ob der Ausdruck passt und gibt die erste passende Position im String zurück, sie ist also im Allgemeinen nützlicher als `!~` .

12.2 Zeichenklassen

Eine erste Form der Abstraktion regulärer Ausdrücke ist es, Zeichenklassen zu definieren. Möchte man Beispielsweise prüfen, ob ein String Ziffern enthält, dann ist es unzweckmäßig, auf jede einzelne Ziffer zu prüfen, sondern kann dies wie folgt tun:

```
s = "abc0123"
if s =~ /[0-9]/
  puts "Enthält Ziffern"
end
```

Zeichenklassen werden mit eckigen Klammern begrenzt und es ist möglich, Bereiche von Ziffern und Buchstaben durch - anzugeben. Wenn der Bindestrich das erste Zeichen der

Klasse ist, so wird es nicht als Sonderzeichen interpretiert. Der reguläre Ausdruck beschreibt eine beliebige Ziffer zwischen 0 und 9. Zeichenklassen ersetzen in ihrer unmodifizierten Form nur ein Zeichen. Möchte man zum Beispiel prüfen, ob ein String nur Ziffern enthält, braucht man zusätzliche Sonderzeichen.

12.3 Sonderzeichen

Sonderzeichen sind Zeichen innerhalb eines regulären Ausdrucks, welche die Bedeutung der anderen Zeichen verändern. Die Wichtigsten von ihnen sind in der folgenden Tabelle erklärt. Diese Tabelle ist natürlich nicht vollständig, insbesondere fehlen zahlreiche Escapesequenzen, diese ersetzen unter anderem Zeichenklassen (`/\d/` für Ziffern) oder erfüllen andere Aufgaben (`/\b/` für Wortgrenze).

Zeichen	Erklärung
`+`	Das vorhergehende Zeichen kommt mindestens einmal aber belieig oft vor.
`*`	Das vorhergehende Zeichen kommt beliebig oft vor.
`?`	Das vorhergehende zeichen ist optional.
`^`	Verneinung der nachfolgenden Zeichenklasse
`.`	Beliebiges Zeichen
`^`	Zeilenanfang
`$`	Zeilenende

Wenn Sie bereits mit regulären Ausdrücken vertraut sind, dann wird Ihnen die Benutzung von `^` und `$` als Zeilenanfang und -beginn eventuell komisch vorkommen. In Ruby ist der Multilinemode von regulären Ausdrücken standardmäßig aktiviert. Wenn Sie sicher gehen wollen, dass ein regulärer Ausdruck tatsächlich auf den gesamten String zutrifft, dann können die Escapesequenzen `\A` und `\z` angewendet werden.

Daneben gibt es noch zahlreiche andere Escapesequenzen, deren Aufzählung unübersichtlich und nicht zielführend ist. Es soll daher auf entsprechende Spezialliteratur verwiesen sein.

12.4 Die Variablen $1, $2, ...

Betrachten Sie einmal folgenden Sourcecode:

```
s = "Hallo"
if s =~ /(H|h)allo/
  puts "#{s} ist kleingeschrieben" if $1 == "h"
  puts "#{s} ist grossgeschrieben" if $1 == "H"
end
```

Drei neue Dinge wurden hier eingefügt. Das erste ist die Alternation `/(H|h)/` . Sie trifft zu, wenn einer der beiden Teilausdrücke zutrifft (wie das logische `||`), dabei können die Teilausdrücke wieder selbst beliebig komplexe reguläre Ausdrücke sein. Die Klammern dienen hier zum einen zur Begrenzung der Alternation und zum anderen sind dies sogenannte einfangende Klammern. Das bedeutet, dass der entsprechende Treffer in den besonderen

Variablen $1, $2, usw. gespeichert wird, je nachdem wie viele einfangende Klammern es gibt.

12.5 Named Groups

Reguläre Ausdrücke können sehr schnell unübersichtlich werden und auch die Verwendung der pseudoglobalen Variablen $1, $2, etc. trägt nicht zu einer besonderlich hohen Lesbarkeit des Programms bei.

In Ruby 1.9 wurde daher den regulären Ausdrücken eine Möglichkeit hinzugefügt, dass man einfangende Klammern bennent und somit sowohl Teile des Ausdrucks besser lesbar macht, als auch die Treffer in Variablen speichert.

Im folgenden Beispiel werden Usernamen eines Programms nach folgendem Schema in einem Array als String gespeichert: `id: name` . Zum Extrahieren der beiden Daten aus dem String wird ein regulärer Ausdruck benutzt und das Ergebnis ausgegeben.

```
users = ["1: Paul", "2: Max"]

r = /(?<id>\d+): (?<name>\w+)/i

users.each do |u|
  result = r.match(u)
  puts "#{ result[:id] }. #{ result[:name] }"
end
```

12.6 Längenkonverter

Mit regulären Ausdrücken ist es einfach, Benutzereingaben zu analysieren und zu verarbeiten. In dieser Aufgabe wollen wir ein Programm entwickeln, das es ermöglicht, zwischen Zoll und Zentimetern umzurechnen. Während die eigentliche Umrechnung (1 Zoll sind 2,54 Zentimeter) kein Problem ist, stellt die Interaktion mit dem Benutzer das größere Problem dar, welches sich mit regulären Ausdrücken dann wiederum sehr schön implementieren lässt.

Die Benutzereingabe soll dem folgenden Format entsprechen: Zunächst soll der Wert (wenn Dezimalzeichen, dann als Punkt) eingegeben werden, der durch beliebig viele Whitespaces von der Einheit getrennt wird (Zum Beispiel 0.3 I, oder 32cm).

In regulären Ausdrücken bedeutet das:

- `/^\d+/` Nach dem Stringanfang kommen beliebig viele Ziffern, mindestens jedoch eine
- `/\.?\d*/` Optional folgt dann ein Punkt gefolgt von beliebig vielen Ziffern
- `/\s*/` Die Escapesequenz `\s` steht für beliebigen Whitespace, also neben Leerzeichen auch Tabulatoren, Zeilenumbrüche, ...
- `/(i|cm)/i` Auswahl der Einheit als entweder i für Inch oder cm für Zentimeter. Der Parameter /i verändert die Regexp, sodass sie unabhängig von Groß- und Kleinschreibung zutrifft.

Der gesamte Ausdruck inklusive benannten Klammern lautet: `/^(?<val>\d+\.?\d*)\s*(?<unit>(i|cm))/i` . Und der dazugehörige Quelltext:

```
#Encoding: utf-8

r = /^(?<val>\d+\.?\d*)\s*(?<unit>(i|cm))/i

loop do
  puts "Bitte geben Sie eine Länge ein.\nEine leere Eingabe beendet das
 Programm."
  input = gets.chomp
  break if input.empty?

  if (result = r.match(input))
    if result[:unit] =~ /i/i
      inch = result[:val].to_f
      sicm = result[:val].to_f/2.54
    else
      sicm = result[:val].to_f
      inch = result[:val].to_f*2.54
    end
    puts "#{ sicm } cm = #{ inch } I"
  else
    puts "#{ input } ist eine fehlerhafte Benutzung dieses Programms."
  end
end
```

13 Kommandozeilenintegration

Auch wenn das Arbeiten auf der Kommandozeile in den Hintergrund gerückt ist, so macht es dennoch für viele kleine Anwendungen Sinn mit den Beschränkungen, aber auch der Einfachheit der Kommandozeile zu leben. Die Programmierung von User Interfaces ist aufwendig und man stößt auf viele Tücken. Was auf der Kommandozeile ein einfaches `gets` ist, ist bei UIs das Handhaben von Tastatureingaben, Mausbewegungen und so weiter. Insbesondere auf Linuxsystemen ist das Arbeiten mit Kommandozeilenprogrammen weiter verbreitet, was man schon daran sieht, dass man ein `skript.rb` nicht via Doppelklick ausführt, sondern man auf der Kommandozeile den Rubyinterpreter aufruft.

13.1 Kommandozeilenparameter

Wenn man sich genauer Gedanken darüber macht, was die Zeile `ruby skript.rb` bedeutet, dann gelangt man zu der Erkenntnis, dass das Ruby in Abhängigkeit von den nachfolgenden Zeichen unterschiedliche Funktionen erfüllt. Entweder interpretiert es ein Skript, gibt eine Hilfe aus (-h) oder die Version (-v). Dieses Verhalten erinnert stark an Methodenparameter, bei denen wir Fallunterscheidungen bei Parametern schon kennengelernt haben und ähnlich funktionieren auch **Kommandozeilenparameter** . Die Parameter werden auf der Kommandozeile mit Leerzeichen voneinander getrennt und sind im Skript im konstanten Array `ARGV` verfügbar.

```
puts ARGV.length
----
```

Das Array enthält dabei immer Strings, gegebenenfalls ist also eine Typumwandlung nötig.

13.2 Zur Kommandozeile zurück

Neben dem Erhalten von Parametern gibt es die Möglichkeit, das System direkt aus Ruby anzusprechen. Dazu gibt es zwei unterschiedliche Möglichkeiten: Der Aufruf der Methode `system` erhält als Parameter einen String, der ausgeführt wird, als ob er sich auf der Kommandozeile befände. `system` erzeugt dazu einen neuen Prozess, daher ist es nicht ohne weiteres möglich auf das Ergebnis dieses Systemcalls zuzugreifen.

Möchte man das tun, ist es möglich einen Systemaufruf mit `` `...` `` oder `%x(...)` auszuführen. Beide Möglichkeiten geben die Standardausgabe des ausgeführten Codes als String zurück, daher ist möglich diesen im eigenen Programm weiter zu verwenden. Dadurch ist es sehr einfach andere Programme um Funktionalität zu erweitern, oder eine grafische Oberfläche für ein Kommandozeilentool zu entwickeln.

```
Dir.open(".").each do |f|
  puts f if f =~ /\.rb/
end

puts

puts `ls *.rb`
```

Die beiden Programmteile sollen alle Rubyskripte im aktuellen Verzeichnis anzeigen, jedoch ist es durch die Verwendung von `` `ls *.rb` `` wesentlich leichter zu lesen, falls man sich bei der Bedienung einer POSIX-Shell auskennt. Im Zweifel sollten sie auf diese Methode nur zurückgreifen, falls sie explizit mit einem Programm arbeiten wollen und ansonsten sich auf reinen Rubycode beschränken. Das oben gezeigte Beispiel ist also schlechter Code, da er viele Voraussetzungen an andere Entwickler stellt - Sie müssen eine POSIX-Shell verwenden und mit dieser vertraut sein.

Eine bessere Verwendung zeigt sich in vielen Gemfiles von Entwicklern, die den Quellcode ihrer Gems mit Git verwalten. Gem und Gemfiles werden im Kapitel Rubygems näher erklärt, an dieser Stelle nur so viel: In einem Gemfile (Rubyquelltext) müssen alle Files stehen, die zu einem Gem gehören. Der Befehl `git ls-files` zeigt alle Dateien an, die aktuell mit Git verwaltet werden, dadurch ist es sehr einfach in Gemfiles durch die Zeile `` `git ls-files`.split("\n") `` alle verwalteten Dateien im Gemfile anzuzeigen.

13.3 Fehlerbehandlung

Die Kommandozeile erlaubt bestimmte Fähigkeiten im Umgang mit Fehlern eines Programmes, wenn es sich nach bestimmten Konventionen verhält. Das folgende Skript berechnet das Quadrat des Arguments und gibt es aus. Außerdem behandelt es den Fehler, das keine Zahl übergeben wurde:

```
number = ARGV[0]

if number.nil?
  $stderr.puts "This program expects a number to calculate it's square."
  exit 1
end

puts number.to_f ** 2
```

Falls Sie dieses Programmausführen sollten Sie keine Unterschiede zu bisherigen Skripten erkennen können unabhängig, ob Sie das Program korrekt oder Fehlerhaft benutzen. Die Unterschiede zeigen sich erst, wenn man die Kommandozeile benutzt, um das Programm in einen größeren Kontext einzubauen.

Es ist Möglich die Ausgabe eines Programmes umzulenken. Das kann dazu benutzt werden die Ausgaben zu ignorieren oder in einer Datei zu speichern. Es stehen dabei zwei unterschiedliche Ausgabewege zur Verfügung, die erste und der default, der mit `puts` angesprochen wird, heißt `$stdout` für das Anzeigen von Fehlern gibt es `$stderr` . Wenn Sie das Programm über die Kommandozeile mit `ruby square.rb 2> /dev/null` aufrufen, dann erhält unser Skript keine Argumente, da `2> /dev/null` von der Kommandozeile verarbeitet wird, also entsteht der Fehler. Er wird aber nicht angezeigt, weil wir sagen, dass alle Meldungen im Zusammenhang mit Fehlern nach `/dev/null` schreiben, einen speziellen Ort

auf Unixsystemen, der im wesentlichen wie ein Mülleimer funktioniert. Das Umleiten der Hauptausgabe erfolgt mit `ruby square.rb 2 > result` .

