

Thomas Theis



Mit  
Kapitel zu  
Raspberry  
Pi

# Einstieg in Python

Ideal für Programmieranfänger geeignet

4. Auflage

- ▶ Schritt für Schritt eigene Programme entwickeln
- ▶ Mit vielen Beispielen und Übungsaufgaben
- ▶ GUI, OOP, Datenbank- und Internetanwendungen u. v. m.



Python-Versionen 3.4 und 2.7  
sowie alle Code-Beispiele des Buchs



Galileo Computing

## Liebe Leserin, lieber Leser,

mit Python haben Sie eine gute Wahl getroffen: Python ist eine vielseitige Programmiersprache, leicht zu lernen und bietet Ihnen später auch das Potenzial für anspruchsvollere Projekte.

Dieses Buch wird Sie bei Ihrem Einstieg in Python von Anfang an begleiten. Schritt für Schritt lernen Sie das Programmieren mit Python. Sie werden schnell Ihre ersten eigenen Programme schreiben, selbst wenn Sie bisher noch nicht programmiert haben. Die zahlreichen Übungsaufgaben im Buch helfen Ihnen dabei, Ihr neu gewonnenes Wissen zu testen und weiter zu vertiefen.

Ein Hinweis vorab zur Version: Python ist seit der Version 3.0 nicht mehr zu 100 % abwärts kompatibel. Das Buch trägt diesem Umstand Rechnung, indem es an den entsprechenden Stellen auf die Unterschiede zwischen Python 3 und Python 2 hinweist. So können Sie später auch problemlos ältere Python-Programme pflegen und weiterentwickeln.

Dieses Buch wurde mit großer Sorgfalt geschrieben, geprüft und produziert. Sollte dennoch einmal etwas nicht so funktionieren, wie Sie es erwarten, freue ich mich, wenn Sie sich mit mir in Verbindung setzen. Ihre Kritik und konstruktiven Anregungen, aber natürlich auch Ihr Lob sind uns jederzeit herzlich willkommen!

Viel Spaß mit Python wünscht Ihnen nun

**Ihre Anne Scheibe**

Lektorat Galileo Computing

[anne.scheibe@galileo-press.de](mailto:anne.scheibe@galileo-press.de)

[www.galileocomputing.de](http://www.galileocomputing.de)

Galileo Press · Rheinwerkallee 4 · 53227 Bonn

# Hinweise zur Benutzung

Dieses E-Book ist **urheberrechtlich geschützt**. Mit dem Erwerb des E-Books haben Sie sich verpflichtet, die Urheberrechte anzuerkennen und einzuhalten. Sie sind berechtigt, dieses E-Book für persönliche Zwecke zu nutzen. Sie dürfen es auch ausdrucken und kopieren, aber auch dies nur für den persönlichen Gebrauch. Die Weitergabe einer elektronischen oder gedruckten Kopie an Dritte ist dagegen nicht erlaubt, weder ganz noch in Teilen. Und auch nicht eine Veröffentlichung im Internet oder in einem Firmennetzwerk.

Die ausführlichen und rechtlich verbindlichen Nutzungsbedingungen lesen Sie im Abschnitt [Rechtliche Hinweise](#).

Dieses E-Book-Exemplar ist mit einem **digitalen Wasserzeichen** versehen, einem Vermerk, der kenntlich macht, welche Person dieses Exemplar nutzen darf:

Exemplar Nr. 5mif-8vhz-rca2-6pxg  
zum persönlichen Gebrauch für  
Peter Christen,  
CSS GmbH,  
peterch@chp.ch

# Impressum

Dieses E-Book ist ein Verlagsprodukt, an dem viele mitgewirkt haben, insbesondere:

**Lektorat** Anne Scheibe

**Korrektorat** Monika Paff, Langenfeld

**Herstellung E-Book** Norbert Englert

**Covergestaltung** Barbara Thoben, Köln

**Satz E-Book** Typography & Computer, Krefeld

Wir hoffen sehr, dass Ihnen dieses Buch gefallen hat. Bitte teilen Sie uns doch Ihre Meinung mit und lesen Sie weiter auf den [Serviceseiten](#).

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

**ISBN 978-3-8362-2861-9 (Buch inkl. E-Book)**

**ISBN 978-3-8362-2983-8 (E-Book)**

4., aktualisierte und erweiterte Auflage 2014

© Galileo Press, Bonn 2014

# Inhalt

<b>1 Einführung</b>	<b>17</b>
<b>1.1 Vorteile von Python</b>	17
<b>1.2 Verbreitung von Python</b>	18
<b>1.3 Aufbau des Buchs</b>	18
<b>1.4 Übungen</b>	20
<b>1.5 Installation von Python unter Windows</b>	20
<b>1.6 Installation von Python unter Ubuntu Linux</b>	21
<b>1.7 Installation von Python unter OS X</b>	21
<b>2 Erste Schritte</b>	<b>23</b>
<b>2.1 Python als Taschenrechner</b>	23
<b>2.1.1 Eingabe von Berechnungen</b>	23
<b>2.1.2 Addition, Subtraktion und Multiplikation</b>	24
<b>2.1.3 Division, Ganzzahldivision und Modulo</b>	24
<b>2.1.4 Rangfolge und Klammern</b>	25
<b>2.1.5 Variablen und Zuweisung</b>	26
<b>2.2 Erstes Programm</b>	28
<b>2.2.1 Hallo Welt</b>	28
<b>2.2.2 Eingabe eines Programms</b>	28
<b>2.3 Speichern und Ausführen</b>	29
<b>2.3.1 Speichern</b>	29
<b>2.3.2 Ausführen unter Windows</b>	30
<b>2.3.3 Ausführen unter Ubuntu Linux und unter OS X</b>	32
<b>2.3.4 Kommentare</b>	33
<b>2.3.5 Verkettung von Ausgaben</b>	34
<b>2.3.6 Lange Ausgaben</b>	34

<b>3 Programmierkurs</b>	37
<b>3.1 Ein Spiel programmieren</b>	37
3.1.1 Das fertige Spiel .....	37
3.1.2 Der Weg zum fertigen Spiel .....	37
3.1.3 Mögliche Erweiterungen .....	38
<b>3.2 Variablen und Operatoren</b>	38
3.2.1 Berechnung und Zuweisung .....	38
3.2.2 Eingabe einer Zeichenkette .....	39
3.2.3 Eingabe einer Zahl .....	39
3.2.4 Spiel, Version mit Eingabe .....	41
3.2.5 Zufallszahlen .....	42
<b>3.3 Verzweigungen</b>	44
3.3.1 Vergleichsoperatoren .....	44
3.3.2 Einfache Verzweigung .....	44
3.3.3 Spiel, Version mit Bewertung der Eingabe .....	45
3.3.4 Mehrfache Verzweigung .....	47
3.3.5 Logische Operatoren .....	48
3.3.6 Mehrere Vergleichsoperatoren .....	50
3.3.7 Spiel, Version mit genauer Bewertung der Eingabe .....	51
3.3.8 Rangfolge der Operatoren .....	52
<b>3.4 Schleifen</b>	53
3.4.1 for-Schleife .....	54
3.4.2 Schleifenabbruch mit »break« .....	54
3.4.3 Geschachtelte Kontrollstrukturen .....	55
3.4.4 Spiel, Version mit for-Schleife und Abbruch .....	56
3.4.5 for-Schleife mit »range()« .....	58
3.4.6 Spiel, Version mit »range()« .....	61
3.4.7 while-Schleife .....	62
3.4.8 Spiel, Version mit while-Schleife und Zähler .....	63
<b>3.5 Entwicklung eines Programms</b>	65
<b>3.6 Fehler und Ausnahmen</b>	66
3.6.1 Basisprogramm .....	66
3.6.2 Fehler abfangen .....	67
3.6.3 Eingabe wiederholen .....	69
3.6.4 Exkurs: Schleifenfortsetzung mit »continue« .....	70
3.6.5 Spiel, Version mit Ausnahmebehandlung .....	71

<b>3.7 Funktionen und Module .....</b>	73
3.7.1 Einfache Funktionen .....	74
3.7.2 Funktionen mit einem Parameter .....	75
3.7.3 Funktionen mit mehreren Parametern .....	77
3.7.4 Funktionen mit Rückgabewert .....	77
3.7.5 Spiel, Version mit Funktionen .....	78
<b>3.8 Das fertige Spiel .....</b>	80

---

## 4 Datentypen

---

<b>4.1 Zahlen .....</b>	85
4.1.1 Ganze Zahlen .....	85
4.1.2 Zahlen mit Nachkommastellen .....	87
4.1.3 Operator ** .....	88
4.1.4 Rundung und Konvertierung .....	88
4.1.5 Modul »math« .....	90
4.1.6 Bitoperatoren .....	91
4.1.7 Brüche .....	94
<b>4.2 Zeichenketten .....</b>	97
4.2.1 Eigenschaften .....	97
4.2.2 Operatoren .....	98
4.2.3 Operationen .....	99
4.2.4 Funktionen .....	102
4.2.5 Umwandlung von einer Zeichenkette in eine Zahl .....	105
4.2.6 Umwandlung von einer Zahl in eine Zeichenkette .....	107
4.2.7 Datentyp »bytes« .....	108
<b>4.3 Listen .....</b>	109
4.3.1 Eigenschaften .....	109
4.3.2 Operatoren .....	111
4.3.3 Funktionen und Operationen .....	112
<b>4.4 Tupel .....</b>	115
4.4.1 Eigenschaften .....	115
4.4.2 Operationen .....	116
4.4.3 Tupel entpacken .....	117

<b>4.5 Dictionary</b>	120
4.5.1 Eigenschaften	120
4.5.2 Funktionen	121
4.5.3 Views	122
4.5.4 Vergleiche	124
<b>4.6 Mengen, Sets</b>	125
4.6.1 Eigenschaften	125
4.6.2 Funktionen	126
4.6.3 Operatoren	128
4.6.4 Frozenset	130
<b>4.7 Wahrheitswerte und Nichts</b>	131
4.7.1 Wahrheitswerte True und False	131
4.7.2 Nichts, None	134
<b>4.8 Referenz, Identität und Kopie</b>	136
4.8.1 Referenz und Identität	136
4.8.2 Ressourcen sparen	138
4.8.3 Objekte kopieren	139

## 5 Weiterführende Programmierung

---

<b>5.1 Allgemeines</b>	141
5.1.1 Kombinierte Zuweisungsoperatoren	141
5.1.2 Programmzeile in mehreren Zeilen	143
5.1.3 Eingabe mit Hilfestellung	144
5.1.4 Anweisung »pass«	145
5.1.5 Funktionen »eval()« und »exec()«	147
<b>5.2 Ausgabe und Formatierung</b>	148
5.2.1 Funktion »print()«	148
5.2.2 Formatierte Ausgabe mit »format()«	151
5.2.3 Formatierte Ausgabe wie in C	155
<b>5.3 Conditional Expression</b>	156
<b>5.4 Iterierbare Objekte</b>	157
5.4.1 Funktion »zip()«	157
5.4.2 Funktion »map()«	158
5.4.3 Funktion »filter()«	160