Daneben ist es manchmal nötig mehrere Programme zur Lösung eines Problems zu verwenden, wobei jedes Program einen Teil löst, aber nicht zwangsläufig funktioniert. Das nicht funktionieren eines Rubyskripts wird mit `exit` und einem Fehlercode, der nicht 0 sein darf, angezeigt. Dann wird bei folgendem Aufruf `ruby square.rb && echo "Hello"` das Wort Hello nicht ausgegeben, weil das zweite Program `echo` nur ausgeführt wird, wenn unser Skript funktioniert hätte.

14 Klassen

Typischerweise erfolgt eine Einführung in objektorientierte Programmierung mittels einem Beispiels anhand dessen man die sowohl die Eigenarten der Objekte, als auch deren Umsetzung in einer bestimmten Sprache kennenlernen soll. Dieses Kapitel folgt diesem Konzept mit einer kleinen Abwandlung: Vor einem solchen Beispiel soll Objektorientierung motiviert werden, denn Sie könnten die berichtigte Frage stellen, wozu ein solches Konzept nötig ist, wenn Sie mit den in den Grundlagen beschriebenen Konzepten vollständig programmieren können.

14.1 Motivation

Diese Einleitung dient vor allem Neulingen beim Verständnis, wenn Sie bereits eine andere Objektorientierte Sprache beherrschen können Sie diesen Abschnitt überspringen.

Kehren wir einmal zurück zu dem Beispiel aus den Grundlagen mit dem Kassenbon. Dieses zugegebenermaßen nicht besonders ausgefeilte Beispiel, soll uns als Grundlage dienen. Das Produkt hatte einen Namen und einen Preis. Diese Daten korrelieren offensichtlich für ein einzelnes Produkt, aber zwischen den Produkten besteht kein Zusammenhang. Realisiert wurde dies durch die Verwendung eines Hashes. Stellen Sie sich jedoch einmal vor, dass mehr Daten zusammenhängen würden. Sie müssten dann innerhalb des Hashes wieder ein Hash oder ein Array verwenden und sich zusätzlich der Struktur ihres Datentyps an jeder Stelle im Programm bewusst sein.

Kapselung ist die erste wichtige Eigenschaft von Objekten, es bedeutet dass innerhalb eines Objektes Daten vorliegen, die über dieses objekt miteinander in Verbindung stehen. In unserem Beispiel also Name und Preis. In den meisten Umsetzungen von Objekten werden jedoch nicht nur Daten gekapselt, sondern auch Methoden. Dies hat den Vorteil, dass ein `String#+` etwas völlig anderes bedeuten kann als `Array#+` , weil es sich um Methoden des Objektes String bzw. Array handelt.

Nun zurück zur Praxis:

14.2 Klassen

In der Motivation wurde das Wort Klasse nicht erwähnt, weil es für das Konzept von Objekten nicht wichtig ist. Es gibt objektorientierte Sprachen, die völlig ohne ein Klassensystem auskommen. Klassen sind in erster Linie ein Bauplan bzw. eine Beschreibung des Objekts, während eine konkrete Umsetzung oder auch **Instanz** genannt ein Objekt ist.

Klassennamen müssen gemäß Sprachdefinition mit Großbuchstaben anfangen. Es ist üblich, Klassennamen im CamelCase zu schreiben. Man verzichtet auf die Verwendung von Unterstrichen und benutzt stattdessen Großbuchstaben jeweils am Anfang eines neuen Wortes oder Wortteils, z.B. `Drink` oder `LongDrink` .

Die Definition einer Klasse erfolgt mit dem Schlüsselwort `class` und wird wie üblich mit `end` beendet. Eine minimale Klasse entspricht folgendem Schema:

```
class KlassenName
  def initialize
  end
end
```

Diese Klasse hat einen Konstruktor `KlassenName.new` . Dieser ist dafür verantwortlich ein neues Object zu erstellen. Dafür sind zwei Schritte notwendig: Zum einen das Anfordern des Speicherplatzes vom Betriebssystem und zum anderen die spezielle Initialisierung des Objekts auf seinen Anfangszustand. Da dies immer der Fall ist, muss nicht jedesmal `KlassenName.new` implementiert werden, sondern diese Methode ruft ihrerseits die oben geschriebene Methode `initialize` auf.

14.3 Instanzmethoden und -variablen

Die oben beschriebene Klasse erfüllt nur begrenzten Sinn, da man zwar von ihr Objekte erzeugen kann, diese aber nichts können. Möchte man also Daten und Funktionalität hinzufügen, so tut man dies mit Instanzmethoden und -variablen. Diese beziehen sich auf ein konkretes Objekt, d.h. dass sich Instanzvariablen also innerhalb Objekten der selben Klasse unterscheiden können und Instanzmethoden erst nach Konstruktion eines Objektes aufgerufen werden können.

Die erste Instanzmethoden ist `initialize` . Sie werden ganz normal mit `def` eingeleitet und mit `end` abgeschlossen. Der einzige Unterschied zu normalen Methoden besteht also darin, dass sie in einer Klassendefinition stehen.

Instanzvariablen beginen mit einem `@` -Zeichen. Bedienen wir uns wieder eines Beispiels: Angenommen Sie entwickeln ein Text-RPG und wollen nun ein Schwert hinzufügen. Es hat eine Beschreibung und einen Namen und man kann damit zuhauen.

```
class Sword
  def initialize(name,description)
    @name = name
    @description = description
  end

  def hit
    puts "The mighty sword #{@name} hits!"
  end
end
```

Alle Instanzvariablen sind nicht von außen sichtbar, dass bedeutet im vorrangegangenen Beispiel weiß zwar das Objekt die Beschreibung des Schwertes, jedoch kann man nicht von außen darauf zugreifen. Alle Veränderungen und Ausgaben des Objektes erfolgen über Instanzmethoden! Es ist Möglich diese manuell zu erstellen indem man zum Beispiel folgende Methoden hinzufügt:

```
class Sword
  def name
    @name
  end

  def name=(name)
    @name = name
  end
end
```

Oder die Methoden `attr_reader` , `attr_writer` oder `attr_accessor` verwenden um jeweils das Lesen, Schreiben oder beides eines Symbols zu ermöglichen. Im Obigen Fall heißt das also:

```
class Sword
  attr_accessor :name, :description
end
```

14.3.1 Attribute von Instanzmethoden

Wie oben erläutert handelt es sich bei den Instanzvariablen grundsätzlich um private Eigenschaften des Objektes, das bedeutet das niemand direkt auf sie zugreifen kann, sondern dies durch die Verwendung von Methoden geschieht (wenn überhaupt). Auch Instanzmethoden kann man auf diese weise vor einem Zugriff schützen. Der Standard ist `public` , dass heißt Instanzmethoden können von überall im Programm aufgerufen werden. Dazu gibt es `private` , d.h. die Methode kann nur innerhalb des Objektes aufgerufen werden und `protected` , d.h. die Methode kann innerhalb aller Objekte der gleichen Klasse und deren Erben aufgerufen werden.

14.4 Klassenmethoden und -variablen

Klassenmethoden wie zum Beispiel `File.delete` können ohne die Erzeugung eines Objekts aufgerufen werden. Beim Beispiel des Files ist die Notwendigkeit offensichtlich: die Erzeugung eines Fileobjekts würde das Öffnen der Datei vorraussetzen, jedoch können geöffnete Dateien nichtmehr gelöscht werden. Die häufigste Verwendung einer Klassenmethode ist der Konstruktor `.new` .

Für die Definition von Klassenmethoden existieren zwei mögliche Schreibweisen, die man Abhängig von der Anzahl zu definierender Klassenmethoden verwendet werden sollte. Sind wenige Methoden zu definieren, dann ist `def self.method` leichter zu lesen, während bei vielen Klassenmethoden die zweite Möglichkeit weniger Code wiederholt.

```
class Example
  def self.hello(str)
    puts "Hello #{ str }!"
  end

  class << self
    def hello2(str)
      puts "And hello #{ str }!"
    end
  end
```

```
end

Example.hello("World")
Example.hello2("Wikipedia")
```

Klassenvariablen beginnen mit `@@` . In jeder Objektinstanz der Klasse haben sie den gleichen Wert und sind daher quasi globale Variablen im Kontext einer Klasse. Aufgrund des seltsamen Verhalten von Klassenvariablen bei der Verwendung von Vererbung, sollte man Sie entweder durch globale Variablen ersetzen, oder durch Umstrukturierung des Programms überflüssig machen.

15 Module

Unter Modulen ist es möglich Rubycode zu einer logischen Einheit zusammenzufassen, das kann sich je nach Anwendungszweck auf einzelne Methoden, Konstanten oder ganze Klassen beziehen. Dadurch wird es möglich diese Bündel zusammen zu verwenden und Konflikte zwischen Namen zu verhindern.

15.1 Module

Module werden wie Klassen definiert und folgen den gleichen Namenskonventionen. Die parallelen zwischen Klassen und Modulen in Bezug auf Kapselung des Codes geht noch weiter, so dass das folgende Skript die gleiche Funktionalität auch mit `class Example` haben würde. Bei der Verwendung von Modulen erreichen Sie nichts, was nicht auch durch Verwendung von Klassen möglich wäre. Die Verwendung eines Moduls sollten Sie immer dann einer Klasse vorziehen, wenn die Bildung einer Instanz keinen Sinn hat oder mehrere Klassen zusammengehören.

```
module Example
  class Hello
    def self.world
      puts "Hello World!"
    end
  end
end

Example::Hello.world
```

15.1.1 Modulfunktionen

Neben der Möglichkeit Module in Objekte einzufügen, was im Kapitel Vererbung besprochen wird, ist es nur möglich die Modulfunktionen eines Moduls zu benutzen, da Instanzmethoden eine Objektinstanz vorraussetzen. Sie unterscheiden sich genauso wie Klassen- und Instanzmethoden der Objekte aus dem vorherigen Kapitel.

Das folgende Listing definiert mehrere Methoden für das Modul. Der Zugriff auf alle Methoden erfolgt gleich mit `Example::METHOD` .

```
module Example
  def one
    puts "module_function"
  end
  module_function :one

  def Example::two
    puts "Example::two"
```

```
  end

  def self.three
    puts "self.three"
  end
end
```

Wenn keine Namenskonflikte vorliegen, kann man ein Module in den aktuellen Namensraum einfügen durch die Verwendung `include` . Mit dem oben definierten Module also so:

```
include Example
one
```

Wenn vorher eine Methode exisitert, die mit dem `include` in Konflikt steht, wird die Methode nicht überschrieben und die Modulmethode ist weiterhin nur über `Example::one` aufrufbar.

15.2 Namensraumoperator

Der Operator `::` beschreibt der Maschine, wie sie einen bestimmten Namen suchen muss. Im obigen Beispiel mit `Example::one` wird also bestimmt, dass es zuerst nötig ist, eine Klasse oder ein Modul mit dem Namen `Example` zu suchen und dann in diesem Modul die Methode mit dem Namen `one` . Dabei werden verwendete Aufrufe innerhalb dieser Methoden dann von innen nach außen gesucht, was unter bestimmten Umständen zu Problem führen kann.

```
module Example
  def self.random_string
    String.new("abc123")
  end

  class String
  end
end

puts Example::random_string
```

Das obige Skript definiert ein Modul, das eine Methode und eine Klasse enthält. Die Methode soll einen String anlegen und wird aufgerufen. Das Problem daran ist, dass beim Aufruf von `random_string` nun in den Namensräumen von innen nach außen nach einer Klasse `String` gesucht wird. Der Konstruktor der Klasse `Example::String` erhält jedoch keine Parameter, sodass das obige Skript mit einem Fehler abbricht. Es ist möglich, explizit die Stringklasse des globalen Namensraums zu verwenden durch das Präfix `::` . Das Skript beendet erwartungsgemäß durch Ändern der entsprechenden Zeile zu `::String.new("abc123")` .

16 Vererbung

Stellen Sie sich einen Baum vor. Er hat einen Stamm, Wurzeln und eine Krone. Betrachtet man nun jedoch zwei konkrete Bäume z.B. Tanne und Birke so zeigen sich Unterschiede in der Farbe des Stammes, der eine hat Blätter der andere Nadeln und so weiter. Unser Verständnis von der Welt ist offensichtlich hierarchischer Art zwar Teilen alle Bäume die Eigenschaft von Bäumen, können sich aber in ihrer konkreten Ausprägung unterscheiden.

Diese hierarchische Form bei der Beschreibung der Welt, findet man ebenfalls in objektorientierten Sprachen wieder. Durch **Vererbung** werden Eltern-Kind-Relationen zwischen den Klassen definiert, die es erlauben das oben genannte Beispiel zu modellieren.

16.1 Vererbung

Gehen wir in diesem Beispiel noch eine Stufe höher und erstellen eine Klasse `Plant` , von der wir konkrete Pflanzen, wie Bäume, ableiten wollen.

```
class Plant
  def initialize(name)
    @name = name
  end
end
```

Vererbung erfolgt mittels eines `<` . Dabei übernimmt die erbende Klasse alle Variablen und Methoden der Elternklasse.

```
class Tree < Plant
  def initialize(name)
    super(name)
    @has_trunk = true
  end
end
```

`super` ruft die gleichnamige Methode der Elternklasse auf und erlaubt es, dass sich eine Veränderung der Methoden der Elternklasse auch auf die Kinder vererbt, anstatt sie zu überschreiben.

16.1.1 Mehrfachvererbung/Mixins

Mehrfachverbung beschreibt die Möglichkeit, dass eine Klasse Eigenschaften von mehreren Elternklassen erbt. Ruby kennt keine direkte Mehrfachverbung, sondern ermöglicht ein ähnliches Verhalten mit **Mixins** .

Mixins sind Module, die zu Klassen hinzugefügt werden und so diese Klassen um Funktionalität erweitern. Sie dienen meist dazu abstrakte Methoden bereitzustellen, die in mehreren

Klassen sinnvoll angewendet werden können. Beispielsweise kann ein Baum wachsen, genauso wie ein Tier. Man könnte argumentieren, dass man ein gemeinsames Elternobjekt einfügen könnte (Lebewesen), das diese Methode bereitstellt, doch gibt es oft nicht diese einfache Möglichkeit neue Elternobjekte einzufügen.

```
module Growable
  def grow!
    @height ||= 0
    @velocity ||= 0.1
    @height += @velocity

    self
  end
end

class Tree
  attr_reader :height

  include Growable
end

puts Tree.new("Birch").grow!.grow!.height
```

In diesem Beispiel wird die Klasse `Tree` durch eine `grow!` -Methode erweitert. Dadurch ist es Möglich, den selben Code auch in anderen Klassen zu verwenden, die gegebenenfalls unterschiedliche Elternklassen haben.

Hier zeigt sich auch der Unterschied zwischen den verschiedenen Möglichkeiten des letzten Kapitels Methoden auf Modulen zu definieren. Eine mit `module_function` definierte Modulmethode wird nicht der Objektinstanz hinzugefügt, kann aber innerhalb der Klasse ohne Modulprefix aufgerufen werden, im Gegensatz zu den anderen beiden aufgezeigten Varianten, die den Aufruf nur über den Modulnamen erlauben. Instanzmethoden werden wie oben gezeigt der Objektinstanz hinzugefügt.

17 Rückblick: Grundlagen

Im ersten Teil des Buches wurden die Grundlagen für die Programmierung in Ruby dargestellt. Bis auf wenige Ausnahmen bzw. Ausblicke wurde dabei auf Objektorientierung verzichtet. Am Anfang des Buches wurde jedoch gesagt, dass Ruby eine durchgehend objektorientierte Sprache ist. Dieses Kapitel soll nun darüber Aufschluss geben, wie die Grundlagen mit der Obejktorientierung zusammenhängen und was sich daraus für Möglichkeiten und ggf. Probleme ergeben.

In anderen objektorientierten Sprachen, wie zum Beispiel Java unterscheidet man zwischen nativen Datentypen und Objekten. Eine Variable von Typ Integer ist also nicht dasselbe, wie eine Objektinstanz der Klasse Integer. Dadurch ist es nicht möglich durch einfache Initialisierung mit `int i = 2` die Möglichkeiten der Objektorientierung zu nutzen, sondern muss danach eine Objektinstanz erzeugen und mit dem Integerwert füllen. (Für dieses Problem existiert eine Kurzform, ähnlich dem in Ruby verwendeten `Fixnum.new(2)` .) Ruby verzichtet auf Datentypen und alles ist eine Instanz.

17.1 Alles, wirklich Alles

Öffnen Sie eine IRB-Session und probieren Sie `self.class` . Methoden, Zahlen usw. sind Objektinstanzen entsprechender Klassen. Unter anderem ist es dadurch auch möglich, vorhandene Klassen zu verändern. Sie können vorhandene Methoden ändern und Ihren Bedürfnissen anpassen, oder neue hinzufügen und zwar ohne auf Vererbung zurückgreifen zu müssen. Dabei ist jedoch Vorsicht geboten: Sie können Ihren kompletten Code ruinieren, wenn Sie an anderer Stelle auf eben eine gewünschte Funktionalität erwarten und diese überschreiben. Andererseits können Sie Ihren Code auch brauchbar behalten, wenn sich eine Klasse durch Ihren Code zieht und Sie eine Methode hinzufügen.

17.2 Nocheinmal: Fakultät

Diese Eigenschaft lässt sich nutzen, wenn man vorhandene Klassen verändern möchte, um beispielsweise Methoden hinzuzufügen, oder auch vorhandene verändern möchte. Das folgende Skript betrachtet nocheinmal die Fakultät, diesmal jedoch als Methode der Klasse Fixnum und durch einen geeigneten Bezeichner dargestellt.

```
class Fixnum
  def !
    self.zero? ? 1 : self * (self - 1).!
  end
end
```

```
if ARGV[0]
  puts "#{ ARGV[0] }! = #{ ARGV[0].to_i.! }"
else
  puts "Fehler."
end
```

18 Funktionale Aspekte

In der Einleitung zu diesem Buch wurde erwähnt, dass Matz versuchte, mit Ruby Aspekte der objektorientierten und der funktionalen Programmierung zusammenzuführen. Diese Seite beschäftigt sich nun mit den funktionalen Anteil in Ruby.

Dazu eine kurze Einführung: Stellen Sie sich vor, Sie schreiben eine Methode zum Sortieren von Arrays. Dann implementieren Sie einen Algorithmus und abhängig davon, welches Element als kleiner bzw. größer angesehen werden soll, steht das Größte am Schluss ganz hinten. Jetzt stellen Sie sich vor, Sie wollen die umgekehrte Reihenfolge. Oder noch schlimmer: Sie wollen statt nach Anfangsbuchstaben von Strings nach deren Länge sortieren. Sie müssen ggf. viele Funktionen schreiben, die sich alle den Algorithmus teilen. Ruby ermöglicht es, an dieser Stelle die Auswahl der Reihenfolge an den Benutzer der Methode zu übergeben und nur den Algorithmus zu schreiben. Dies geschieht dadurch, dass die Sortiermethode vom Benutzer eine Methode als Parameter erhält, die aufgerufen wird, wenn es um diese Entscheidung geht.

18.1 Blöcke

Methodenaufrufe können zusätzlich zu Parametern noch einen sogenannten **Block** entgegennehmen. Ein sehr einfaches Beispiel hierfür ist `Integer#times` , wobei eine Zahl den Block entsprechend ihrem Wert x-mal aufruft. Optional können Blöcke Parameter aufnehmen, mehrere Paramter werden dann durch Kommata getrennt. Diese werden zwischen zwei Pipe-Zeichen geschrieben:

```
3.times { |i| puts "Hallo Nummer " + i.to_s }
```

Man kann auch folgende Syntax verwenden:

```
3.times do |i|
puts "Hallo" + i.to_s
end
```

Die beiden Varianten unterscheiden sich in ihrer Bindung zum davorgehenden Objekt. So führt ein `1.upto 3 { |i| puts i }` zu einem Fehler, weil nicht die Methode `upto` den Block übergeben bekommt, sondern die Zahl 3, die dann mit dem Block nichts anfangen kann. Entweder schreiben man die 3 also in Klammern, oder verwendet `do ... end` .

18.2 Blöcke in eigenen Methoden

18.2.1 yield

Für die Verwendung von Blöcken in eigenen Methoden können diese mit dem Schlüsselwort `yield` aufgerufen werden:

```
def execute
  yield
end

execute { puts "Hallo" }
```

Wenn diese Methode ohne Block aufgerufen wird, dann kommt es zu einem Fehler, da versucht wird nicht existenten Code auszuführen. Das kann man mit der Methode `block_given?` prüfen und in den Kontrollfluss der Methode einbauen, um eventuell eine Defaultmethode auszuführen, oder einen Fehler zurückzugeben

```
def execute
  yield if block_given?
end
```

Die Übergabe von Parametern an den Block erfolgt, wie bei normalen Methoden auch in einer mit Kommas getretenten Liste hinter `yield` .

18.2.2 Der & Operator

Es ist alternativ möglich den Block in einer lokalen Variable zu speichern, um ihn beispielsweise an andere Methoden weiter zu übergeben.

```
def execute(&block)
  block.()
end
```

Dabei wird eine Instanz der Klasse `Proc` erzeugt und in der Variable `block` gespeichert. Dadurch ist möglich auch mit normalem Kontrollfluss auf die Übergabe einen Blockes zu reagieren und die Variable direkt zu manipulieren. Es ist nur Möglich einen Block an eine Methode zu übergeben. Wichtig bei der Übergabe an eine andere Methode ist das erneute Auspacken des Blockes aus der Onjektinstanz wieder mit dem `&` Operator:

```
def five_times(&block)
  5.times(&block)
end
```

18.3 Iteratoren

Die wichtigsten Methoden in diesem Zusammenhang sind Iteratoren. Sie arbeiten auf Objekten, die mehrere Objekte beinhalten, zum Beispiel Arrays oder Hashes und deren Aufgabe ist es jedes Element an einen übergebenen Codeblock zu übergeben. Das folgende Skript implementiert eine eigene Version von `Array#each` , die ähnliche Funktionalität bereitstellt.

```
class Array
  def my_each(&block)
    if block
      for i in (0..self.length)
        block.(self[i])
      end
    else
      raise ArgumentError
    end
  end
end

[1, 2, 3].my_each { |i| puts i }
```

18.4 Objektinstanzen von ausführbarem Code

Wie schon durch die Verwendung von `&` vorausgenommen, ist es möglich, Objektinstanzen von ausführbarem Code anzulegen und diese in Variablen zu speichern, zu manipulieren und an andere Methoden zu übergeben. In Ruby gibt es dafür sehr viele verschiedene Möglichkeiten, die hier vorgestellt werden sollen.

```
procs = []
procs << -> x { x + 1 }
procs << lambda { |x| x + 1 }
procs << Proc.new { |x| x + 1 }
procs << proc { |x| x + 1 }

procs.each do |p|
  begin
    puts p.class
    puts p.(1, 2)
  rescue
    puts "Falsche Parameter"
  end
end
```

Der Unterscheid zwischen den Methoden zeigt sich auch im obigen Skript. Er besteht beim Aufruf der Objekte mit einer falschen Anzahl an Parametern. Lambdaausdrücke und die in Ruby 1.9 eingeführte `->` Notation verhalten sich eher wie Methoden, während die Procmethoden nicht ihre Parameter überprüfen. Dadurch kann es zu seltsamen Fehlern bei der Verwendung von zu wenigen Parametern kommen, die schwer zu finden sind. Die ersten beiden gezeigten Möglichkeiten sind daher zu bevorzugen, da der Fehler näher am falschen Code auch entdeckt wird.

19 Threads

Mehrkernprozessoren bieten die Möglichkeit eine höhere Geschwindigkeit der Programme zu erreichen, da die Berechnungen parallel ausgeführt werden können. Diese Form der Parallelität bezeichnet man als echt, da die Berechnungen wirklich zur gleichen Zeit ausgeführt werden. So könnte man sich ein Programm vorstellen, dass auf einem großen Array eine `parallel_each` ausführt. Dies wird von MRI nicht unterstüzt. In anderen Implementationen wie JRuby und Rubinius ist es dagegen möglich mehrere Prozessoren zu nutzen.

Trotzdem ist es sinnvoll sich auch bei der Verwendung von MRI über die Benutzung von Threads Gedanken zu machen, nämlich dann, wenn es um virtuelle Parallelität handelt. Dabei werden zwar keine Operationen zeitlich nebeneinander ausgeführt, jedoch erfolgt die Berechnung und der Wechsel so schnell, dass es für den Nutzer wie eine parallele Ausführung aussieht. Zu Geschwindigkeitsvorteilen kann es dann kommen, wenn der eine genutzte Prozessor ansonsten warten würde, zum Beispiel auf Daten, die über eine Netzwerkverbindung übertragen werden.

19.1 Threads

Ein neuer Thread ist ein Objekt der Klasse `Thread` . Beim initialisieren erhält es einen Codeblock, den der neue Thread ausführt. Für die Benutzung von Variablen innerhalb des Threads ergeben sich die selben Sichtbarkeitsregeln wie für Blöcke im Allgemeinen, daher kann der Thread auf alle Variablen zugreifen, die er benutzt oder vor Aufrufen des Threads sichtbar waren.

```
i = 0
Thread.new do
  loop do
    i += 1
  end
end

gets
puts "Der Thread hat bis #{ i } gezaehlt!"
```

Dieses einfache Beispiel erzeugt eine Variable `i` und inkrementiert diese in einem neuen Thread in einer Endlosschleife. Am Ende wird die Zählung ausgegeben. Ein Thread wird entweder beendet, wenn das Ende des Codeblocks erreicht ist, oder wenn das Hauptprogramm sich beendet. Auf das Ende eines Threads kann man mit der Methode `join` warten.

```
i = 0
t = Thread.new do
  while i < 100000
    i += 1
  end
end
```

```
t.join
puts i
```

Wenn Sie die Zeile `t.join` auskommentieren, werden Sie feststellen, dass dem Skript nichtmehr die Zeit bleibt um die Zählung zu beenden.