<b>5.5 List Comprehension .....</b>	161
<b>5.6 Fehler und Ausnahmen .....</b>	163
5.6.1 Allgemeines .....	163
5.6.2 Syntaxfehler .....	163
5.6.3 Laufzeitfehler .....	165
5.6.4 Logische Fehler und Debugging .....	165
5.6.5 Fehler erzeugen .....	170
5.6.6 Unterscheidung von Ausnahmen .....	171
<b>5.7 Funktionen .....</b>	172
5.7.1 Variable Anzahl von Parametern .....	173
5.7.2 Benannte Parameter .....	174
5.7.3 Voreinstellung von Parametern .....	175
5.7.4 Mehrere Rückgabewerte .....	176
5.7.5 Übergabe von Kopien und Referenzen .....	177
5.7.6 Lokal, global .....	180
5.7.7 Lambda-Funktion .....	181
5.7.8 Funktionsname als Parameter .....	182
<b>5.8 Eingebaute Funktionen .....</b>	183
5.8.1 Funktionen »max()«, »min()« und »sum()« .....	185
5.8.2 Funktionen »chr()« und »ord()« .....	186
5.8.3 Funktionen »reversed()« und »sorted()« .....	187
<b>5.9 Statistikfunktionen .....</b>	188
<b>5.10 Eigene Module .....</b>	190
5.10.1 Eigene Module erzeugen .....	190
5.10.2 Eigene Module verwenden .....	190
<b>5.11 Parameter der Kommandozeile .....</b>	192
5.11.1 Übergabe von Zeichenketten .....	192
5.11.2 Übergabe von Zahlen .....	193
5.11.3 Beliebige Anzahl von Parametern .....	193
<hr/> <b>6 Objektorientierte Programmierung .....</b>	195
<b>6.1 Was ist OOP? .....</b>	195
<b>6.2 Klassen, Objekte und eigene Methoden .....</b>	196

<b>6.3 Konstruktor und Destruktor .....</b>	198
<b>6.4 Besondere Methoden .....</b>	200
<b>6.5 Operatormethoden .....</b>	201
<b>6.6 Referenz, Identität und Kopie .....</b>	202
<b>6.7 Vererbung .....</b>	204
<b>6.8 Mehrfachvererbung .....</b>	207
<b>6.9 Enumerationen .....</b>	209
<b>6.10 Spiel, objektorientierte Version .....</b>	211
<b>7 Verschiedene Module .....</b>	215
<hr/>	
<b>7.1 Datum und Zeit .....</b>	215
7.1.1 Spielen mit Zeitangabe .....	215
7.1.2 Aktuelle Zeit ausgeben .....	215
7.1.3 Zeitangabe erzeugen .....	219
7.1.4 Mit Zeitangaben rechnen .....	220
7.1.5 Programm anhalten .....	222
7.1.6 Spiel, Version mit Zeitmessung .....	223
7.1.7 Spiel, objektorientierte Version mit Zeitmessung .....	225
<b>7.2 Modul »collections« .....</b>	226
<b>7.3 Multithreading .....</b>	228
7.3.1 Wozu dient Multithreading? .....	229
7.3.2 Erzeugung eines Threads .....	229
7.3.3 Identifizierung eines Threads .....	230
7.3.4 Gemeinsame Objekte .....	232
7.3.5 Threads und Exceptions .....	233
<b>7.4 Reguläre Ausdrücke .....</b>	234
7.4.1 Suchen von Teiltexten .....	235
7.4.2 Ersetzen von Teiltexten .....	239
<b>7.5 Audio-Ausgabe .....</b>	241

<b>8 Dateien</b>	243
<b>8.1 Dateitypen</b>	243
<b>8.2 Öffnen und Schließen einer Datei</b>	244
<b>8.3 Sequentielle Dateien</b>	245
8.3.1 Sequentielles Schreiben	245
8.3.2 Sequentielles Lesen	247
8.3.3 CSV-Datei schreiben	252
8.3.4 CSV-Datei lesen	254
<b>8.4 Dateien mit festgelegter Struktur</b>	256
8.4.1 Formatiertes Schreiben	256
8.4.2 Lesen an beliebiger Stelle	258
8.4.3 Schreiben an beliebiger Stelle	259
<b>8.5 Serialisierung</b>	261
8.5.1 Objekte in Datei schreiben	261
8.5.2 Objekte aus Datei lesen	263
<b>8.6 Bearbeitung mehrerer Dateien</b>	264
<b>8.7 Informationen über Dateien</b>	266
<b>8.8 Dateien und Verzeichnisse verwalten</b>	267
<b>8.9 Beispielprojekt Morsezeichen</b>	269
8.9.1 Morsezeichen aus Datei lesen	269
8.9.2 Ausgabe auf dem Bildschirm	271
8.9.3 Ausgabe mit Tonsignalen	272
<b>8.10 Spiel, Version mit Highscore-Datei</b>	274
8.10.1 Eingabebeispiel	274
8.10.2 Aufbau des Programms	275
8.10.3 Code des Programms	275
<b>8.11 Spiel, objektorientierte Version mit Highscore-Datei</b>	280
<b>9 Internet</b>	285
<b>9.1 Laden und Senden von Internetdaten</b>	285
9.1.1 Daten lesen	286
9.1.2 Daten kopieren	289

9.1.3	Daten senden per GET .....	289
9.1.4	Daten senden per POST .....	293
<b>9.2</b>	<b>Webserver-Programmierung .....</b>	<b>296</b>
9.2.1	Erstes Programm .....	296
9.2.2	Beantworten einer Benutzereingabe .....	298
9.2.3	Formularelemente mit mehreren Werten .....	301
9.2.4	Typen von Formularelementen .....	303
<b>9.3</b>	<b>Browser aufrufen .....</b>	<b>309</b>
<b>9.4</b>	<b>Spiel, Version für das Internet .....</b>	<b>309</b>
9.4.1	Eingabebeispiel .....	309
9.4.2	Aufbau des Programms .....	311
9.4.3	Code des Programms .....	312

---

## **10 Datenbanken**

---

<b>10.1</b>	<b>Aufbau von Datenbanken .....</b>	<b>319</b>
<b>10.2</b>	<b>SQLite .....</b>	<b>320</b>
10.2.1	Datenbank, Tabelle und Datensätze .....	320
10.2.2	Daten anzeigen .....	323
10.2.3	Daten auswählen, Operatoren .....	324
10.2.4	Operator »LIKE« .....	326
10.2.5	Sortierung der Ausgabe .....	328
10.2.6	Auswahl nach Eingabe .....	329
10.2.7	Datensätze ändern .....	331
10.2.8	Datensätze löschen .....	333
<b>10.3</b>	<b>SQLite auf dem Webserver .....</b>	<b>334</b>
<b>10.4</b>	<b>MySQL .....</b>	<b>336</b>
10.4.1	XAMPP und Connector / Python .....	337
10.4.2	Datenbank erzeugen .....	337
10.4.3	Tabelle anlegen .....	338
10.4.4	Datensätze anlegen .....	340
10.4.5	Daten anzeigen .....	341
<b>10.5</b>	<b>Spiel, Version mit Highscore-Datenbank .....</b>	<b>342</b>
<b>10.6</b>	<b>Spiel, objektorientierte Version mit Highscore-Datenbank .....</b>	<b>345</b>

# 11 Benutzeroberflächen

347

---

<b>11.1 Einführung .....</b>	347
11.1.1 Eine erste GUI-Anwendung .....	347
11.1.2 Ändern von Eigenschaften .....	349
<b>11.2 Widget-Typen .....</b>	351
11.2.1 Anzeigefeld, Label .....	351
11.2.2 Einzelige Textbox, Entry .....	354
11.2.3 Versteckte Eingabe .....	356
11.2.4 Mehrzeilige Textbox, Text .....	358
11.2.5 Scrollende Textbox, ScrolledText .....	360
11.2.6 Listbox mit einfacher Auswahl .....	361
11.2.7 Listbox mit mehrfacher Auswahl .....	363
11.2.8 Scrollbar, scrollende Widgets .....	365
11.2.9 Radiobuttons zur Auswahl, Widget-Variablen .....	367
11.2.10 Radiobuttons zur Auswahl und Ausführung .....	369
11.2.11 Checkbuttons zur mehrfachen Auswahl .....	371
11.2.12 Schieberegler, Scale .....	373
11.2.13 Mausereignisse .....	375
11.2.14 Tastaturereignisse .....	379
<b>11.3 Geometrische Anordnung von Widgets .....</b>	381
11.3.1 Frame-Widget, Methode »pack()« .....	381
11.3.2 Ein einfacher Taschenrechner .....	384
11.3.3 Methode »grid()« .....	388
11.3.4 Methode »place()«, absolute Koordinaten .....	390
11.3.5 Methode »place()«, relative Koordinaten .....	391
11.3.6 Absolute Veränderung von Koordinaten .....	393
11.3.7 Relative Veränderung von Koordinaten .....	395
<b>11.4 Menüs, Messageboxen und Dialogfelder .....</b>	398
11.4.1 Menüleisten .....	398
11.4.2 Kontextmenüs .....	403
11.4.3 Messageboxen .....	407
11.4.4 Eigene Dialogfelder .....	411
11.4.5 Ausführung verhindern .....	413
<b>11.5 Spiel, GUI-Version .....</b>	414

<b>12 Neues in Python 3</b>	421
<b>12.1 Neue und geänderte Eigenschaften</b>	421
12.1.1 Auffällige Änderungen	421
12.1.2 Weitere Änderungen	422
<b>12.2 Konvertierung von Python 2 zu Python 3</b>	423
<b>13 Raspberry Pi</b>	425
<b>13.1 Einzelteile und Installation</b>	425
13.1.1 Einzelteile	425
13.1.2 Sicherheit und Schäden	427
13.1.3 Zusammenbau	428
13.1.4 Installation	429
13.1.5 Update	429
13.1.6 WLAN und Internet	430
<b>13.2 Elektronische Schaltungen</b>	430
13.2.1 Gleichspannungs-Stromkreis	430
13.2.2 Spannung ist Information	431
13.2.3 Bauelemente und Ausrüstung	431
13.2.4 Widerstände	432
<b>13.3 Aufbau des GPIO-Anschlusses</b>	434
<b>13.4 Leuchtdiode</b>	436
13.4.1 Schaltung	436
13.4.2 LED ein- und ausschalten	437
13.4.3 LED blinken lassen	437
13.4.4 LED blinkt Morsezeichen	438
<b>13.5 Taster</b>	439
13.5.1 Schaltung	439
13.5.2 Einfache Prüfung des Tasters	440
13.5.3 Verbesserte Prüfung des Tasters	441
<b>13.6 Messwerte ermitteln</b>	442
13.6.1 Schaltung	442
13.6.2 Programm zum Ermitteln der Messwerte	443
13.6.3 Vorbereitung	443

<b>13.7 Messwerte in Datenbank speichern .....</b>	<b>444</b>
13.7.1 Datenbank erstellen .....	445
13.7.2 Messen und Speichern .....	445
13.7.3 Anzeige der Daten .....	447
<b>Anhang</b>	<b>449</b>
<b>A.1 Installation von XAMPP .....</b>	<b>449</b>
A.1.1 Installation von XAMPP unter Windows .....	449
A.1.2 Installation von XAMPP unter Ubuntu Linux .....	450
A.1.3 Installation von XAMPP unter OS X .....	451
<b>A.2 UNIX-Befehle .....</b>	<b>451</b>
A.2.1 Inhalt eines Verzeichnisses .....	452
A.2.2 Verzeichnis anlegen, wechseln und löschen .....	452
A.2.3 Datei kopieren, verschieben und löschen .....	453
<b>A.3 Lösungen .....</b>	<b>454</b>
A.3.1 Lösungen zu Kapitel 2 .....	454
A.3.2 Lösungen zu Kapitel 3 .....	455
A.3.3 Lösungen zu Kapitel 5 .....	461
<b>Index .....</b>	<b>463</b>



# Kapitel 1

## Einführung

In diesem Kapitel stelle ich Ihnen Python kurz vor. Sie lernen die Vorteile von Python kennen und erfahren, wie Sie Python unter Windows, unter Ubuntu Linux und unter OS X installieren.