19.2 Prozesse unter Unix mit fork

Neben Threads exisitiert auch das Konzept von Prozessen. In beiden Fällen wird Code parallel ausgeführt, allerdings besteht ein großer Unterschied was die mögliche Kommunikation der Prozesse betrifft. Zwei Threads eines Programms teilen sich einen gemeinsamen Hauptspeicherbereich, dadurch ist es möglich Daten zwischen den Threads einfach auszutauschen über Variablen, die vor dem aufrufen des Threads sichtbar waren. Prozesse laufen in unterschiedlichen Speicherbreichen und können daher nicht direkt miteinander kommunizieren. Das Betriebssystem bietet an dieser Stelle andere Möglichkeiten Daten zwischen den Prozessen auszutauschen.

Die Methode `fork` verhält sich ähnlich wie `Thread.new` .

```
puts "Before fork"
pid = fork { puts "Inside fork" }
puts "pid: #{ pid }"
```

Um den `fork` auszuführen wird der gesamte Prgrammstatus, wie zum Beispiel der Inhalt aller Variablen kopiert und erst danach das neue Programm ausgeführt. Dadurch ist es zwar möglich auf den Zustand des Hauptprogramms zuzugreifen, nicht jedoch diesen zu verändern. Ein Hauptunterschied zu Threads ist, das selbst beim Beenden des Hauptprogramms alle Forks weiter laufen und sich selber Beenden müssen.

Diese Methode kann tatsächlich auf mehreren Prozessorkernen parallel laufen, da die komplette Rubyimplementierung mitgeforkt wird, was auch deren Nachteil gegenüber Threads leicht erkennbar macht. Forks sind teuer und lohnen sich nur bei langlaufenden Anteilen im Programm, da ansonsten der Aufwand das Programm zu kopieren den Nutzen der Gleichzeitigkeit aufhebt. Threads dagegen sind recht leichtgewichtig erzeugen jedoch keine echte Parallelität.

20 Exceptions

In vielen modernen Programmiersprachen gibt es Konstrukte, die es erlauben auf Fehler bei der Programmierung des Programms hinzuweisen. Diese Fehler werden in Ruby Exceptions gennant, dabei handelt es sich um einen alternativen Programmablauf, wenn ein Fehler auftritt.

20.1 Exceptions

Exceptions werden durch das Schlüsselwort `raise` angezeigt. Die dadurch erzeugten Fehler werden im Rubyprogramm an die aufrufende Methode weitergereicht. Dadurch ist es Möglich an jeder Stelle des Programms den aufgetretenen Fehler abzufangen und ihn zu behandeln. Diese Fehlerbehandlung erfolgt mit den Schlüsselwörtern `begin` , `rescue` und `ensure` . Dabei wird versucht den Teil des Programms innerhalb des `begin` auszuführen, falls dabei ein Fehler auftritt wird der mit `rescue` eigeleitete Block ausgeführt, während Code innerhalb von `ensure` in jedem Fall ausgeführt wird.

```
def exception_if(bool)
  raise ArgumentError if bool
end

[true, false].each do |b|
  begin
    puts "Running with b=#{ b }"
    exception_if(b)
    puts "After possible exception"
  rescue ArgumentError => e
    puts "An error occured: #{ e }!"
  ensure
    puts "Always excuted, no matter what."
  end
end
```

Das Skript verdeutlicht den oben geschilderten Sachverhalt. Es ist möglich `rescue` auf bestimmte Fehler zu beschränken und diese Fehler in einer Variablen zu speichern.

Da das auftreten von Exceptions im Program in den meisten Fällen auf einen Programmierfehler zurückzuführen ist, sollte das Einfangen aller Fehler durch ein einfaches `rescue` verhindert werden. Es gibt Bibliotheken, zum Beispiel für Netzwerkzugriff, die einen Timeout als Exception anzeigen. An dieser Stelle sollte nur die bestimmte Timeoutexception eingefangen werden. Exceptions sollten nicht den Programmablauf beeinflussen, sondern die sollte durch normale Verzweigungen geschehen.

```
def div(a, b)
  return a / b
  rescue
    "Can't divide by zero"
```

```
end

puts div(1, 0)
puts div(6, 3)
puts div(1.0, 0.0)
puts div("a", 2)
```

Dieses Skript ist ein Negativbeispiel für den oben erwähnten Punkt, soll aber an dieser Stelle dazu dienen den Nachteil aufzuzeigen.

Zum einen wird jede mögliche Exception abgefangen, auch wenn sie mit der Division durch Null gar nichts zu tun hat, das ist im Zweifelsflall verwirrend. So führt die fehlerhafte Benutzung der Methode mit einem String als Parameter trotzdem zur Ausgabe, das man nicht mit Null dividieren könne.

Außerdem ist die Ausgabe der dritten Zeile Unendlich, statt der korrekten Fehlerbehandlung. Dies hat mit der binären Darstellung von gebrochenen Zahlen zu tun und wird nicht korrekt abgefangen.

Die gezeigte Methode ist also aus mehreren Gründen schlecht: zum einen fängt sie nicht jeden Fehler ab und manche Fehler führen zudem zu einer falschen Ausgabe und damit zu Konfusion entweder bei Ihnen als Programmierer oder beim Benutzer. Auf die Verwendung von Exceptions zur Änderung des Programmablaufs sollte man unbedingt vezichten und stattdessen auf `throw` oder normale Verzweigungen zurückgreifen. Die oben gezeigte Methode sollte daher besser so aussehen:

```
def div(a, b)
  if b == 0
    "Can't divide by zero."
  else
    a / b
  end
end

puts div(1, 0)
puts div(6, 3)
puts div("a", 2)
```

20.2 StandardError

In manchen Fällen reichen die verfügbaren Fehlerklassen nicht aus, um sinnvoll auszudrücken, was im jeweiligen Fall passiert ist. Es ist einfach möglich neue Fehlerklassen durch Vererbung einzuführen und diese zu benutzen. Die Klasse `StandardError` stellt dabei insbesondere zwei interessante Methoden zur verfügung, nämlich den Backtrace des Fehlers, sowie eine übergebene Nachricht, die den Fehler genauer beschreiben kann.

```
class CustomError < StandardError; end

def test
  raise CustomError, "Something is wrong"
end

begin
  test
rescue CustomError => e
```

```
  puts "An Error occured: #{ e.message } in #{ e.backtrace }"
end
```

20.3 Rückblick: Threads

Um diesem Kapitel eine praktische Relevanz zu verleihen wird auf ein Problem bei der Verwendung von Threads eingegangen. Falls Sie mit diesen schon ein bisschen experimentiert haben, ist Ihnen vielleicht eine Situation unterlaufen, die ähnlich zu der folgenden ist und Sie verwundert hat.

```
Thread.new do
  1/0
end

sleep(1)
puts "Everything is fine."
```

In diesem sehr einfachen Beispiel ist der Fehler einfach zu erkennen. Eine Division durch Null ist nicht möglich und wirft eine Exception, wenn Sie diesen Code allerdings ausführen werden Sie keine Exception erkennen. Der Interpreter weiß von der Exception in dem neuen Thread nichts und kann Sie daher auch nicht wie oben erklärt anzeigen. Es ist jedoch möglich das Anzeigen des Fehlers in Threads manuell durchzuführen.

```
def thread(&block)
  Thread.new do
    begin
      block.()
    rescue StandardError => e
      puts "Error:#{ e } occured\n#{ e.backtrace.join("\n") }"
    end
  end
end

thread { 1/0 }
sleep(1)
puts "Nothing is fine ):"
```

20.4 Catch und Throw

In anderen Programmiersprachen werden Exceptions manchmal verwendet um den Programmablauf zu beeinflussen, um zum Beispiel aus geschachtelten Schleifen herrauszukommen. Darauf sollte man in Ruby verzichten, da `raise` sehr langsam ist und daher nur in Ausnahmefällen sinnvoll ist.

In Ruby gibt es stattdessen eine andere Konstruktion, die für diesen Fall geeignet ist. `catch` leitet einen Block ein und erhält eine Sprungmarke auf die man mit `throw` zugreifen kann und damit die Ausführung des Blockes beenden kann. Ein optionaler zweiter Parameter von `throw` führt dazu, dass `catch` diesen zurückgibt.

```
i = catch :test  do
  (0..10).each do |i|
    throw(:test, i) if i == 5
    puts i
```

```
  end
end

puts "after loop: #{ i }"
```

`throw` ist im Gegensatz zu `raise` sehr leichtgewichtig, sollte aber in Anbetracht der eventuell schlechteren Lesbarkeit des Programms nur mit Vorsicht angewendet werden.

21 Debugging

Debuggen beschreibt eine Handlung Fehler im Programm zu suchen und zu beheben, da der interne Zustand des Programms im normalen Ablauf nicht bekannt sein sollte, ist es schwierig die Quelle eines Fehlers zu finden ohne das Programm zu ändern. Neben der sehr einfachen Möglichkeit das Programm mittels `puts` aufzublähen, werden hier zwei Bibliotheken vorgestellt die das untersuchen des Programmtextes mit nur sehr kleinen Änderungen an ihm ermöglichen.

21.1 debug

Die Bibliothek **debug** ist ein einfaches Tool, das es ermöglicht den Programmablauf zu unterbrechen und an dieser Stelle das Programm Schritt für Schritt auszuführen und zudem den Inhalt von Variablen auszugeben.

Das folgende Skript definiert eine Methode, die die Fakultät einer Zahl berechnen soll, doch funktioniert sie nicht, denn bei jedem Parameter gibt sie null zurück, anstatt des richtigen Wertes.

```
require "debug"

def fak(n)
  res = 1
  n.downto(0) { |i| res *= i }
  res
end

puts fak(5)
```

Die Bibliothek unterbricht die Ausführung des Programms und stattdessen ist es möglich mit `s` die Ausführung schrittweise durchzuführen. Mit `v l` werden alle lokalen Variablen angezeigt, die im aktuellen Kontext existieren. Damit ist es möglich das Problem einzugrenzen auf die Zeile 5, wo zwar die Fakultät korrekt berechnet wird, schließlich jedoch mit null multipliziert. Eine Änderung der Zeile zu `n.downto(1) { |i| res *= i }` behebt das Problem.

In größeren Programmen kann das komplette schrittweise Ausführen des Programms sehr schnell mühselig werden, daher existieren dafür zwei weitere Befehle: Das Anlegen von Breakpoints und das Fortsetzen des Programmablaufs bis zum nächsten Breakpoint. Dies erfolgt durch `b` (`break`) mit Angabe der Zeilennummer oder einem Methodennamen und `c` (`continue`).

21.2 pry-debugger

Die Bibliothek debug stellt eine einfache REPL (Read-Eval-Print-Loop) zur Verfügung, die es erlaubt Rubycode auszuführen und durch das vorhandene Skript zu navigieren. **Pry** ist ebenfalls eine REPL, die jedoch einige weitere Funktionen bereitstellt. Man kann die Debugvariante ähnlich einfach verwenden, wie in folgendem Skript gezeigt:

```
require "pry-debugger"

def fak(n)
  res = 1
  n.downto(0) { |i| res *= i; binding.pry }
  res
end

puts fak(5)
```

Im Gegensatz zu debug wird Pry erst beim Aufruf von `binding.pry` aktiv und erlaubt es somit auf die Verwendung von Breakpoints zu verzichten und im Sourcecode festzulegen welche Stellen einen interessieren, insbesondere kann man damit auch auf Verzweigungen zurückgreifen, um die Unterbrechungen im Programmablauf möglichst gut einzuschränken.

Das schrittweise Fortsetzen des Programms erfolgt dann mit `next` bzw. `step` und ebenfalls kann man an allen Stellen die lokalen Variablen usw anzeigen.

22 Testing

Für das automatisierte Testen des Programms existiert eine große Anzahl an Bibliotheken für Ruby. Neben der in der Standardbibliothek enthaltenen Bibliothek Test::Unit ist vor allem RSpec beliebt. Diese Seite soll sich insbesondere mit dem allgemeinen Ablauf von Test Driven Development (TDD) beschäftigen anhand eines Beispiels, das Test::Unit verwendet. Mit der Version Ruby 2.0 wurde Test::Unit durch Minitest ersetzt, die Nutzung ist jedoch noch möglich.

22.1 Test Driven Development

Es handelt sich dabei um eine Technik, die automatisierte Tests in den Entwicklungsprozess einbindet und ist in der Rubyentwicklergemeinschaft weit verbreitet. Dabei kommt es weniger darauf an, welche Bibliothek verwendet wird, sondern vielmehr, dass getestet wird. Dabei gibt es verschiedene Möglichkeiten die Güte einer Testsuite (Sammlung von Tests) zu messen: zum Beispiel Codecoverage. An dieser Stelle sei jedoch nur kurz darauf verwiesen, dass Übung den Umgang mit Tests sehr erleichtert.

Falls Sie bisher Aufgaben dieses Buches oder andere Probleme gelöst haben, sah Ihr Ablauf wahrscheinlich in etwa so aus. Nachdem sie eine entsprechende Funktionalität dem Programm hinzugefügt haben, benutzten Sie Ihr Programm, um zu überprüfen, ob es funktioniert. Die Problematik dieser Reihenfolge ist recht offensichtlich, da es nicht möglich ist alle Teile des Programms manuell zu testen und außerdem dauern manuelle Tests sehr lange und können nicht häufig durchgeführt werden.

TDD kehrt diese Reihenfolge um. Bevor man Funktionalität dem Programm hinzufügt wird ein automatisierter Test geschrieben, der das Programm auf diese Funktionalität hin überprüft, danach wird das Programm erweitert, bis die Tests nicht mehr fehlschlagen. In dem kommenden Beispiel soll unter Verwendung von automatisierten Tests eine einfache Applikation entwickelt werden.

22.2 Notizbuch

Es soll eine einfache Applikation entwickelt werden, die es erlaubt, Notizen zu erzeugen, anzuzeigen und zu löschen. Dafür wird eine Ordnerstruktur verwendet, wie sie bei Rubygems üblich ist und in dem entsprechenden Kapitel ausführlicher behandelt wird. Der Projektordner besteht dabei im wesentlichen aus zwei Unterordnern: lib/ und test/. In lib/ wird die Funktionalität des Programms beschrieben, Objekte definiert usw., während in test/ die Tests der korrespondierenden Programmteile liegen.

Zuerst befindet sich im Testordner eine Datei, welche die Bibliothek und eventuell andere Abhängigkeiten lädt, diese können dann alle Testdateien laden.

```
require "test/unit"
require_relative "../lib/notebook"
```

Dann wird der erste Test implementiert. Da sich das Notizbuch nur um Notizen dreht, wird dieses Objekt das Einzige sein, für den Tests existieren. Test::Unit funktioniert über das Anlegen einer Testklasse, deren Instanzmethoden als Tests ausgeführt werden. Die spezielle Methode `setup` wird am Anfang der Tests aufgerufen und dient dazu, die Tests vorzubereiten zum Beispiel durch das anlegen von Objekten.

```
require_relative "../test_helper"

class NoteTest < Test::Unit::TestCase
  CAPTION, CONTENT = "TDD", "Test Driven Development ist eine tolle Sache"

  def setup
    @note = Notepad::Note.new(CAPTION, CONTENT)
  end

  def test_caption
    assert @note.caption == CAPTION
  end
end
```

Das Ausführen dieses Tests führt wie erwartet zu einem Fehler, dass `Notepad::New` nicht existiert. Daher müssen wir an dieser Stelle die Klasse definieren.

```
module Notebook
  class Note
    def initialize(caption, content)
      @caption = caption
      @content = content
    end
  end
end
```

Das erneute Ausführen des Tests führt zu einem neuen Fehler, dass es keine Instanzmethode `caption` gibt. Diese kann leicht über `attr_reader` angelegt werden.

```
module Notebook
  class Note
    attr_reader :caption
  end
end
```

Danach läuft der Test erfolgreich durch und es danach leicht Möglich neue Tests für den Content einer Note, sowie für das ganze Notebook zu erstellen.

Der Vorteil von TDD zeigt sich besonders deutlich, wenn Sie bisher bereits größere Aufgaben bewältigt haben. Sie kamen dann eventuell an eine Stelle, dass sie ihren Code anders organisieren und umstrukturieren müssen, dann ist es von Vorteil, dass automatisierte Tests die Funktionalität des Programms vor und nachdem Refactoring garantieren.

23 Netzwerkprogrammierung

Mit der Hochverfügbarkeit von Internet seitdem letzen Jahrtausendwechsel bekamen auch alle Programme die Chance die Vernetzung zwischen Nutzern und Computern zu nutzen. Das erlaubt Ihnen als Programmentwickler beispielsweise einfache Updatemethoden oder Feedbackmachnismen zu implementieren.

Trotz der neuartigen Verfügbarkeit des Internets ist die zugrundeliegende Technologie bereits sehr alt. So bauen alle Netzwerkverbindungen heutzutage auf der Berkley Socket API für die Programmiersprache C auf, die in den siebziger Jahren entwickelt wurde. Daher wundert es nicht, das die Benutzung der Socketklasse in Ruby zwar in den meisten Fällen eine sehr elegante Lösung erlaubt, in einigen Grenzfällen jedoch Merkwürdigkeiten aufzeigt.

23.1 Sockets und Protokolle und ...

Bei der Verwendung von Netzwerken und insbesondere in diesem Kapitel sind und werden relativ viele Bezeichnungen benötigt, die dem einsteigenden Leser vielleicht ohne weitere Recherche überfordern würden, daher dient dieser Abschnitt dazu die einzelnen Stichworte kurz zu erläutern und sie miteinander in Kontext zu setzen.

- *Sockets* sind bidirektionale Verbindungen, die es erlauben Daten zwischen den beiden Endpunkten der Verbindung austauschen. Dies können Netzwerkverbindungen sein, oder UNIXSockets, die zwei Programme verbinden.
- *Protokolle* sind Vereinbarungen zwischen den beteiligten Parteien (Computer, Netzwerkinfrastruktur) über die Art und Weise der Kommunikation
  - *IP* das *internet protocol* ist vor allem in Verbindung mit der IPNummer wichtig, die jeden Rechner in einem Netzwerk eindeutig addressiert und ihn so für andere Rechner erreichbar macht.
  - *TCP* ist ein Protokoll darüber, wie die übermittelten Daten aussehen, insbesondere erzwingt TCP das Erreichen der Daten beim Kommunikationspartner in der richtigen Reihenfolge. Im Gegensatz dazu stellt *UDP* keine solche Funktionalität bereit und eignet sich daher besser für Echtzeitanwendungen.
- Client-Server-Modell ist eine hierarchische Methode mehrere Computer miteinander zu verbinden. Jeder Client verbindet sich zu einem Server, der die Kommunikation verwaltet. Dabei erfolgt jede Kommunikation zwischen den Clients über den Server. Im Gegensatz dazu verwalten die Kommunikation beim *peer-to-peer* die Programme selbst.

23.2 Chat

Um die recht theoretischen Betrachtungen des letzen Abschnitts in einen praktischen Kontext zu setzen, sollen zwei Programme entwickelt werden, die einen Chat zwischen beliebig vielen Personen erlaubt.

Um eine korrekte Übermittlung von Daten auf der Netzwerkseite zu verbessern bietet sich TCP als Protokoll an. Da beliebig viele Nutzer erlaubt sein sollen bietet sich ein Client-Server-Modell an, sodass ein Server die Kommuonikation zwischen allen verbundenen Clients verwaltet. Der Server wird außerdem für jede Verbindung mit einem Client einen Thread benötigen, um den entsprechenden Socket zu verwalten.

23.2.1 Client

Nach diesen Vorüberlegungen widmen wir uns zunächst der Clientseite.

```
require "socket"

host = 'localhost'
port = 2000
sock = TCPSocket.open(host, port)

puts "A Simple Chat"

# recieving messages
Thread.new { loop { puts sock.gets } }

# sending messages
catch :exit do
  loop do
    input = gets.chomp

    case input
    when "!exit", "!e"
      sock.puts input
      throw :exit
    else
      sock.puts input
    end
  end
end

sock.close
```

Das Programm besteht im wesentlichen aus drei Teilen: Initialisierung, Empfang und Senden.

Beim Initialisieren öffnen wir eine Verbindung über den Netzwerkstack des Betriebssystem zum eigenen Rechner an Port 2000. Das es sich nicht über eine wirkliche Verbindung über ein Netzwerk zu einem anderen Rechner handelt ist dabei irrelevant, da durch Austauschen der IP eine Verbindung hergestellt werden kann.

Das Empfangen von Nachrichten soll zeitlich unabhängig vom Senden funktionieren, daher benötigt der Client eine eigene parallel ausgeführte Schleife zum Empfangen. Die Nachrichten werden einfach ausgegeben.

Das Senden funktioniert analog zum Empfangen mit dem Unterschied, das jetzt auf den Socket geschrieben wird anstatt ihn zu lesen. Beendet wird das Programm mit `!e` oder `!exit` .

23.2.2 Server

```
require "socket"

class User
  attr_reader :name, :sock

  def initialize(name, sock)
    @name = name
    @sock = sock
  end
end

port = 2000
server = TCPServer.new(port)

users = []

loop do
  Thread.start(server.accept) do |sock|

    sock.puts "Please enter a name."
    name = sock.gets.chomp
    user = User.new(name, sock)
    users << user

    loop do
      msg = sock.gets.chomp

      case msg
      when "!exit", "!e"
        users.delete user
      else
        users.each do |u|
          u.sock.puts "#{ user.name }: #{ msg }"
        end
      end
    end
  end
end
```

Da viele Sachen Ihnen bekannt vorkommen sollten, wird nur näher auf die eigentliche Serverloop eingegangen. Bei jeder eingehenden Verbindung `server.accept` wird ein neuer Thread gestartet, der die Initialisierung dieser Verbindung übernimmt. Der Nutzer wird nach einem Namen gefragt und dieser zusammen mit seinem Socket in einem Array gespeichert. Bei einer eingehenden Nachricht wird überprüft ob es sich um das ausloggen des Nutzers handelt (`!e`) oder um eine gewöhnliche Nachricht, diese wird an alle Nutzer weitergeleitet.

Das interessante an der Verwendung von Netzwerken ist, dass Sie sehr wenig neues in diesem Kapitel gelernt haben. Hätten Sie vordem Lesen dieses Kapitels die Quelltexte überflogen, dann hätten Sie bis auf zwei, drei Zeilen keinen Code entdecken können der Ihnen unbekannt vorkommt. Wichtig bei der Verwendung der Verbindung zwischen zwei Rechnern ist weniger die Tatsache, sondern was das Programm damit anstellt.

23.3 HTTP

In der Einleitung wurde über das Internet gesprochen, dann jedoch nicht weiter erwähnt. TCP ist ein Netzwerkprotokoll und hat mit dem Internet grundlegend sehr viel zu tun. Mehr jedoch mit HTTP, dem Protokoll mit dem die meisten Dateien (zum Beispiel Webseiten) im Internet übertragen werden.

Die Bibliothek Net::HTTP unterscheidet sich ganz grundlegend von den oben kennengelernten Sockets, da zwar die Kommunikation über die Sockets läuft, aber HTTP keine Verbindungen kennt. Der Client schickt eine Anfrage an den Server und dieser Antwortet danach, es wird in seiner einfachen Form keine Verbindung aufrecht erhalten, sondern jede Abfrage muss über eine neue Verbindung ausgeführt werden.

23.3.1 Ein einfacher Browser

```
HELP = "VSB - very simple browsing
!exit, !e  --- exits the program
!help, !h  --- prints this help
Anything else is assumed as URL."

def sanitize(input)
  return input if input =~ /http:\/\//
  "http://#{ input }"
end

require "uri"
require "net/http"

input = ""

catch :exit do
  loop do
    puts "VSB - very simple browsing - try !help"
    print "CMD: "
    input = gets.chomp
    content = ""

    case input
    when "!e", "!q", "!exit", "!quit"
      throw :exit
    when "!h", "!help"
      puts HELP
    else
      input = sanitize(input)
      uri = URI(input)
      content = Net::HTTP.get(uri)
      content.gsub!(/<.+?>/, "").gsub!(/\s+/, " ")
      puts content
    end
  end
end
```

Dieser sehr einfache Browser nutzt die Bibliotheken `Net::HTTP` , um eine Website zu besuchen und `URI` um eine gegebene Adresse zu parsen. Der HTMLCode der entsprechenden Seite wird danach etwas aufbereitet und angezeigt.

Die Methode `sanitize` garantiert das am Anfang der Adresse das Protokoll angegeben wird, dies ist notwendig, um dem Nutzer auf die Engabe des Protokolls zu verzichten.

24 Rubyerweiterungen mit C

Ruby erlaubt es bei Bedarf nativen Code zu verwenden und ihn aus Rubyquelltexten heraus anzusprechen. Nativer Code beschreibt dabei Programmteile, die in einer anderen Programmiersprache geschrieben wurde und für die jeweilige Plattform übersetzt wurden. Das Ergebnis ist je nach Plattform eine `.dll` oder `.so` oder Ähnliches. Das ist insbesondere von Vorteil, wenn man auf Geschwindigkeitsprobleme stößt oder bereits vorhandene Bibliotheken benutzen möchte.