### 1.1 Vorteile von Python

Python ist eine sehr einfach zu erlernende Programmiersprache und für den Einstieg in die Welt der Programmierung ideal geeignet. Trotz ihrer Einfachheit bietet diese Sprache auch die Möglichkeit, komplexe Programme für vielfältige Anwendungsgebiete zu schreiben.

Python eignet sich besonders zur schnellen Entwicklung umfangreicher Anwendungen. Diese Technik ist unter dem Stichwort RAD (= *Rapid Application Development*) bekannt geworden. Python vereint zu diesem Zweck folgende Vorteile:

- ▶ Eine einfache, eindeutige Syntax: Python ist eine ideale Programmiersprache für Einsteiger. Sie beschränkt sich auf einfache, klare Anweisungen. In anderen Programmiersprachen werden vielfältige Lösungswege für das gleiche Problem angeboten, sodass der Entwickler weniger geleitet als unsicher wird. Python beschränkt sich dagegen häufig auf einen einzigen möglichen Lösungsweg. Dieser prägt sich schnell ein und wird dem Entwickler vertraut. Leicht zu lernen
- ▶ Klare Strukturen: Python zwingt den Entwickler, in einer gut lesbaren Struktur zu schreiben. Die Anordnung der Programmzeilen ergibt gleichzeitig die logische Struktur des Programms. Einfach
- ▶ Wiederverwendung von Code: Die Modularisierung, also die Zerlegung eines Problems in Teilprobleme und die anschließende Zusammenführung der Teillösungen zu einer Gesamtlösung, wird in Python sehr leicht gemacht. Die vorhandenen Teillösungen können unkompliziert für weitere Aufgabenstellungen genutzt werden, sodass Sie als Entwickler bald über einen umfangreichen Pool an Modulen verfügen. Klar
- ▶ Objektbearbeitung: In Python werden alle Daten als Objekte gespeichert. Dies führt zu einer einheitlichen Datenbehandlung für Objekte unterschiedlichen Typs. Andererseits erfolgt die physikalische Speicherung der Objekte von Python automatisch, also ohne Eingriff des Entwicklers. Der Entwickler muss sich nicht um die Reservierung und Freigabe geeigneter Speicherbereiche kümmern. Wiederverwendbar
- ▶ Einheitliche Objekte

- |                      |  |
|----------------------|--|
| <b>Interpretiert</b> | ► Interpreter/Compiler: Python-Programme werden unmittelbar interpretiert. Sie müssen nicht, wie bei vielen anderen Programmiersprachen, erst kompiliert und gebunden werden. Dies ermöglicht einen häufigen, schnellen Wechsel zwischen Codierungs- und Testphase.                  |
| <b>Unabhängig</b>    | ► Unabhängigkeit vom Betriebssystem: Sowohl Programme, die von der Kommandozeile aus bedient werden, als auch Programme mit grafischen Benutzeroberflächen können auf unterschiedlichen Betriebssystemen (Windows, Linux, OS X) ohne Neuentwicklung und Anpassung eingesetzt werden. |

## 1.2 Verbreitung von Python

Aufgrund seiner vielen Vorteile gehört Python zu den beliebtesten Programmiersprachen überhaupt. So wird es z. B. innerhalb des Projekts *100-Dollar-Laptop*, das der Schulausbildung von Kindern in aller Welt dient, für die Benutzeroberfläche verwendet. Aber auch in zahlreichen großen Unternehmen wird Python eingesetzt, hier ein paar Beispiele:

- Google: Python ist eine der drei offiziellen Programmiersprachen. (Guido van Rossum, der Entwickler von Python, arbeitet für Google.)
- YouTube: Wurde zum großen Teil mithilfe von Python entwickelt.
- NASA: Nutzt Python zur Softwareentwicklung im Zusammenhang mit den Space-Shuttle-Missionen.
- Industrial Light & Magic: Auch Hollywood setzt auf Python – die Produktionsfirma ILM (Star Wars, Indiana Jones, Fluch der Karibik) nutzt es z. B. bei der Entwicklung von Spezialeffekten.
- Honeywell: Python wird weltweit in vielen Firmen zur allgemeinen Hardware- und Softwareentwicklung eingesetzt.

## 1.3 Aufbau des Buchs

- |                         |  |
|-------------------------|--|
| <b>Aktiv lernen</b>     | Das vorliegende Buch soll Sie in die Programmiersprache Python einführen. Es wird besonderer Wert darauf gelegt, dass Sie selbst praktisch mit Python arbeiten. Daher möchte ich Sie von Anfang an dazu auffordern, dem logischen Faden von Erklärungen und Beispielen zu folgen.  |
| <b>Python-Versionen</b> | Python 3 ist die neuere Version der Programmiersprache und an vielen Stellen nicht abwärtskompatibel mit der klassischen Version Python 2. Allerdings ist Python 2 weiterhin sehr verbreitet und wird noch auf Jahre hinaus von den Python-Entwicklern unterstützt werden. Alle Erläuterungen und Programme dieses Buchs liegen für beide Versionen vor, damit stehen Ihnen beide Mög- |

lichkeiten offen. Auf die Unterschiede der beiden Versionen gehe ich besonders ein, sodass ein späterer Umstieg von der einen zur anderen Version leichtfällt.

Erste Zusammenhänge werden in [Kapitel 2](#), »Erste Schritte«, anhand von einfachen Berechnungen vermittelt. Außerdem lernen Sie, ein Programm einzugeben, zu speichern und es unter den verschiedenen Umgebungen auszuführen.

Sie sollen die Sprache spielerisch kennenlernen. Daher wird Sie ein selbst programmiertes Spiel durch das Buch begleiten. Dieses Spiel wird mit dem »Programmierkurs« in [Kapitel 3](#) eingeführt und im weiteren Verlauf des Buchs kontinuierlich erweitert und verbessert.

Nach der Vorstellung der verschiedenen Datentypen mit ihren jeweiligen Eigenschaften und Vorteilen in [Kapitel 4](#), »Datentypen«, werden die Programmierkenntnisse in [Kapitel 5](#), »Weiterführende Programmierung«, vertieft.

[Kapitel 6](#), »Objektorientierte Programmierung«, widmet sich der objektorientierten Programmierung mit Python.

**Spielerisch lernen**

**Objektorientiert**

Einige nützliche Module zur Ergänzung der Programme werden in [Kapitel 7](#), »Verschiedene Module«, vorgestellt.

In den Kapiteln 8, »Dateien«, und 10, »Datenbanken«, lernen Sie, Daten dauerhaft in Dateien oder Datenbanken zu speichern. Python wird zudem in der Internetprogrammierung eingesetzt. Die Zusammenhänge zwischen Python und dem Internet vermittelt [Kapitel 9](#), »Internet«.

Sowohl Windows als auch Ubuntu Linux und OS X bieten komfortable grafische Benutzeroberflächen (GUIs). [Kapitel 11](#), »Benutzeroberflächen«, beschäftigt sich mit der GUI-Erzeugung mithilfe des Moduls `tkinter`. Dieses stellt eine Schnittstelle zwischen dem grafischen Toolkit `Tk` und Python dar.

**Grafische Oberflächen**

Python gibt es in den Versionen 2 und 3, die parallel nebeneinander bestehen. Sofern es keine Gründe gibt, die dagegensprechen, starten Sie am besten mit Python 3, das auch den Beispielen zugrunde liegt. Auf etwaige Abweichungen der Version 2 weise ich Sie in den Erläuterungen und Programmen in diesem Buch aber natürlich jederzeit hin. [Kapitel 12](#) fasst die wesentlichen Unterschiede zwischen Python 2 und 3, die besonders für Einsteiger interessant sind, dann noch einmal übersichtlich zusammen.

**Python-Versionen**

Python ist die primäre Sprache für die Programmierung von elektronischen Schaltungen mithilfe des Einkartencomputers Raspberry Pi. In [Kapitel 13](#) werden wir uns damit beschäftigen.

Für die Hilfe bei der Erstellung dieses Buchs bedanke ich mich bei dem ganzen Team von Galileo Press, ganz besonders bei Anne Scheibe.

## 1.4 Übungen

In einigen Kapiteln finden Sie Übungsaufgaben, die Sie unmittelbar lösen sollten. Auf diese Weise können Sie Ihre Kenntnisse prüfen, bevor Sie zum nächsten Thema übergehen. Die Lösungen der Übungsaufgaben finden Sie in Anhang A.3. Dabei ist Folgendes zu beachten:

- |                           |  |
|---------------------------|--|
| <b>Viele Lösungen</b>     | ► Es gibt für jedes Problem viele richtige Lösungen. Sie sollten sich also nicht davon beunruhigen lassen, dass Ihre Lösung eventuell nicht genauso aussieht wie die angegebene. Betrachten Sie die angegebene Lösung vielmehr als Anregung, was Sie anders und gegebenenfalls besser machen können.                     |
| <b>Aus Fehlern lernen</b> | ► Bei der eigenen Lösung der Aufgaben wird sicherlich der eine oder andere Fehler auftreten – lassen Sie sich dadurch nicht entmutigen ...<br>► ... denn nur aus Fehlern kann man lernen. Auf die vorgeschlagene Art und Weise werden Sie Python wirklich erlernen – nicht allein durch das Lesen von Programmierregeln. |

## 1.5 Installation von Python unter Windows

**Software auf Datenträger** Python ist eine frei verfügbare Programmiersprache, die auf verschiedenen Betriebssystemplattformen eingesetzt werden kann. Auf dem Datenträger zu diesem Buch finden sich die Versionen 2.7.6 und 3.4.0 für Windows. Sie können sie unter Windows 8, aber natürlich auch unter älteren Windows-Versionen installieren.