In diesem Kapitel soll eine C-Bibliothek entwickelt werden, die in einem Rubyprogramm eingebettet wird, um die Geschwindigkeit des Programms zu erhöhen. Dazu vergleichen wir mehrere Möglichkeiten auf ihre Vor- und Nachteile insbesondere in Hinblick auf Geschwindigkeit beim Ausführen des Programms und Einfachheit der Implementation. Ein Wrapper, der es erlaubt bereits vorhandene C-Bibliotheken anzusprechen würde dann ähnliche Schritte benötigen.

24.1 Beispielprogramm

Das Programm berechnet die Kreiszahl Pi und der entsprechende Quelltext in reinem Ruby sieht so aus:

```
class CalcPi
  def withPlainRuby(n)
    count = 0
    n.times do
      x = rand
      y = rand

      if x**2 + y**2 < 1
              count += 1;
      end
    end
    return 4.0*count/n
   end
end

n = ARGV[0].to_i
startTime = Time.new
CalcPi.new.withPlainRuby(n)
puts "Processing withplainRuby took #{ Time.new - startTime } seconds."
```

Der Algorithmus folgt dem Beispiel unter w:Kreiszahl#Statistische Bestimmung[1] und ist für eine tatsächliche Umsetzung nicht zu Empfehlen, da er nicht deterministisch ist. Bessere Berechnungsmöglichkeiten sind Reihendarstellungen, da diese eine definiertes Verhalten haben, wie sich die Genauigkeit der Berechnung in Abhängigkeit der Iterationen verhält. Die

1 https://de.wikipedia.org/wiki/Kreiszahl%23Statistische%20Bestimmung

statistische Bestimmung eignet sich jedoch gut, um die Geschwindigkeit in verschiedenen Implementationen zu vergleichen, da sie leicht zu verstehen und einfach zu implementieren ist. Die Zeit die das Programm benötigt unterscheidet sich je nach System und Anzahl der Iterationen und Bedarf ein paar Versuche, um eine sichtbare Dauer zu benötigen. In diesen Beispiel erzeugte das Programm folgende Ausgabe: `Processing withplainRuby took 3.682503619 seconds.`

24.2 RubyInline

Die beiden kommenden Abschnitte erfordern das kompillieren von C-Code und die Integration in Ruby. Dafür benötigt man die C-Bibliothek `ruby.h` und einen Compiler. Ke nach System unterscheidet sich die Installation der benötigten Software, falls Sie ein Linux verwenden, benutzen Sie am bessten die Paketverwaltung des Systems. Als Compiler eignet sich die gcc, die Headerdatei befindet sich in dem Ruby-Entwicklerpaket. Unter MacOSX müssen Sie für beide Probleme X-Code installieren.

Das Gem RubyInline erlaubt es einem C-Code direkt in den Rubycode einzubetten und damit seine Klasse zu erweitern.

```
require 'rubygems'
require 'inline'

class CalcPi
  inline :C do |builder|
  builder.flags << 'std=c99'
  builder.c '
    souble withInlineC(int n) {
      int count = 0;
      for(int i = 0; i <= n; i++) {
              double x = (rand()%1000)/1000;
              double y = (rand()%1000)/1000;

              if(x*x + y*y < 1){
                count ++;
              }
      }
      return 4.0*count/n;
    }'
  end
end

n = ARGV[0].to_i
startTime = Time.new
CalcPi.new.withInlineC(n)
puts "Processing withInlineC took #{ Time.new - startTime } seconds."
```

Das Codeschnipsel erzeugt eine neue Instanzmethode withInlineC, die Sie aus dem restlichen Programm wie gewohnt aufrufen können. Typkonvertierungen werden automatisch vorgenommen. Aufgrund der kompillierten Eigenschaften von C, sowie der strengen Typisierung und hochoptimierter Compiler ist dieser Code um eine Größenordnung schneller als der reine Ruby Code und erzeugt auf dem Testsystem folgende Ausgabe: `Processing withInlineC took 0.307251781 seconds.` . Der Nachteil liegt in einer längeren Ladezeit des Skriptes, da der Inlinecode erst kompiliert werden muss. Diese Variante lohnt sich nur, falls es sich um eine kritische Komponente handelt, die während der Laufzeit des Programms oft benutzt wird, so dass sich die anfänglich größere Ladezeit lohnt.

24.3 C Erweiterung

Das Schreiben einer C-Erweiterung sollte genau überdacht werden, denn obwohl C eine sehr schnelle Sprache ist bringt es einige Probleme mit sich und es ist aufgrund niedriger Abstraktion sehr viel schwieriger funktionierenden Code zu erzeugen. Trotzdem soll an dieser Stelle auf die Möglichkeit und deren Umsetzung hingewiesen werden. Da es an einigen Stellen nötig werden kann, dass Sie eine Erweiterung in C schreiben müssen, zum Beispiel wenn Sie auf bestehende C-Programme und Bibliotheken zurückgreifen müssen.

Zunächst der C-Quelltext, der wie in den beiden Beispielen oben eine Klasse `CalcPi` erzeugt mit zwei Instanzmethoden: `initialize` und `withCExtension` .

```
//calcPi.c

#include <ruby.h>

static VALUE t_withCExtension(VALUE self, VALUE n) {
  // Führt Konvertierung zwischen der Rubydarstellung von n und einer C-internen
 Darstellung durch.
  int limit = NUM2INT(n);
  int count = 0;
  int i;
  for(i = 0; i <= limit; i++) {
    double x = (rand()%1000)/1000.0;
    double y = (rand()%1000)/1000.0;

    if(x*x + y*y <= 1.0){
      count++;
    }
  }

  // Erzeugt eine neue Ruby-Gleitkommazahl und gibt diese an das aufrufende
 Programm.
  return rb_float_new(4.0 * count / limit);
}


static VALUE t_init(VALUE self) {
        return self;
}

// Diese Funktion wird vom Rubyinterpreter aufgerufen, wenn er die Erweiterung
 lädt, dabei muss wie in diesem Fall calcPi dem Dateinamen entsprechen.
void Init_calcPi() {
  // Erzeugt die Klasse CalcPi, deren Elternklasse Object ist.
  VALUE calcPi = rb_define_class ("CalcPi", rb_cObject);

  // Erzeugt die Methoden
  rb_define_method(calcPi, "withCExtension", t_withCExtension, 1);
  rb_define_method(calcPi, "initialize", t_init, 0);
}
```

Nachdem Sie diesen Quelltext unter dem Namen `calcPi.c` gespeichert haben. Müssen Sie diesen kompillieren, eine Bibliothek, die diese Aufgabe übernimmt ist `mkmf` (make makefile). Konventionell ist es eine Datei `extconf.rb` anzulegen mit folgendem minimalen Inhalt:

```
require 'mkmf'
create_makefile('calcPi')
```

Nachdem ausführen dieses Skriptes und dem starten von `make` wurde eine Datei erstellt mit dem Namen `calcPi.so` (unter Linux, je nach Betriebssystem auch `.dll` oder `.bundle`). Diese kann nun mittels `require 'calcPi'` in Rubyskripten verwendet werden.

```
require './calcPi'

n = ARGV[0].to_i

startTime = Time.new
puts CalcPi.new.withCExtension(n)
puts "Processing withCExtension took #{ Time.new - startTime } seconds."
startTime = Time.new
```

Die Erweiterung muss jetzt nicht beim laden des Skriptes kompilliert werden, dadurch reduziert sich die Ladezeit gegenüber der Inlinemethode. Nachteile sind das komplizierte Interface und die vielen Fallstricke aufgrund der Verwendung von C (die natürlich bei der Inlinemethode auch auftreten). Zur Laufzeit ergeben sich gegenüber der Inlinemethode kaum Geschwindigkeitsvorteile, sondern die getestete Ausführungszeit liegt im selben Bereich: `Processing withCExtension took 0.291250096 seconds.`

24.4 Zusammenfassung

Prinzipiell sollte das Nutzen einer nativen Bibliothek genau überdacht werden, weil es sich um mehr Aufwand handelt als eine vergleichbare Implementation in Ruby. Falls Sie in ihren Programmen auf Geschwindigkeitsprobleme stoßen, sollten Sie zunächst zu andere Mitteln greifen als die Funktionen in C umzuschreiben. Überprüfen Sie genau welche Programmteile kritisch sind, wo also viel zeit verbraucht wird und prüfen Sie ob Sie diese Bereiche verbessern können. Das kann das Zwischenspeichern von Berechnungen umfassen, oder das Wechseln eines Algorithmus. Je nach Anwendung kann es auch danach noch zu Engpässen kommen, dann kann es Vorteilhaft sein das gesamte Programm modularer zu gestallten und Programmteile in anderen Programmiersprachen zu schreiben. Als Stichwort soll hier Service Oriented Architecture erwähnt sein. Das löst zwar nicht die Probleme, die durch das Verwenden mehrerer Programmiersprachen entstehen, aber erlaubt es die Programmteile als abgeschlossene Teile zu betrachten.

Auf der anderen Seite ist das Verwenden von C-Code relativ einfach möglich, wenn Sie also C-Bibliotheken finden die nützliche Funktionalität bereitstellen, dann können Sie diese Nutzen und müssen keinen Arbeit darauf verwenden diese Funktionen neu zu schreiben.

25 Rubygems

Sie haben bisher in diesem Buch viele Bibliotheken kennengelernt und benutzt. Das einzige was an diesen Bibliotheken besonders war, ist die Tatsache, das sie Teil der Standardbibliothek sind und daher mit einer Rubyinstallation ausgeliefert werden. Es ist natürlich auch möglich eigene Bibliotheken zu entwickeln und Drittanbieterbibliotheken zu verwerden. Darum hat sich in der Rubycommunity ein eigenes Ökosystem gebildet, das es erlaubt auf einfache Art und Weise Bibliotheken zu verwenden und auch Ihre Bibliotheken Dritten einfach zugänglich machen lässt.

25.1 Rubygems und Bundler

Gems ist ein anderer Name für Rubybibliotheken, das entsprechende Programm Rubygems ist Teil einer Rubyinstallation. Es erlaubt einen einfachen Zugang zu Gems auf der Plattform Rubygems anderer Entwickler und somit die einfache Nutzung dieser Bibliotheken. Es ist auch möglich andere Quellen als Rubygems zu verwenden. Bundler hingegen baut auf Rubygems auf und erlaubt die Verwaltung von Rubygems zwischen verschiedenen Projekten, dies ist insbesondere dann notwendig, wenn ein Projekt auf einer anderen Version eines Gems beruht als ein anderes Projekt und Sie auch anderen Entwicklern einen einfachen Zugang zu Ihrem Projekt ermöglichen wollen.

25.2 Entwicklung eines Rubygems

Weil die Nutzung und verschiedene damit verbundene Probleme natürlich auch die Entwicklung eines Gems betreffen, wird dieses Buch die herangehensweise an Gems etwas Umkehren. Nicht die Nutzung des Gems, sondern seine Entwicklung steht im Vordergrund, dies ist insbesondere von Vorteil, da man die Unterschiedlichen Benutzungsmöglichkeiten hintergründig versteht.

In diesem Abschnitt soll ein Gem entwickelt werden, dass die Standadrdbibliothek nicht bereitstellt. Es soll eine Methode `Object#any?` entwickelt werden, dass eine Kollektionklasse erhält und prüft, ob sie das Objekt beinhaltet. Starten wir also mit der Generierung der Filestruktur:

```
bundle gem any
```

Hierbei wird die typische Filestruktur eines einfachen Gems erzeugt. Insbesondere wichtig ist an dieser Stelle die Datei `any.gemspec` .

Sie enthält Metadaten, die das Gems betreffen, wie deren Autoren, die beinhalteten Dateien, usw. Es wird davon ausgegangen, das die Versionskontrolle mit Git erfolgt ansonsten

muss die Zeile `gem.files = `git ls-files`.split($\)` angepasst werden. Da Git und insbesondere Github in der Rubycommunity weit verbreitet sind und auch die Distribution von Gems über Github sehr konfortabel ist, lohnt sich jedoch ein Blick auf Git und insbesondere, falls Sie noch kein Versionskontrollsystem kennen ist Git eine gute Wahl damit zu beginnen. Diese Buch wird nicht näher auf Versionskontrolle mit Git eingehen. Da der Umfang dieses Projekts überschaubar ist, sind jedoch keine Vorkenntnisse nötig, da an den entsprechenden Stellen die Funktion der Befehle kurz erläutert wird.

Der nächste Schritt ist also der erste Commit: `git add . && git commit -m"initial commit"` . `add` fügt die angegebenen Dateien der Versionskontrolle hinzu, in diesem Fall das aktuelle Verzeichnis inklusive aller Unterverzeichnisse. Ein `commit` erzeugt eine neue Version des Quellcodes und versieht sie mit einem Kommentar, dadurch ist es möglich später zwischen den einzelnen Commits zu wechseln und sie zu untersuchen.

Als nächstes ist es nötig die Tests und die Implementierung der gewünschten Methode zu schreiben, um den Fokus nichts zu verlieren sind an dieser Stelle nur die entsprechenden Quelltexte angegeben.

```
# test/any.rb

require "test/unit"
require_relative "../lib/any"

class AnyTest < Test::Unit::TestCase
  def test_object
    assert Object.instance_methods.include? :any?
  end

  def test_arrays
    ary = [1, "abc"]
    assert "abc".any? ary
    assert !(2.any? ary)
    assert [2,3].any? [[2,3],[3],[2,5]]
  end
end


# lib/any.rb

require_relative "any/version"

class Object
  def any?(collection)
    collection.include? self
  end
end
```

An dieser Stelle sei nur auf die Zeile `require_relative "any/version"` hingewiesen. Der Codegenerator erzeugt an dieser Stelle `require` und das wird durch Rubygems abgefangen, um die Pfade anzupassen. Es handelt sich dabei um ein Überbleibsel aus Ruby 1.8 und sollte durch das in Ruby 1.9 eingeführte `require_relative` ersetzt werden.

Zum Schluss müssen die neuen Dateien noch in Git hinzugefügt werden, dies erfolgt analog zum ersten Commit mit `git add . && git commit -m"Object#any? implemented"` .

25.2.1 Nutzung und Distribution

In diesem einfachen Fall ist es natürlich ohne weiteres Möglich den Quellcode direkt in andere Projekte zu kopieren und dort zu nutzen. Dies sollten Sie immer dann in Erwähnung ziehen, falls die Bibliothek einen sehr speziellen Nutzungsbereich hat und zwischen einzelnen Applikationen verändert werden muss. Dann ist es einfacher den Quellcode vor Ort zu verwalten, als für jedes Projekt das Gem anzupassen.

Natürlich birgt obiges Verfahren einige Probleme, da Sie keine Updates für Ihr Gem bereitstellen können und ist somit nur für sehr begrenzte Anwendungsfälle geeignet. Besser ist es dann das Gem insgesamt zu verteilen. Dafür muss es zunächst gebaut werden, wozu das `any.gemspec` dient. Das bauen erfolgt mit dem Befehl `gem build any.gemspec` und erzeugt eine Datei mit dem entsprechenden Namen und der Version des Gems. Jetzt ist es möglich das Gem mittels `gem install any-0.0.1.gem` zu installieren und mittels `require "any"` zu nutzen. In einer Prysession ergibt sich zum Beispiel:

```
[1] pry(main)> require "any"
=> true
[2] pry(main)> 2.any? [2,3,4]
=> true
```

Dieser Weg ist für die Nutzer bereits deutlich angenehmer, da Sie jetzt Updates des Gems verteilen können, indem Sie die Versionsnummer ändern und das Gem neu verteilen. Das Problem ist, dass Ihre Nutzer noch von einer neuen Version erfahren müssen. Es eignet sich also zum Beispiel für einen engen Nutzerkreis. Für generalisierte Bibliotheken und weite Nutzerkreise eignen sich die Plattformen Rubyforge und Github. Dadurch wird es für Sie zwar etwas mühseliger ihr gem zu verteilen, da Sie sich bei diesen Plattformen anmelden müssen und das hochgeladene Gem warten müssen, doch ist es für ihre Nutzer möglich die Gems einfach mittels `gem install GEMNAME` von Rubyforge oder mittels `gem install GEMNAME --source GITHUBURL` um von Github zu installieren.

25.3 Gemfile

Da das kleine Beispiel der Bibliothek Any keine Abhängigkeiten zu anderen Gems beinhaltete, soll an dieser Stelle noch einmal extra darauf eingegangen werden, wie Sie mit Abhängigkeiten in Ihren Projekten umgehen können. An dieser Stelle wird **bundler** benutzt um das Gemfile auszuwerten und dementsprechend die Abhängigkeiten zu verwalten.

Ein Gemfile beginnt mit Angabe der Quelle für die Pakete, das ist im Normalfalls Rubyforge und die Zeile lautet: `source 'https://rubygems.org'` . Danach erfolgt die Angabe der Abhängigkeiten durch `gem GEMNAME VERSION` . Es ist möglich die Gems Gruppen zuzuordnen und damit beispielsweise für bestimmte Abschnitte der Entwicklung nur bestimmte Gems zu benutzen. Als Beispiel aus einem frisch generierten Railsprojekt aus dem einige Kommentare entfernt wurden:

```
source 'https://rubygems.org'

gem 'rails', '3.2.11'

# Bundle edge Rails instead:
```

```
# gem 'rails', :git => 'git://github.com/rails/rails.git'

gem 'sqlite3'

group :assets do
  gem 'sass-rails',   '~> 3.2.3'
  gem 'coffee-rails', '~> 3.2.1'
  gem 'uglifier', '>= 1.0.3'
end

gem 'jquery-rails'
```

Installiert wird mit dem Aufruf von `bundle install` . Es ist möglich zahlreiche Optionen zu übergeben, wie in diesem Beispiel: `git => 'git://github.com/rails/rails.git'` , um die Quelle des Gems festzulegen. Die unterschiedlichen Versionsangaben beziehen sich auf das Verhalten der Gems bei Installation und Updates und verhalten sich wie folgt:

- "3.2.11" verwende exakt diese Version
- ">=1.0.3" verwende mindestens die Version 1.0.3, aber auch jede höhere
- "~> 3.2.1" verwende mindestens die Version 3.2.1, führe jedoch kein Majorupdate auf Version 4 durch.

Die bei der Installation der Gems angelegte Datei Gemfile.lock spezifiziert die installierte Version und kann dazu verwendet werden Versionsanforderungen im Gemfile anzupassen, oder kann auch mit anderen Projektbeteiligten geteilt werden, damit alle die gleichen Versionen benutzen. Der Normalfall im generierten Gem ist, dass das Gemfile.lock nicht versioniert wird und kann in der Datei `.gitignore` geändert werden.

26 Rake

Im Unixumfeld ist das Programm Make weit verbreitet, um Aufgaben zu automatisieren, die mehrere Zwischenschritte erfordern. Dies kann zum Beispiel das kompillieren eines Programmes sein oder das Updaten eines Servers.

Für Ruby gibt es das Programm Rake. Es ergänzt die Vorzüge von Make wie leichte Automatisierbarkeit von Abläufen und einfaches ansprechen über die Kommandozeile um die Programmiersprache Ruby. Ein Rakefile ist eine Rubyquelltextdatei.

26.1 Syntax

```
today = Time.new

task :default => [:time]

desc "Printing the time"
task :time do
  puts today.to_s
end
```

Dieses Skript beschreibt einen einzigen `task` und definiert ihn als default. Dadurch kann er sowohl durch `rake time` als auch durch `rake` auf der Kommandozeile ausgeführt werden. In diesem Fall wird einfach die aktuelle Zeit auf der Kommandozeile angezeigt werden. Trotzdem dieses Beispiel nicht besonders umfangreich ist, kann man sich bei mehreren Tasks, die unterschiedliches bewirken sollen schnell Vorteile von Rake gegenüber einfachem Ruby vorstellen.

Rake erlaubt es außerdem Abhängigkeiten zwischen den Tasks zu definieren. Oben sieht man das schon, dass der Task `default` den Task `time` als Abhängigkeit besitzt, dadurch wird dieser Task ausgeführt bevor `default` ausgeführt wird. In diesem Fall wird also zuerst der Task `time` ausgeführt und zeigt die aktuelle Zeit an, danach wird `default` ausgeführt und tut nichts.

Besonders stark ist Rake bei der Transformation von Dateien, indem zum Beispiel Dateien aus anderen erzeugt werden. Dann lassen sich die Dateien als Abhängigkeiten der Zieldatei angeben und die Zieldatei wird nur dann neu erzeugt, wenn sich die Abhängigkeiten verändert haben.

Das folgende Skript fügt alle Dateien aus dem Unterordner `src` zusammen in einer Datei `result` :

```
files = FileList["src/**/*"]

task :default => "result"
```

```
file "result" => files do |t|
  content = files.map { |file| File.read(file) }.join("\n")
  puts "I'm writing the file"
  File.write(t.name, content)
end
```

Hier wurde an mehreren Stellen auf Rake zurückgegriffen, um die Arbeit zu erleichtern. Die FileList Klasse erlaubt das leichte zugreifen auf Dateien, die einem bestimmten Format entsprechen, in diesem Fall alle Dateien im Ordner `src` und dessen Unterordner, es ist auch möglich die Auswahl auf bestimmte Dateinamen einzuschränken.

Die Methode `file` erlaubt es wie `task` einen RakeTask zu erstellen. Im Gegensatz zu `task` das immer ausgeführt wird, wird `file` nur ausgeführt wenn die Abhängigkeiten sich geändert haben. Dadurch wird beim erstmaligen ausführen des Skriptes mit `rake` sowohl die Datei geschrieben, als auch die Zeile ausgegeben. Bei jedem weiteren Aufruf wird aber nichts mehr getan, außer eine der Dateien im Ordner `src` hat sich geändert. Dieses Verhalten ist insbesondere wünschenswert, wenn es sich um langsame Aufgaben handelt, so dass der langsame Teil nur bei Bedarf ausgeführt wird.

26.1.1 Automatisches Kompillieren

Dieses Rakefile wurde benutzt um im Kapitel über C-Erweiterungen die Erweiterungen automatisch zu kompillieren. Es müssen sich alle Dateien im Unterordner `lib` befinden.

```
Dir.chdir Dir.pwd + "/lib/"

FILES = []
Dir.open("./").each do |file|
  if file =~ /(.+)\.c$/
    FILES << $1
  end
end

task :default => [:compile]

desc "Compiling ..."
task :compile => [:extconf] do
  ruby "extconf.rb"
  system "make"
end

desc "Making extconf.rb ..."
task :extconf do
  file = File.open("extconf.rb", "w")
  file.puts "require 'mkmf'"
  FILES.each do |ext|
    file.puts "create_makefile '#{ ext }'"
  end
  file.close
end

desc "Cleaning everything ..."
task :clean do
  system "rm *.so *.o Makefile extconf.rb"
end
```

27 Aufgaben

Dieses Kapitel soll dieses Buch mit Aufgaben zu verschiedenen Themen abschließen. Dabei können Sie versuchen die verschiedenen Aufgaben an jeder Stelle des Buches versuchen zu lösen, eventuell ist es dadurch jedoch schwerer oder unmöglich die Aufgabe zu lösen, da Ihnen bestimmte Mittel aus späteren Kapiteln fehlen. Das kann auch dazu führen, dass Sie eventuell die vorgestellte Lösung nicht nachvollziehen können.

Mindestens sollten Sie an dieser Stelle den Abschnitt Grundlagen durchgearbeitet haben, um die Aufgaben lösen zu können.

Die Aufgaben bestehen aus einem Thema und einer Aufgabenbeschreibung, sowie Gedanken zur Lösung des Problems. Wenn Sie eine Aufgabe bearbeiten sollten Sie das tun, ohne auf die hier vorgestellte Lösung zu schauen, um am meisten aus der Bearbeitung der Aufgabe zu lernen.

27.1 Passwortgenerator

Diese Aufgabe besteht darin ein Programm zu entwickeln, das zufällig generierte Passwörter ausgibt. Dabei könnte das Programm den folgenden Funktionsumfang haben:

- Variable Länge der Passwörter
- Auswahl legitimer Zeichen
- Speichern der letzten Einstellungen
- Erneutes Erzeugen von Standardeinstellungen

Musterlösung

```
Encoding: utf-8

require "yaml
require ßet

class Preferences < Hash
  def initialize
    super
    self[:length] = 10
    self[:chars] = ::Set.new
    (ä
..ß
).each { |c| self[:chars] << c }
    (Ä
..SZ
).each { |c| self[:chars] << c }
    "!\
§$%&/()=?*:;{[]}\\+-.,".split().each { |c| self[:chars] << c }
  end
end

class Password
  def self.gen(pref)
    pw =
    pref[:length].times { pw << pref[:chars].to_a.shuffle[0] }

    pw
  end
end

if File.file? ".pref
  pref = YAML.load_file(".pref")
else
  pref = Preferences.new
end


catch :exit do
  input =


  loop do
    input = gets.chomp
    value = input[/[=]+$/]


    case input
    when /!e/
      throw :exit
    when /!n/
      pref = Preferences.new
    when /!a/
      value.split().each { |c| pref[:chars] << c }
    when /!r/
      value.split().each { |c| pref[:chars].delete c }
    when /!l/
      pref[:length] = value.to_i
    else
      puts "Generated password: " + Password.gen(pref)
    end
  end
end


open(".pref","w") { |f| YAML.dump(pref,f) }
```

27.2 Primzahlprüfer

Das Programm soll bei Eingabe einer Zahl überprüfen, ob es sich um eine Primzahl handelt oder nicht. Es kann sowohl als Kommandozeilenapplikation, als auch als interaktive Applikation gestalltet werden.