Die jeweils neuesten Python-Versionen können Sie von der offiziellen Python-Website <http://www.python.org> aus dem Internet laden. Die Version 2.7.x wird die letzte Version für Python 2 bleiben und von den Python-Entwicklern noch auf Jahre hinaus unterstützt werden.

**Installation** Rufen Sie zur Installation unter Windows die ausführbare Datei *python-3.4.0.msi* (bzw. *python-2.7.6.msi*) auf. Die Voreinstellungen des Installationsvorganges können Sie unverändert übernehmen. Dabei wird Python im Verzeichnis *C:\Python34* (bzw. *C:\Python27*) installiert. Anschließend verfügen Sie im Startmenü über einen Eintrag PYTHON 3.4 (bzw. PYTHON 2.7), siehe Abbildung 1.1.

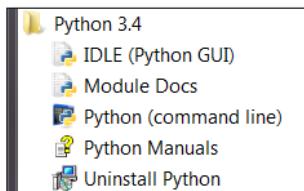


Abbildung 1.1 Startmenü Python

Falls Sie sich mit beiden Versionen beschäftigen möchten: kein Problem. Sie können parallel installiert und benutzt werden.

Der Eintrag **IDLE**, den Sie nach der Installation im Startmenü sehen, ist eine Entwicklungsumgebung, die selbst in Python geschrieben wurde und mit der Sie im Folgenden Ihre Programme schreiben werden.

**IDLE**

## 1.6 Installation von Python unter Ubuntu Linux

Stellvertretend für andere Linux-Distributionen wird in diesem Buch Ubuntu Linux 13.10 genutzt. Python 2 und Python 3 sind unter Ubuntu Linux bereits installiert. Sie sollten auch nicht deinstalliert werden.

**Ubuntu Linux**

Zur Installation der Entwicklungsumgebung IDLE starten Sie das UBUNTU SOFTWARE CENTER und geben als Suchbegriff ein: *Idle*. Es erscheinen die Einträge IDLE (VERWENDET PYTHON-3.3) und IDLE (VERWENDET PYTHON-2.7).

**IDLE**

Es ist kein Problem, beide Versionen von Python parallel zu installieren bzw. zu benutzen. Das Gleiche gilt für die beiden Versionen von IDLE. Sie erscheinen nach der Installation im Starter.

## 1.7 Installation von Python unter OS X

Nachfolgend wird die Installation von Python auf einem Mac unter OS X 10.9 Mavericks beschrieben. Auf dem Datenträger zu diesem Buch finden sich die Versionen 2.7.6 und 3.4.0 für OS X. Die jeweils neuesten Versionen können Sie von der offiziellen Python-Website <http://www.python.org> aus dem Internet laden.

**OS X**

Auf der Python-Website wird darauf hingewiesen, dass es bei bestimmten Versionen von OS X Probleme mit der Schnittstelle tkinter zur Bibliothek Tk gibt, mit der Sie Programme mit Benutzeroberflächen erschaffen können, siehe Kapitel 11. Abhilfe schafft hier das Paket ActiveTcl von ActiveState. Eine Version des Pakets liegt auch auf dem Datenträger.

Führen Sie auf den **DMG**-Dateien einen Doppelklick aus. Es wird jeweils ein neues Laufwerk angelegt, mit dem Namen PYTHON 2.7.6 bzw. PYTHON 3.4.0. Auf dem Laufwerk gibt es eine **MPKG**-Datei, die Sie über den Kontextmenüpunkt ÖFFNEN aufrufen können. Damit startet die Installation, das Programm landet im Verzeichnis *Programme/Python 2.7* bzw. *Programme/Python 3.4*.

**DMG-Datei**

In diesem Verzeichnis finden Sie einen Eintrag für IDLE. Ziehen Sie diesen auf den Desktop. Damit legen Sie jeweils eine Verknüpfung mit dem Namen IDLE an, die Sie passend umbenennen sollten, z. B. IDLE 2.7.6 und IDLE 3.4.0. Zu guter Letzt können Sie das Laufwerk auswerfen.

**IDLE**



# Kapitel 2

## Erste Schritte

In diesem Kapitel werden Sie Python zum ersten Mal einsetzen – zunächst als Taschenrechner. Außerdem lernen Sie, ein Programm einzugeben, zu speichern und auszuführen. Alle Erläuterungen beziehen sich zunächst auf Python 3 unter Windows. Falls es Unterschiede zu Python 2, zu Linux oder zu OS X gibt, werden sie jeweils im gleichen Abschnitt direkt anschließend erwähnt.

### 2.1 Python als Taschenrechner

Sie können Python zunächst wie einen einfachen Taschenrechner benutzen. Dies erleichtert Ihnen den Einstieg in Python.

#### 2.1.1 Eingabe von Berechnungen

Rufen Sie IDLE für Python auf:

- ▶ Unter Windows erreichen Sie IDLE über START • ALLE PROGRAMME • PYTHON 3.4 (bzw. PYTHON 2.7) • IDLE (PYTHON GUI).
- ▶ Unter Ubuntu Linux liegen beide Versionen von IDLE im Starter. Alternativ können Sie auch im DASH den Begriff *Idle* eingeben.
- ▶ Unter OS X können Sie die beiden Verknüpfungen nutzen, die Sie bei der Installation angelegt haben. Ansonsten finden Sie IDLE über den Finder im Verzeichnis *Programme/Python 2.7* bzw. *Programme/Python 3.4*.

Die Entwicklungsumgebung IDLE wird auch *Python Shell* genannt und kann sowohl als Editor zur Eingabe der Programme als auch als einfacher Taschenrechner genutzt werden. Eine Darstellung von IDLE unter Windows sehen Sie in Abbildung 2.1.

Python Shell

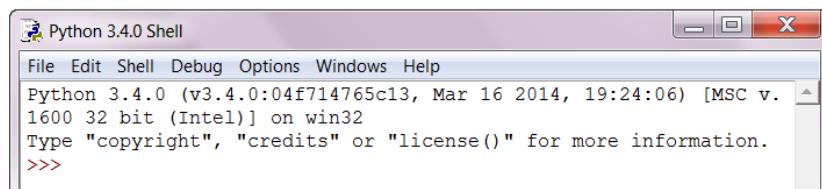


Abbildung 2.1 Python-Entwicklungsumgebung IDLE

Die Abbildungen in diesem Buch sind normalerweise für Python unter Windows erstellt worden. Sie gelten natürlich auch sinngemäß für Python unter Ubuntu Linux und Python unter OS X.

**Kleinere Berechnungen** Zur Durchführung kleinerer Berechnungen müssen Sie keine vollständigen Programme schreiben und starten. Sie können die gewünschten Rechenoperationen direkt an der Eingabestelle, erkennbar an den Zeichen >>>, eingeben. Eine abschließende Betätigung der Taste führt zur unmittelbaren Berechnung und Ausgabe des Ergebnisses.

Die dabei angewendeten Rechenregeln finden auch bei der Erstellung von Python-Programmen Verwendung. Es lohnt sich also eine genauere Betrachtung.

### 2.1.2 Addition, Subtraktion und Multiplikation

Im Folgenden werden Eingabe, Verarbeitung und Ausgabe einiger einfacher Berechnungen dargestellt:

```
>>> 41 + 7.5
48.5
>>> 12 - 18
-6
>>> 7 * 3
21
```

Zur Erläuterung:

**Operatoren +, -, \***

- ▶ Der Reihe nach werden die Operatoren + (Addition), – (Subtraktion) und \* (Multiplikation) eingesetzt. Wenn Sie die Taste betätigen, erscheint das Ergebnis jeweils in der Zeile darunter.
- ▶ Als Dezimalzeichen gilt der Punkt, wie bei der ersten Berechnung ( $41 + 7.5 = 48.5$ ) erkennbar.

### 2.1.3 Division, Ganzzahldivision und Modulo

Es folgen zwei Divisionsaufgaben:

```
>>> 22 / 8
2.75
>>> 22 / -8
-2.75
```

Zur Erläuterung:

**Operator /**

- ▶ Der Operator / dient zur normalen Division.

Seltener genutzt werden die beiden Operatoren für die Ganzzahldivision und die Modulo-Berechnung.

Zunächst zwei Beispiele zur Ganzzahldivision:

```
>>> 22 // 8
2
>>> 22 // -8
-3
```

Zur Erläuterung:

- ▶ Bei einer Ganzzahldivision wird das Ergebnis berechnet, indem
  - zunächst das exakte Ergebnis inklusive etwaiger Nachkommastellen ermittelt und
  - anschließend die nächstkleinere ganze Zahl ermittelt wird.

**Operator //**

Ein Beispiel zum Operator % (Modulo):

**Operator %**

**22 % 8**

**6**

Zur Erläuterung:

- ▶ Mit dem Operator % wird der Rest einer Ganzzahldivision berechnet.
- ▶ Die Ganzzahldivision »22 durch 8« ergibt »2 Rest 6«. Der Modulo-Operator liefert also den Rest 6.

### Unterschiede in Python 2

Der Operator / ergibt in Python 2 bereits eine Ganzzahldivision. Zur Vermeidung des Verlusts der Nachkommastellen können Sie eine der folgenden Änderungen durchführen:

- ▶ Hängen Sie ein .0 an die ganze Zahl im Zähler oder Nenner an, also z. B. 22.0 / 8 oder 22 / 8.0 statt 22 / 8.
- ▶ Falls der Zähler oder Nenner aus mehreren ganzen Zahlen besteht, so reicht es aus, .0 an eine Zahl des Zählers oder des Nenners anzuhängen, also z. B. (4.0 + 9) / (8 - 5) oder (4 + 9) / (8.0 - 5) statt (4 + 9) / (8 - 5).
- ▶ Falls die Division ohne Zahlen, nur mit Variablen durchgeführt wird, hilft eine Multiplikation des Zählers oder des Nenners oder eines Teils davon mit 1.0 weiter, also z. B. 1.0 \* x / (y + z) oder x / (1.0 \* y + z) statt x / (y + z).

**.0 anhängen**

Die genannten Änderungen verfälschen nicht das Rechenergebnis für Python 3, sind also »aufwärtskompatibel«. Der Operator // aus Python 3 für die Ganzzahldivision ist unter Python 2.7 bekannt und hat auch die gleiche Wirkung.

### 2.1.4 Rangfolge und Klammern

Es gilt, wie in der Mathematik, Punkt- vor Strichrechnung. Multiplikation und Division haben also Vorrang vor Addition und Subtraktion. Das Setzen von Klammern führt dazu, dass die Ausdrücke innerhalb der Klammern zuerst berechnet werden. Die folgenden Beispiele verdeutlichen die Rangfolge der Operatoren:

**Rangfolge**

```
>>> 7 + 2 * 3
13
>>> (7 + 2) * 3
27
```

**Listing 2.1 Rangfolge der Operatoren**

Zur Erläuterung: Bei der ersten Rechnung werden zunächst 2 und 3 multipliziert, anschließend wird 7 addiert. Bei der zweiten Rechnung werden zunächst 7 und 2 addiert, anschließend wird mit 3 multipliziert.

### Übung u\_grundrechenarten

Es wird vorausgesetzt, dass Python wie angegeben installiert wurde. Rufen Sie die Entwicklungsumgebung IDLE auf. Ermitteln Sie die Ergebnisse der folgenden vier Aufgaben:

$$\begin{aligned} 13 - 5 \times 2 + \frac{12}{6} \\ \underline{\frac{7}{2} - \frac{5}{4}} \\ \underline{\frac{12 - 5 \times 2}{4}} \\ \left( \frac{1}{2} - \frac{1}{4} + \frac{4+3}{8} \right) \times 2 \end{aligned}$$

Die Lösungen aller Übungsaufgaben finden Sie in Anhang A.3.

## 2.1.5 Variablen und Zuweisung

**Variablen zuweisen** Bisher haben wir nur mit Zahlen gerechnet. Python kann, wie jede andere Programmiersprache auch, Zahlen in Variablen speichern, falls sie im Verlauf einer Berechnung mehrmals benötigt werden.

**Dezimalpunkt** Dies soll bei der Umrechnung einer Entfernungsgabe von englischen Meilen in Kilometer gezeigt werden. Für die Umrechnung gilt: 1 Meile = 1.609,344 Meter = 1,609344 Kilometer. Beachten Sie, dass Nachkommastellen mit einem Dezimalpunkt abgetrennt werden.

```
>>> mi = 1.609344
>>> 2 * mi
3.218688
>>> 5 * mi
8.04672
>>> 22.5 * mi
36.21024
>>> 2.35 * mi
3.7819584000000006
```

Zur Erläuterung:

- ▶ Der Umrechnungsfaktor wird zunächst in der Variablen `mi` gespeichert. Auf diese Weise können mehrere Umrechnungen einfacher nacheinander durchgeführt werden. Anschließend werden die entsprechenden Kilometerwerte für 2 Meilen, 5 Meilen, 22,5 Meilen und für 2,35 Meilen berechnet und ausgegeben. Variable
- ▶ Die Speicherung des Umrechnungsfaktors geschieht mit einer Zuweisung (`mi = 1.609344`). Folglich erhält die Variable `mi` den Wert, der rechts vom Gleichheitszeichen steht (`1.609344`). Zuweisung

### Einige Hinweise

- ▶ In Python-Variablen können Sie ganze Zahlen, Zahlen mit Nachkommastellen, Zeichenketten (also Texte) und andere Objekte eines Programms mithilfe von Zuweisungen speichern. Runden
- ▶ Das Ergebnis der letzten Berechnung ist streng genommen mathematisch falsch. 2,35 Meilen entsprechen 3,7819584 Kilometer, ohne eine weitere 6 an der 16. Nachkommastelle. Der Grund für diesen Fehler: Zahlen mit Nachkommastellen können nicht mathematisch exakt gespeichert werden, im Gegensatz zu ganzen Zahlen. Es gibt einige Stellen im Buch, an denen dies auffallen wird. Bei den meisten Berechnungen ist allerdings die Abweichung so gering, dass die Auswirkungen in der Praxis zu vernachlässigen sind.
- ▶ Es ist üblich, den Wert in Kilometer so auszugeben, dass auf drei Stellen hinter dem Komma gerundet wird, also auf den Meter genau. Auf eine solche formatierte Ausgabe gehe ich später in Abschnitt 5.2.2 noch ein.

Den Namen einer Variablen können Sie unter Befolgung der folgenden Regeln weitgehend frei wählen:

- ▶ Er kann aus den Buchstaben a bis z, A bis Z, Ziffern oder dem Zeichen `_` (Unterstrich) bestehen.
- ▶ Er darf nicht mit einer Ziffer beginnen.
- ▶ Er darf keinem reservierten Wort der Programmiersprache Python entsprechen. Eine Liste der reservierten Wörter finden Sie auf der vorderen Außenklappe.
- ▶ Die Groß- und Kleinschreibung ist zu beachten: Namen und Anweisungen müssen stets exakt wie vorgegeben geschrieben werden. Die Namen `mi` und `Mi` bezeichnen also verschiedene Variablen. Case sensitive

**Übung u\_inch**

Für das englische Längenmaß Inch gilt folgende Umrechnung: 1 Inch entspricht 2,54 Zentimetern. Berechnen Sie für die folgenden Angaben den Wert in Zentimeter: 5 Inch, 20 Inch und 92,7 Inch. Vereinfachen Sie Ihre Berechnungen, indem Sie den Umrechnungsfaktor zunächst in einer Variablen speichern.

## 2.2 Erstes Programm

**Eingabe, Verarbeitung, Ausgabe**

Es soll ein erstes Python-Programm eingegeben, abgespeichert und aufgerufen werden. Dieser Vorgang wird im Folgenden ausführlich erklärt. Die einzelnen Schritte sind später analog bei jedem Python-Programm auszuführen.

**Python-Versionen**

Alle Programme beziehen sich zunächst auf Python 3 unter Windows. Falls es Unterschiede zu Python 2, zu Ubuntu Linux oder zu OS X gibt, so werden sie im jeweils gleichen Abschnitt direkt anschließend erwähnt. Jedes Programm finden Sie in zwei Versionen auf dem Datenträger zum Buch, in den Verzeichnissen *Python34* und *Python27*.

### 2.2.1 Hallo Welt

Die Ausgabe Ihres ersten Python-Programms soll lauten:

**Hallo Welt**

**Hallo Welt**

Das Programm soll also den Text »Hallo Welt« auf dem Bildschirm anzeigen. Dies ist häufig das erste Programm, das man in einer beliebigen Programmiersprache schreibt.

### 2.2.2 Eingabe eines Programms

**Neues Programmfenster**

Zur Eingabe des Programms rufen Sie in der Entwicklungsumgebung IDLE im Menü FILE zunächst den Menübefehl NEW WINDOW auf. Es öffnet sich ein neues Fenster mit dem Titel UNTITLED. Das Hauptfenster mit dem Titel PYTHON SHELL rückt in den Hintergrund. In dem neuen Fenster geben Sie das Programm wie in Abbildung 2.2 dargestellt ein.

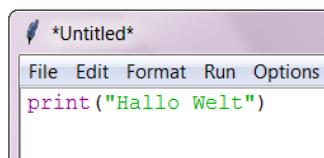


Abbildung 2.2 Eingabe des Programms in neuem Fenster

Zur Erläuterung:

- Die eingebaute Funktion `print()` gibt Text oder Werte von Variablen auf dem Bildschirm aus.

### Unterschiede in Python 2

Die Programmzeile wird ohne Klammern eingegeben, also `print "Hallo Welt"`. Auch in Python 2 dient `print` zur Ausgabe von Texten oder Variablenwerten, allerdings ist `print` eine Anweisung und keine Funktion.

## 2.3 Speichern und Ausführen

Damit Sie das Ergebnis des Programms sehen können, müssen Sie es zunächst in einer Datei speichern und anschließend ausführen.

### 2.3.1 Speichern

Zur Speicherung des Programms im Python-Verzeichnis (*Python34* bzw. *Python27*) rufen Sie im aktuellen Fenster im Menü FILE den Menübefehl SAVE auf. Das Programm wird in der Datei mit dem Namen *hallo.py* gespeichert (siehe Abbildung 2.3).

Speichern

Nach der Betätigung des Buttons SPEICHERN ist die Speicherung vollzogen. Den Namen der Datei (hier *hallo*) können Sie frei wählen. Die Dateiendung *.py* ist für Python-Programme allerdings zwingend vorgeschrieben.

Dateiendung *.py*

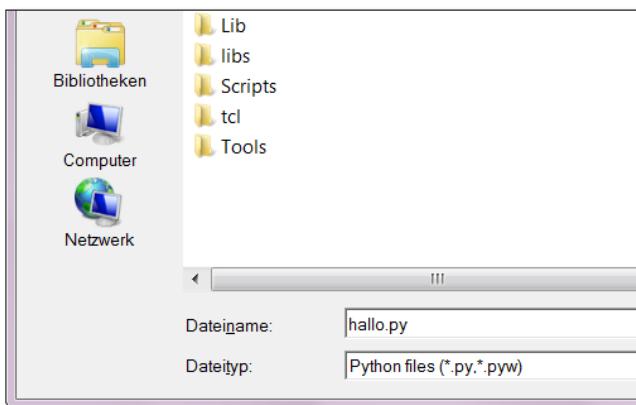


Abbildung 2.3 Speichern des Python-Programms

Das Fenster mit dem Programm sieht nun aus wie in Abbildung 2.4 dargestellt. In der Titelzeile erkennen Sie den Dateinamen bzw. den vollständigen Pfadnamen.

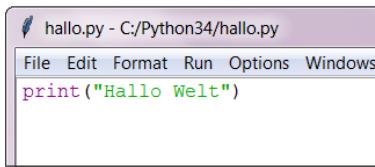


Abbildung 2.4 Dateiname und Pfad in Titelzeile

### Encoding/Decoding

Falls eine Python-Datei Umlaute oder Eszetts enthält, werden Sie (abhängig von der Version) beim Öffnen mit IDLE nach dem Encoding der Datei gefragt. Um diese Nachfrage zu vermeiden, geben Sie als oberste Zeile in der Datei ein:

```
# -*- coding: cp1252 -*-
```

Dadurch wird eine Codepage eingestellt, die die richtige Darstellung und Nutzung der Umlaute und Eszetts ermöglicht.

Bei CGI-Dateien (siehe [Abschnitt 9.2, »Webserver-Programmierung«](#)) geben Sie die oben genannte Codierung in der zweiten Zeile ein, da in der ersten Zeile der Hinweis auf den Python-Interpreter für den Webserver steht.

### 2.3.2 Ausführen unter Windows

#### Aufrufen

Sie können das Programm unter Windows auf zwei verschiedene Arten aufrufen:

- ▶ innerhalb der Entwicklungsumgebung IDLE (diese Methode nutzen wir bei den meisten Programmen dieses Buchs)
- ▶ von der Kommandozeile aus

#### Innerhalb der Entwicklungsumgebung IDLE

#### Aufruf in IDLE

Betrachten wir im Folgenden zunächst den Aufruf innerhalb der Entwicklungsumgebung IDLE. Dazu führen Sie im Menü RUN den Menübefehl RUN MODULE aus (Taste **[F5]**). Das Hauptfenster der Entwicklungsumgebung (die Python Shell) rückt in den Vordergrund, und der Ausgabetext erscheint, siehe [Abbildung 2.5](#).