Musterlösung

```
Encoding: utf-8
require "prime

class PrimeChecker
  def self.run(argv)
    if argv.empty?
      interactive_prime_checker
    else
      prime_checker(argv.map(&:to_i))
    end
  end

  def self.prime_checker(numbers)
    numbers.each do |num|
      puts
{ num } { Prime.prime?(num) ? ïs a prime.
 : ïsn't a prime.
 }
    end
  end

  def self.interactive_prime_checker
    puts "Please enter some numbers or nothing to exit.
    loop do
      input = gets.chomp
      break if input.empty?
      numbers = input.split(/\s+/).map(&:to_i)
      prime_checker(numbers)
    end
  end
end


PrimeChecker::run(ARGV)
```

27.3 Konvertieren in das metrische System

Im amerikanischen Sprachraum wird häufig anstelle des metrischen das imperiale Einheitensystem verwendet, was bei Menschen zu Konfusionen führen kann, die die Einheiten nicht gewohnt sind. Die Aufgabe ist es also einen Konverter für imperiale Einheiten wie Zoll, Fuß, Gallonen und so weiter zu schreiben, der dann eine metrische Ausgabe vornimmt.

Bedenken Sie, dass es viele solcher Einheiten gibt und das die Struktur ihres Program das Hinzufügen neuer Einheiten so leicht wie möglich machen sollte. Die Beispiellösung erlaubt das Aufrufen des Programms mittels `ruby converter.rb 200lbs 6ft 3inch` .

Musterlösung

```
module Converter
  def self.run(args)
    puts convert(args)
  end

  def self.convert(args)
    args.map { |arg| Unit.parse(arg).to_metric }
  end

  class Unit
    def self.units
      {
        "lbs
 => Pound,
        "ft
 => Feet,
        "inch
 => Inch
      }
    end

    def self.parse(arg)
      unit = arg[/\D+\Z/i]
      value = arg[/\d+(\.\d+)?/].to_f

      units.fetch(unit).new(value)
    end

    def initialize(value)
      @value = value
    end

    def to_s

{ @value }{ unit_string }
    end
  end

  class Kilogram < Unit
    def to_metric
      self
    end

    def unit_string
      "kg
    end
  end

  class Pound < Unit
    def to_metric
      Kilogram.new(@value * 0.454)
    end

    def unit_string
      "lbs
    end
  end

  class Meter < Unit
    def to_metric
      self
    end

    def unit_string
      "m
    end
  end

  class Feet < Unit
    def to_metric
      Meter.new(@value * 0.3048)
    end

    def unit_string
      "ft
    end
  end
```

28 Ich brauche Hilfe!

28.1 Ruby-Doc.org

Eine umfangreiche Dokumentation findet sich auf `http://www.ruby-doc.org/`. Dort sind viele Klassen mit ihren Methoden aufgeführt. Dort kann man nachschlagen, wenn man eine bestimmte Funktionalität sucht.

28.2 Integrierte Dokumentation

Im Paketumfang von Ruby ist **ri** enthalten, ein Dokumentationsbetrachter für die Konsole.

Wird das Programm ohne Parameter aufgerufen, so zeigt es Hinweise zur Bedienung an. Zu beachten ist, dass ri seine Ausgabe immer über den in der Umgebungsvariable *$PAGER* eingestellten Ausgabefilter leitet.

Um einen Überblick über alle Klassen mit Dokumentation zu erhalten, ruft man ri mit dem Parameter **-c** (oder **--classes**) auf:

```
$ ri -c
---------------------------------------------- Known classes and modules

     Abbrev, ArgumentError, Array, BDB, BDB::Btree, BDB::Common,
     BDB::Cursor, BDB::Env, BDB::Hash, BDB::Lock, BDB::LockDead,
     BDB::LockError, BDB::LockGranted, BDB::LockHeld, BDB::Lockid,
     BDB::Lsn, BDB::Queue, BDB::Recno, BDB::Recnum, BDB::Sequence,
     BDB::Txn, Base64, Base64::Deprecated, Benchmark, Benchmark::Job,
     Benchmark::Report, Benchmark::Tms, Bignum, Binding, CGI,
...
     Iconv, Iconv::BrokenLibrary, Iconv::Failure,
     Iconv::IllegalSequence, Iconv::InvalidCharacter,
     Iconv::InvalidEncoding, Iconv::OutOfRange, IndexError, Integer,
     Interrupt, Jabber, Jabber::AuthenticationFailure, Jabber::Client,
     Jabber::Component, Jabber::Connection, Jabber::DiscoFeature,
     Jabber::DiscoIdentity, Jabber::DiscoItem, Jabber::Error,
     Jabber::ErrorException, Jabber::Helpers,
     Jabber::Helpers::FileSource, Jabber::Helpers::FileTransfer,
...
     Jabber::XMucUserInvite, Jabber::XMucUserItem, Jabber::XRoster,
     Jabber::XRosterItem, Kernel, LoadError, LocalJumpError, Logger,
     Logger::Application, Logger::Error, Logger::Formatter,
     Logger::LogDevice, Logger::LogDevice::LogDeviceMutex,
     Logger::Severity, Logger::ShiftingError, Marshal, MatchData, Math,
     Matrix, Matrix::Scalar, MediaWiki, MediaWiki::Article,
     MediaWiki::Category, MediaWiki::MiniBrowser, MediaWiki::Table,
     MediaWiki::Wiki, Method, Module, Mutex, NameError,
     NameError::message, NilClass, NoMemoryError, NoMethodError,
...
     Zlib::VersionError, Zlib::ZStream, fatal
```

Für die Übersicht über Methoden einer Klasse genügt ein Aufruf mit dem Klassenname als Parameter:

```
% ri String|cat
------------------------------------------------------------ Class: String
     A +String+ object holds and manipulates an arbitrary sequence of
     bytes, typically representing characters. String objects may be
     created using +String::new+ or as literals.

     Because of aliasing issues, users of strings should be aware of the
     methods that modify the contents of a +String+ object. Typically,
     methods with names ending in ``!'' modify their receiver, while
     those without a ``!'' return a new +String+. However, there are
     exceptions, such as +String#[]=+.

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


Includes:
---------
     Comparable(<, <=, ==, >, >=, between?), Enumerable(all?, any?,
     collect, detect, each_cons, each_slice, each_with_index, entries,
     enum_cons, enum_slice, enum_with_index, find, find_all, grep,
     include?, inject, map, max, member?, min, partition, reject,
     select, sort, sort_by, to_a, to_set, zip)


Class methods:
--------------
     new


Instance methods:
-----------------
     %, *, +, <<, <=>, ==, =~, [], []=, capitalize, capitalize!,
     casecmp, center, chomp, chomp!, chop, chop!, concat, count, crypt,
     delete, delete!, downcase, downcase!, dump, each, each_byte,
     each_line, empty?, eql?, gsub, gsub!, hash, hex, include?, index,
     initialize_copy, insert, inspect, intern, length, ljust, lstrip,
     lstrip!, match, next, next!, oct, replace, reverse, reverse!,
     rindex, rjust, rstrip, rstrip!, scan, size, slice, slice!, split,
     squeeze, squeeze!, strip, strip!, sub, sub!, succ, succ!, sum,
     swapcase, swapcase!, to_f, to_i, to_s, to_str, to_sym, tr, tr!,
     tr_s, tr_s!, unpack, upcase, upcase!, upto
```

Um eine Funktion nachzuschlagen, wird dieser mit einem Punkt (**.**) an den Klassenname angehangen:

```
% ri Fixnum.to_s
---------------------------------------------------------- Fixnum#to_s
     fix.to_s( base=10 ) -> aString
------------------------------------------------------------------------
     Returns a string containing the representation of _fix_ radix
     _base_ (between 2 and 36).

        12345.to_s       #=> "12345"
        12345.to_s(2)    #=> "11000000111001"
        12345.to_s(8)    #=> "30071"
        12345.to_s(10)   #=> "12345"
        12345.to_s(16)   #=> "3039"
        12345.to_s(36)   #=> "9ix"
```

Da jedoch Instanzmethoden und Klassenmethoden (vgl. *statische Methoden* in Java) den gleichen Namen haben können, kann für Instanzmethoden ein Doppelkreuz (#) und Klassenmethoden zwei Doppelpunkte (**::**) verwendet werden.

28.3 Interactive Ruby Shell

Wenn sie Linux/OSX verwenden geben sie `irb` im Terminal ein.

```
irb(main):001:0>
```

29 Autoren

Edits	User
2	Bk1 168[1]
23	Daigoro[2]
2	Dexbot[3]
34	Dirk Hünniger[4]
2	Emesem[5]
1	Erkan Yilmaz[6]
3	Gebu[7]
2	Graf Westerholt[8]
12	GreasanDev[9]
9	Hagemann[10]
1	Heuler06[11]
2	JackPotte[12]
3	Jan˜dewikibooks[13]
4	JohannesB.˜dewikibooks[14]
1	John N.[15]
3	Juetho[16]
5	Kaarlie[17]
3	Kuroi-ryu˜dewikibooks[18]
1	Lipedia[19]
1	MRosetree[20]
1	MichaelFrey[21]

1 https://de.wikibooks.org/wiki/Benutzer:Bk1_168
2 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Daigoro&action=edit&redlink=1
3 https://de.wikibooks.org/wiki/Benutzer:Dexbot
4 https://de.wikibooks.org/wiki/Benutzer:Dirk_H%25C3%25BCnniger
5 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Emesem&action=edit&redlink=1
6 https://de.wikibooks.org/wiki/Benutzer:Erkan_Yilmaz
7 https://de.wikibooks.org/wiki/Benutzer:Gebu
8 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Graf_Westerholt&action=edit&redlink=1
9 https://de.wikibooks.org/wiki/Benutzer:GreasanDev
10 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Hagemann&action=edit&redlink=1
11 https://de.wikibooks.org/wiki/Benutzer:Heuler06
12 https://de.wikibooks.org/wiki/Benutzer:JackPotte
13 https://de.wikibooks.org/wiki/Benutzer:Jan~dewikibooks
14 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:JohannesB.~dewikibooks&action=edit&redlink=1
15 https://de.wikibooks.org/wiki/Benutzer:John_N.
16 https://de.wikibooks.org/wiki/Benutzer:Juetho
17 https://de.wikibooks.org/wiki/Benutzer:Kaarlie
18 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Kuroi-ryu~dewikibooks&action=edit&redlink=1
19 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Lipedia&action=edit&redlink=1
20 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:MRosetree&action=edit&redlink=1
21 https://de.wikibooks.org/wiki/Benutzer:MichaelFrey

1	NeuerNutzer2009[22]
1	Nkoehring[23]
1	NullPlan[24]
94	Philip91[25]
1	Qwertz84[26]
3	Stephan Kulla[27]
1	Sundance Raphael[28]
157	Tray[29]
1	Tz92[30]
8	Xml user[31]

22 https://de.wikibooks.org/wiki/Benutzer:NeuerNutzer2009
23 https://de.wikibooks.org/wiki/Benutzer:Nkoehring
24 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:NullPlan&action=edit&redlink=1
25 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Philip91&action=edit&redlink=1
26 https://de.wikibooks.org/wiki/Benutzer:Qwertz84
27 https://de.wikibooks.org/wiki/Benutzer:Stephan_Kulla
28 https://de.wikibooks.org/wiki/Benutzer:Sundance_Raphael
29 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Tray&action=edit&redlink=1
30 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Tz92&action=edit&redlink=1
31 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Xml_user&action=edit&redlink=1

Abbildungsverzeichnis

- GFDL: Gnu Free Documentation License. http://www.gnu.org/licenses/fdl.html
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/deed.en
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- GPL: GNU General Public License. http://www.gnu.org/licenses/gpl-2.0.txt
- LGPL: GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. http://artlibre.org/licence/lal/de
- CFR: Copyright free use.

- EPL: Eclipse Public License. `http://www.eclipse.org/org/documents/epl-v10.php`

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[32]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

32 Kapitel 30 auf Seite 105

30 Licenses

30.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from

conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

30.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free"in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Versionöf the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A SSecondary Sectionïs a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The Ïnvariant Sectionsäre certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Textsäre certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent"copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparentïs called Öpaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page"means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page"means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher"means any person or entity that distributes copies of the Document to the public.

A section Ëntitled XYZ"means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as Äcknowledgements", "Dedications", Ëndorsements", or "History".) To "Preserve the Titleöf such a section when you modify the Document means that it remains a section Ëntitled XYZäccording to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "Historyïn the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History§ection. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. * K. For any section Entitled Äcknowledgementsör "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled Ëndorsements". Such a section may not be included in the Modified Version. * N. Do not retitle any existing section to be Entitled Ëndorsementsör to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled Ëndorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "Historyïn the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled Äcknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled Ëndorsements". 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ägg regateïf the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled Äcknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ör any later versionäpplies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

"Massive Multiauthor Collaboration Site"(or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration"(or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA"means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

Ïncorporate"means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is ëligible for relicensingïf it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts."line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

30.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below. 0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or * b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the Combined Work with a copy of the GNU GPL and this license document. * c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. * d) Do one of the following: o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user’s computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License. * b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.