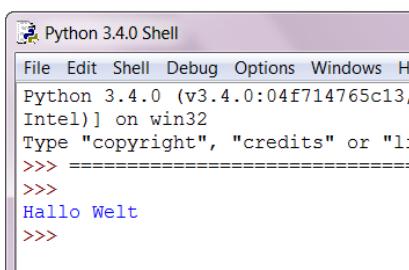


Abbildung 2.5 Aufruf in IDLE

Über die Taskleiste können Sie anschließend das Programmfenster wieder in den Vordergrund rücken.

### Von der Kommandozeile aus

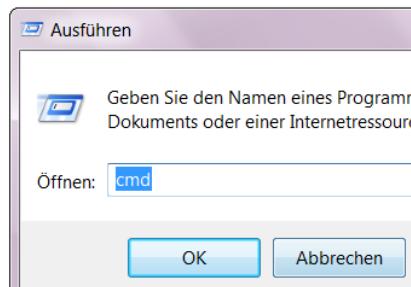
Das Programm in der Datei *hallo.py* ist ein Kommandozeilenprogramm. Es generiert also eine einfache Ausgabe auf dem Bildschirm, ohne eine eigene Benutzeroberfläche zu erzeugen. In [Kapitel 11](#), »Benutzeroberflächen«, lernen Sie, wie Sie mithilfe von Python Programme mit grafischen Benutzeroberflächen erzeugen.

Für einen Aufruf des Programms aus der Kommandozeile müssen Sie unter Windows zunächst auf die Komandozeilenebene wechseln. Dazu gehen Sie wie folgt vor:

- ▶ Rufen Sie den Menüpunkt START • (ZUBEHÖR) • AUSFÜHREN auf.
- ▶ Geben Sie im nun erscheinenden Eingabefenster die Anweisung cmd ein, und drücken Sie den Button OK (siehe [Abbildung 2.6](#)).

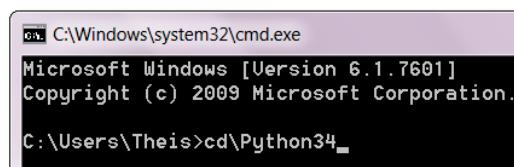
Kommandozeile

Wechsel zur Kommandozeile



**Abbildung 2.6** Aufruf des Kommandozeilenfensters

Es erscheint ein Kommandozeilenfenster wie in [Abbildung 2.7](#).



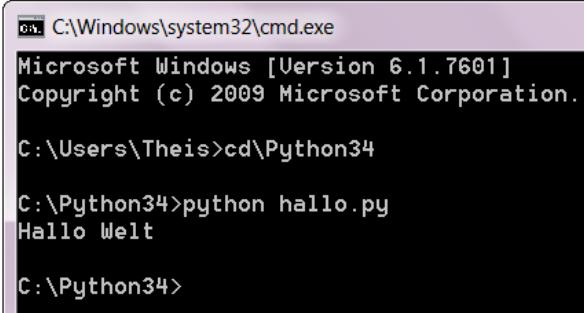
**Abbildung 2.7** Kommandozeilenfenster, nächste Eingabe

Mit Eingabe des Befehls cd\Python34 (bzw. cd\Python27) wechseln Sie in das Verzeichnis C:\Python34 (bzw. C:\Python27), in dem das Programm *hallo.py* abgelegt wurde. Anschließend veranlassen Sie die Ausführung des Programms durch Aufruf des Python-Interpreters sowohl für Python 3 als auch für Python 2 mit der Anweisung python hallo.py. Es erscheint die in [Abbildung 2.8](#) gezeigte Ausgabe.

Verzeichnis wechseln

**Versionsnummer** Durch die Eingabe von `python -V` können Sie an dieser Stelle auch die genaue Versionsnummer des benutzten Python feststellen.

**exit** Dieses Kommandozeilenfenster können Sie für spätere Aufrufe von Python-Programmen geöffnet lassen oder mit Eingabe des Befehls `exit` wieder schließen.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\Theis>cd\Python34

C:\Python34>python hallo.py
Hallo Welt

C:\Python34>
```

Abbildung 2.8 Aufruf im Kommandozeilenfenster

Unter einigen Windows-Versionen erreichen Sie dieses Fenster auch über **START • ZUBEHÖR • EINGABEAUFLÖSERUNG**.

**Python-Interpreter** Nur in dem genannten Verzeichnis (*Python34* bzw. *Python27*) steht der Python-Interpreter direkt, ohne weitere Pfadangabe, zur Verfügung.

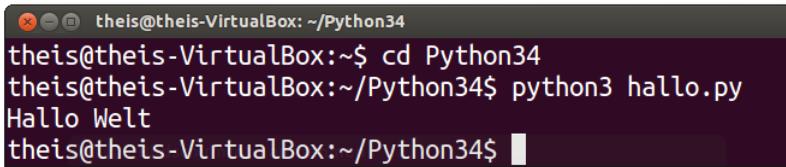
### 2.3.3 Ausführen unter Ubuntu Linux und unter OS X

**Neue erste Zeile** Sie können das Programm unter Ubuntu Linux und unter OS X ebenfalls auf zwei Arten aufrufen:

- ▶ innerhalb der Entwicklungsumgebung IDLE, siehe entsprechender Abschnitt in Abschnitt 2.3.2, »Ausführen unter Windows«
- ▶ von der Kommandozeile aus, mit dem Python-Interpreter

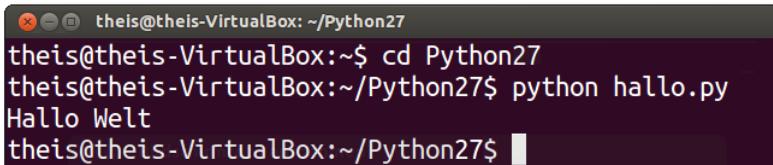
**Terminal** Öffnen Sie ein Terminal, und wechseln Sie in das Verzeichnis, in dem Sie die Datei *hallo.py* gespeichert haben. In Ubuntu Linux finden Sie das Terminal nach Eingabe des Begriffs im DASH. Unter OS X erreichen Sie das Terminal über das DOCK. Öffnen Sie darin das LAUNCHPAD, und wählen Sie die Gruppe ANDERE.

Veranlassen Sie die Ausführung des Programms für Python 3 mit der Anweisung `python3 hallo.py` und für Python 2 mit der Anweisung `python hallo.py`, siehe Abbildung 2.9 bis Abbildung 2.12.



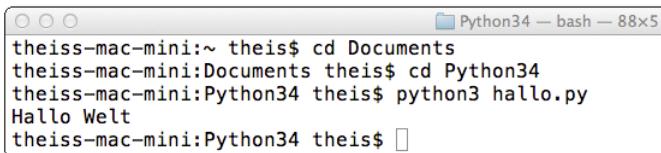
```
theis@theis-VirtualBox:~/Python34$ cd Python34
theis@theis-VirtualBox:~/Python34$ python3 hallo.py
Hallo Welt
theis@theis-VirtualBox:~/Python34$
```

Abbildung 2.9 Aufruf von Python 3 in Ubuntu Linux



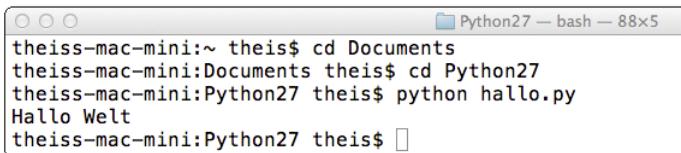
```
theis@theis-VirtualBox:~/Python27$ cd Python27
theis@theis-VirtualBox:~/Python27$ python hallo.py
Hallo Welt
theis@theis-VirtualBox:~/Python27$
```

Abbildung 2.10 Aufruf von Python 2 in Ubuntu Linux



```
theiss-mac-mini:~ theis$ cd Documents
theiss-mac-mini:Documents theis$ cd Python34
theiss-mac-mini:Python34 theis$ python3 hallo.py
Hallo Welt
theiss-mac-mini:Python34 theis$
```

Abbildung 2.11 Aufruf von Python 3 in OS X



```
theiss-mac-mini:~ theis$ cd Documents
theiss-mac-mini:Documents theis$ cd Python27
theiss-mac-mini:Python27 theis$ python hallo.py
Hallo Welt
theiss-mac-mini:Python27 theis$
```

Abbildung 2.12 Aufruf von Python 2 in OS X

Durch die Eingabe von `python -V` bzw. `python3 -V` können Sie an dieser Stelle auch die genaue Versionsnummer des benutzten Python feststellen.

Versionsnummer

### Hinweis

Sie finden weitere Kommandozeilenbefehle für Ubuntu Linux und OS X im Anhang A.2.

Linux-Befehle

### 2.3.4 Kommentare

Bei umfangreicheren Programmen erweist es sich als sehr nützlich, Kommentare zur Erläuterung in den Programmtext einzufügen. Einzelige Kommentare werden durch das Raute-Zeichen `#` eingeleitet und reichen bis zum

# und "'''

Zeilenende. Mehrzeilige Kommentare beginnen und enden jeweils mit drei doppelten Anführungszeichen. Kommentare werden vom System nicht als Programmschritte betrachtet und folglich nicht ausgeführt. Im folgenden Listing wurde das erste Programm durch Kommentare ergänzt:

```
# Mein erstes Programm
print("Hallo Welt")      # Eine Ausgabe
"""Kommentar in
mehreren Zeilen"""


```

**Listing 2.2** Datei hallo.py

### 2.3.5 Verkettung von Ausgaben

Mithilfe der Funktion `print()` können auch mehrere Ausgaben in einer Zeile ausgegeben werden. Entsprechend wurde das Programm weiter verändert:

```
# Mein erstes Programm
print("Hallo", "Welt")
```

**Listing 2.3** Datei hallo.py

Zur Erläuterung:

- Kommata**
- ▶ Die einzelnen Teile der Ausgabe werden durch Kommata voneinander getrennt.
  - ▶ Nach jedem Teil der Ausgabe wird automatisch ein Leerzeichen eingesetzt. Dieses Verhalten können Sie als Programmierer beeinflussen; mehr dazu in [Abschnitt 5.2.1](#).

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 2.3.6 Lange Ausgaben

Zeichenketten, die mithilfe der Funktion `print()` ausgegeben werden, können sehr lang werden, eventuell sogar über den rechten Rand innerhalb des Editors hinausgehen. Damit wird der Programmcode allerdings etwas unübersichtlich. Sie können solche Zeichenketten aber auch auf mehrere Zeilen verteilen – dieses Vorgehen wird auch im vorliegenden Buch aufgrund der begrenzten Druckbreite häufig angewendet.

- Übersichtlicher** Wenngleich es für dieses kurze Programm eigentlich nicht notwendig ist, verteilen wir hier zur Verdeutlichung den Code auf mehrere Zeilen:

```
# Mein erstes Programm
print("Hallo",
      "Welt")
```

```
print("Hallo "
      "Welt")
```

**Listing 2.4** Datei hallo.py

Zur Erläuterung:

- ▶ Es bietet sich an, den Zeilenumbruch nach einem Komma durchzuführen, wie bei der ersten Ausgabe von »Hallo Welt«. Dabei wird, wie bereits ausgeführt, automatisch ein Leerzeichen gesetzt.
- ▶ Einzelne lange Texte können in Teiltexte zerlegt werden, wie bei der zweiten Ausgabe von »Hallo Welt«. Teiltexte müssen nicht durch Kommata voneinander getrennt werden. Sie sind allerdings jeweils mit Anführungsstrichen zu begrenzen. Das trennende Leerzeichen zwischen »Hallo« und »Welt« müssen Sie nun von Hand setzen.

Teiltexte

Zeichen \

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Außerdem müssen Sie das Zeichen `\` einsetzen. Damit wird gekennzeichnet, dass die Programmzeile über mehr als eine Bildschirmzeile geht. Der Code sieht dann wie folgt aus:

```
print "Hallo", \
      "Welt"
print "Hallo" \
      "Welt"
```

### Hinweis

In Abschnitt 5.1.2, »Programmzeile in mehreren Zeilen«, erläutere ich, wie Sie allgemein lange Programmzeilen aufteilen können.



# Kapitel 3

## Programmierkurs

Der folgende Programmierkurs mit ausführlichen Erläuterungen führt Sie schrittweise in die Programmierung mit Python ein. Begleitet wird der Kurs von einem Programmierprojekt, das die vielen Teilespekte zu einem Ganzen verknüpft.

### 3.1 Ein Spiel programmieren

Damit Sie die Programmiersprache Python auf unterhaltsame Weise kennenlernen, sollen Sie im Folgenden selbst ein Spiel programmieren. Im weiteren Verlauf des Buchs wird dieses Spiel dann kontinuierlich erweitert und verbessert.

#### 3.1.1 Das fertige Spiel

Das Spiel läuft wie folgt ab: Dem Anwender wird nach Aufruf des Programms eine Kopfrechenaufgabe gestellt. Er gibt das von ihm ermittelte Ergebnis ein, und das Programm bewertet seine Eingabe.

Kopfrechnen

Die Aufgabe:  $9 + 26$

Bitte eine Zahl eingeben:

34

34 ist falsch

Bitte eine Zahl eingeben:

35

35 ist richtig

Ergebnis: 35

Anzahl Versuche: 2

#### 3.1.2 Der Weg zum fertigen Spiel

Das Spiel wird in mehreren Einzelschritten erstellt. Zunächst entsteht eine ganz einfache Version. Mit wachsender Programmierkenntnis entwickeln Sie dann immer komplexere Versionen. Die im jeweiligen Abschnitt erlernten Programmierfähigkeiten setzen Sie unmittelbar zur Verbesserung des Spielablaufs ein.

Das Spiel wächst

#### 3.1.3 Mögliche Erweiterungen

- Versionen** In späteren Abschnitten entstehen weitere Versionen des Spiels. Das Spiel begleitet Sie so durch das gesamte Buch. Es wird unter anderem um folgende Möglichkeiten erweitert:
- ▶ mehrere Aufgaben nacheinander oder gleichzeitig stellen
  - ▶ Messung der benötigten Zeit
  - ▶ Speicherung der Spieldaten in einer Datei
  - ▶ Aktualisierung und Darstellung einer Highscore-Liste
  - ▶ Spielen im Internet

### 3.2 Variablen und Operatoren

Zur Speicherung von Werten werden Variablen benötigt. Operatoren dienen zur Ausführung von Berechnungen.

#### 3.2.1 Berechnung und Zuweisung

Im folgenden Programm wird eine einfache Berechnung mithilfe eines Operators durchgeführt. Das Ergebnis der Berechnung wird mit dem Gleichheitszeichen einer Variablen zugewiesen. Es erfolgt eine Ausgabe. Diese Schritte kennen Sie bereits aus Abschnitt 2.1.5, »Variablen und Zuweisung«.

```
# Werte  
a = 5  
b = 3  
  
# Berechnung  
c = a + b  
  
# Ausgabe  
print("Die Aufgabe:", a, "+", b)  
print("Das Ergebnis:", c)
```

**Listing 3.1 Datei zuweisung.py**

Die Ausgabe des Programms lautet:

**Die Aufgabe: 5 + 3**  
**Das Ergebnis: 8**

Zur Erläuterung:

- Eingangsdaten** ▶ In den beiden Variablen `a` und `b` wird jeweils ein Wert gespeichert.
- Verarbeitung** ▶ Die Werte der beiden Variablen werden addiert, das Ergebnis wird der Variablen `c` zugewiesen.

- Die Aufgabenstellung wird ausgegeben. Anschließend wird das Ergebnis **Ausgabe** ausgegeben.
- Noch hat der Benutzer des Programms keine Möglichkeit, in den Ablauf einzugreifen.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 3.2.2 Eingabe einer Zeichenkette

In diesem Abschnitt wird die Funktion `input()` zur Eingabe einer Zeichenkette **input()** durch den Benutzer eingeführt. Ein kleines Beispiel:

```
# Eingabe einer Zeichenkette
print("Bitte einen Text eingeben")
x = input()
print("Ihre Eingabe:", x)
```

**Listing 3.2 Datei eingabe\_text.py**

Die Ausgabe könnte wie folgt aussehen:

```
Bitte einen Text eingeben
Python ist toll
Ihre Eingabe: Python ist toll
```

Zur Erläuterung:

- Der Benutzer gibt einen kleinen Satz ein. Dieser Satz wird in der Variablen `x` gespeichert und anschließend ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Somit lautet das Programm:

```
print "Bitte einen Text eingeben"
x = raw_input()
print "Ihre Eingabe:", x
```

### 3.2.3 Eingabe einer Zahl

Im weiteren Verlauf des Spiels ist es notwendig, die Eingabe des Benutzers als Zahl weiterzuverwenden. Dazu muss die Zeichenkette, die die Funktion `input()` liefert, in eine Zahl umgewandelt werden.

**Umwandlung**

Zur Umwandlung gibt es unter anderem die folgenden Funktionen:

- int()** ► Die eingebaute Funktion `int()` wandelt eine Zeichenkette in eine ganze Zahl um; eventuell vorhandene Nachkommastellen werden abgeschnitten.
- float()** ► Die eingebaute Funktion `float()` wandelt eine Zeichenkette in eine Zahl mit Nachkommastellen um.
- Abbruch** ► Sollte die Zeichenkette für `float()` keine gültige Zahl enthalten, so kommt es zu einem Programmabbruch. Bei `int()` muss es sich sogar um eine gültige ganze Zahl handeln. In Abschnitt 3.6, »Fehler und Ausnahmen«, lernen Sie, wie Sie Programmabbrüche abfangen.

Ein Beispiel:

```
# Eingabe einer Zahl
print("Bitte eine ganze Zahl eingeben")
x = input()
print("Ihre Eingabe:", x)

# Umwandlung in ganze Zahl
xganz = int(x)
print("Als ganze Zahl:", xganz)

# Mit Berechnung
xdoppel = xganz * 2
print("Das Doppelte:", xdoppel)
```

**Listing 3.3 Datei `eingabe_zahl.py`**

Die Ausgabe könnte wie folgt aussehen:

```
Bitte eine ganze Zahl eingeben
6
Ihre Eingabe: 6
Als ganze Zahl: 6
Das Doppelte: 12
```

Zur Erläuterung:

- Der Benutzer gibt eine Zeichenkette ein.
  - Diese Zeichenkette wird mithilfe der eingebauten Funktion `int()` in eine ganze Zahl umgewandelt.
  - Die Zahl und das Doppelte der Zahl werden ausgegeben.
- EVA** In diesem Programm wird auch das EVA-Prinzip der klassischen Programmierung verdeutlicht:
- E = Eingabe (der Anwender gibt Daten über die Tastatur ein)
  - V = Verarbeitung (das Programm verarbeitet diese und andere Daten)
  - A = Ausgabe (das Programm gibt das Ergebnis der Verarbeitung auf dem Bildschirm aus)

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

### 3.2.4 Spiel, Version mit Eingabe

Das Spielprogramm wird mit einer Eingabe versehen. Es wird wie folgt geändert:

```
# Werte und Berechnung
a = 5
b = 3
c = a + b

print("Die Aufgabe:", a, "+", b)

# Eingabe
print("Bitte eine Zahl eingeben:")
z = input()

# Eingabe in Zahl umwandeln
zahl = int(z)

# Ausgabe
print("Ihre Eingabe:", z)
print("Das Ergebnis:", c)
```

**Listing 3.4** Datei `spiel_eingabe.py`

Eine mögliche Ausgabe des Programms:

```
Die Aufgabe: 5 + 3
Bitte eine Zahl eingeben:
9
Ihre Eingabe: 9
Das Ergebnis: 8
```

Zur Erläuterung:

- ▶ Das Programm gibt die Aufforderung `Bitte eine Zahl eingeben:` aus und hält an.
- ▶ Die Eingabe des Benutzers wird in der Variablen `z` gespeichert.
- ▶ Die Zeichenkette `z` wird mithilfe der Funktion `int()` in eine ganze Zahl verwandelt.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

Inch in Zentimeter

#### Übung `u_eingabe_inch`

Schreiben Sie ein Programm zur Eingabe und Umrechnung von beliebigen Inch-Werten in Zentimeter. Speichern Sie das Programm in der Datei `u_eingabe_inch.py`. Rufen Sie das Programm auf, und testen Sie es. Die Ausgabe sollte z. B. wie folgt aussehen:

Bitte geben Sie den Inch-Wert ein:

3.5

3.5 inch sind 8.89 cm

Steuer  
berechnen

#### Übung `u_eingabe_gehalt`

Schreiben Sie ein Programm zur Berechnung des monatlich zu zahlenden Steuerbetrags. Der Anwender soll zunächst dazu aufgefordert werden, sein monatliches Bruttogehalt einzugeben. Anschließend werden (hier sehr vereinfacht) 18 % dieses Betrags berechnet und ausgegeben. Speichern Sie das Programm in der Datei `u_eingabe_gehalt.py`. Die Ausgabe sollte z. B. wie folgt aussehen:

Geben Sie Ihr Bruttogehalt in Euro ein:

2500

Es ergibt sich ein Steuerbetrag von 450.0 Euro

### 3.2.5 Zufallszahlen

Natürlich soll nicht immer die gleiche Kopfrechenaufgabe gestellt werden. In Python steht ein Zufallszahlengenerator zur Verfügung. Dieser generiert zufällige Zahlen, die wir zur Bildung der Aufgabenstellung nutzen wollen.

Modul  
importieren

Die Funktionen des Zufallszahlengenerators befinden sich in einem zusätzlichen Modul, das zunächst importiert werden muss. Das Programm wird wie folgt verändert:

```
# Modul random importieren
import random

# Zufallsgenerator initialisieren
random.seed()
```

```
# Zufallswerte und Berechnung
a = random.randint(1,10)
b = random.randint(1,10)
c = a + b

print("Die Aufgabe:", a, "+", b)
# Eingabe
print("Bitte eine Zahl eingeben:")
z = input()
zahl = int(z)

# Ausgabe
print("Ihre Eingabe:", z)
print("Das Ergebnis:", c)
```

**Listing 3.5** Datei spiel\_zufallszahl.py

Eine mögliche Ausgabe des Programms, abhängig von den gelieferten zufälligen Werten, sieht wie folgt aus:

**Die Aufgabe:** 8 + 3

**Bitte eine Zahl eingeben:**

7

**Ihre Eingabe:** 7

**Das Ergebnis:** 11

Zur Erläuterung:

- ▶ Zusätzliche Module können Sie mithilfe der Anweisung `import` in das Programm einbinden.
- ▶ Die Funktionen dieser Module können Sie anschließend in der Schreibweise `Modulname.Funktionsname` aufrufen.
- ▶ Der Aufruf der Funktion `seed()` des Moduls `random` führt dazu, dass der Zufallszahlengenerator mit der aktuellen Systemzeit initialisiert wird. Andernfalls könnte es passieren, dass anstelle einer zufälligen Auswahl immer wieder dieselben Zahlen geliefert würden.
- ▶ Die Funktion `randint()` des Moduls `random` liefert eine ganze Zufallszahl im angegebenen Bereich. Im vorliegenden Fall ist dies also eine zufällige Zahl von 1 bis 10.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

### 3.3 Verzweigungen

#### Programmablauf steuern

In den bisherigen Programmen wurden alle Anweisungen der Reihe nach ausgeführt. Zur Steuerung eines Programmablaufs werden allerdings häufig Verzweigungen benötigt. Innerhalb des Programms wird dann anhand einer Bedingung entschieden, welcher Zweig des Programms ausgeführt werden soll.

#### 3.3.1 Vergleichsoperatoren

##### Kleiner, größer, gleich

Bedingungen werden mithilfe von Vergleichsoperatoren formuliert. [Tabelle 3.1](#) listet die Vergleichsoperatoren mit ihrer Bedeutung auf.

Operator	Bedeutung
>	größer als
<	kleiner als
>=	größer als oder gleich
<=	kleiner als oder gleich
==	gleich
!=	ungleich (in Python 2 gab es dafür zusätzlich den Operator <>)

**Tabelle 3.1** Vergleichsoperatoren

#### 3.3.2 Einfache Verzweigung

##### if-else

Im folgenden Beispiel wird untersucht, ob eine Zahl positiv ist. Ist dies der Fall, so wird Diese Zahl ist positiv ausgegeben, anderenfalls lautet die Ausgabe Diese Zahl ist 0 oder negativ. Es wird also nur eine der beiden Anweisungen ausgeführt.

```
x = 12
print("x:", x)

if x > 0:
    print("Diese Zahl ist positiv")
else:
    print("Diese Zahl ist 0 oder negativ")
```

**Listing 3.6** Datei `verzweigung_einfach.py`

Die Ausgabe des Programms lautet:

```
x: 12
Diese Zahl ist positiv
```

Zur Erläuterung:

- ▶ In der ersten Zeile bekommt die Variable `x` den Wert `12` zugewiesen.
- ▶ In der zweiten Zeile wird mithilfe der Anweisung `if` eine Verzweigung eingeleitet. Danach wird eine Bedingung formuliert (hier  $x > 0$ ), die entweder *wahr* oder *falsch* ergibt. Anschließend folgt der Doppelpunkt (`:`). Doppelpunkt
- ▶ Es folgen eine oder mehrere Anweisungen, die nur ausgeführt werden, wenn die Bedingung *wahr* ergibt. Die Anweisungen müssen innerhalb des sogenannten `if`-Zweigs mithilfe der -Taste eingerückt werden, damit Python die Zugehörigkeit zur Verzweigung erkennen kann. Gleichzeitig macht die Einrückung das Programm übersichtlicher. Einrücken
- ▶ In der vierten Zeile wird mithilfe der Anweisung `else` der alternative Teil der Verzweigung eingeleitet. Es folgt wiederum ein Doppelpunkt. else
- ▶ Es folgen eine oder mehrere Anweisungen, die nur ausgeführt werden, falls die Bedingung *falsch* ergibt. Auch diese Anweisungen müssen eingerückt werden.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 3.3.3 Spiel, Version mit Bewertung der Eingabe

Nun soll die Eingabe des Benutzers bewertet werden. Mithilfe einer einfachen Verzweigung wird untersucht, ob die Eingabe richtig oder falsch war. Das Programm wird wie folgt verändert:

**Eingabe prüfen**

```
# Zufallsgenerator
import random
random.seed()

# Werte und Berechnung
a = random.randint(1,10)
b = random.randint(1,10)
c = a + b

print("Die Aufgabe:", a, "+", b)

# Eingabe
print("Bitte eine Zahl eingeben:")
z = input()
zahl = int(z)

# Verzweigung
if zahl == c:
```

```

        print(zahl, "ist richtig")
else:
    print(zahl, "ist falsch")
    print("Ergebnis: ", c)

```

**Listing 3.7 Datei spiel\_verzweigung.py**

Eine mögliche Ausgabe des Programms wäre:

```

Die Aufgabe: 2 + 8
Bitte eine Zahl eingeben:
11
11 ist falsch
Ergebnis: 10

```

Zur Erläuterung:

- ▶ Die Eingabe wird in eine Zahl umgewandelt.
- Richtig**
- ▶ Entspricht diese Zahl dem Ergebnis der Rechnung, so wird `ist richtig` ausgegeben.
- Falsch**
- ▶ Entspricht die Zahl nicht dem Ergebnis, so wird `ist falsch` ausgegeben, und das korrekte Ergebnis wird genannt.

**Unterschiede in Python 2**

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

**Übung u\_verzweigung\_einfach**

Das Programm zur Berechnung des monatlich zu zahlenden Steuerbetrags soll verändert werden. Der Anwender soll zunächst dazu aufgefordert werden, sein monatliches Bruttogehalt einzugeben. Liegt das Gehalt über 2.500 €, so sind 22 % Steuern zu zahlen, ansonsten 18 % Steuern. Speichern Sie das Programm in der Datei `u_verzweigung_einfach.py`.

Zur Erläuterung: Es ist nur *eine* Eingabe erforderlich. Innerhalb des Programms wird anhand des Gehalts entschieden, welcher Steuersatz zur Anwendung kommt.

Die Ausgabe sollte z. B. wie folgt aussehen:

Geben Sie Ihr Bruttogehalt in Euro ein:

3000

Es ergibt sich ein Steuerbetrag von 660.0 Euro

oder

Geben Sie Ihr Bruttogehalt in Euro ein:

2000

Es ergibt sich ein Steuerbetrag von 360.0 Euro

### 3.3.4 Mehrfache Verzweigung

In vielen Anwendungsfällen gibt es mehr als zwei Möglichkeiten, zwischen denen zu entscheiden ist. Dazu wird eine mehrfache Verzweigung benötigt. Im folgenden Beispiel wird untersucht, ob eine Zahl positiv, negativ oder gleich 0 ist. Es wird eine entsprechende Meldung ausgegeben:

```
x = -5
print("x:", x)
if x > 0:
    print("x ist positiv")
elif x < 0:
    print("x ist negativ")
else:
    print("x ist gleich 0")
```

Mehrere  
Möglichkeiten

if-elif-else

**Listing 3.8** Datei verzweigung\_mehrfach.py

Die Ausgabe des Programms:

```
x: -5
x ist negativ
```

Zur Erläuterung:

- ▶ Mit der Anweisung `if` wird die Verzweigung eingeleitet. Falls `x` positiv ist, werden die folgenden, eingerückten Anweisungen ausgeführt.
- ▶ Nach der Anweisung `elif` wird eine weitere Bedingung formuliert. Diese wird nur untersucht, wenn die erste Bedingung (nach dem `if`) nicht zutraf. Falls `x` negativ ist, werden die folgenden, eingerückten Anweisungen ausgeführt.
- ▶ Die Anweisungen nach der Zeile mit der `else`-Anweisung werden nur durchgeführt, falls keine der beiden vorherigen Bedingungen zutraf (nach dem `if` und nach dem `elif`). In diesem Fall bedeutet das, dass `x` gleich 0 ist, da es nicht positiv und nicht negativ ist.

if

elif

else

Innerhalb einer Verzweigung können mehrere `elif`-Anweisungen vorkommen. Diese werden der Reihe nach untersucht, bis das Programm zu der Bedingung kommt, die zutrifft. Die weiteren `elif`-Anweisungen oder eine `else`-Anweisung werden dann nicht mehr beachtet.

Mehrere elif-  
Anweisungen

**else muss nicht vorkommen** Eine else-Anweisung ist weder bei einer einfachen Verzweigung noch bei einer mehrfachen Verzweigung zwingend notwendig.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Übung u\_verzweigung\_mehrgefach

Das Programm zur Berechnung des monatlich zu zahlenden Steuerbetrags soll weiter verändert werden (Datei *u\_verzweigung\_mehrgefach.py*). Der Anwender soll zunächst dazu aufgefordert werden, sein monatliches Bruttogehalt einzugeben. Anschließend soll das Gehalt nach der folgenden Steuertabelle berechnet werden:

Gehalt	Steuersatz
mehr als 4.000 €	26 %
2.500 bis 4.000 €	22 %
weniger als 2.500 €	18 %

### 3.3.5 Logische Operatoren

**Logische Verknüpfung** Mithilfe der logischen Operatoren `and`, `or` und `not` können mehrere Bedingungen miteinander verknüpft werden.

- and** ▶ Eine Bedingung, die aus einer oder mehreren Einzelbedingungen besteht, die jeweils mit dem Operator `and` (= und) verknüpft sind, ergibt *wahr*, wenn *jede* der Einzelbedingungen *wahr* ergibt.
- or** ▶ Eine Bedingung, die aus einer oder mehreren Einzelbedingungen besteht, die mit dem Operator `or` (= oder) verknüpft sind, ergibt *wahr*, wenn *mindestens eine* der Einzelbedingungen *wahr* ergibt.
- not** ▶ Der Operator `not` (= nicht) kehrt den Wahrheitswert einer Bedingung um, das heißt, eine falsche Bedingung wird *wahr*, eine wahre Bedingung wird *falsch*.

**Wahrheitswerte** Der gleiche Zusammenhang wird in der Wahrheitswerttabelle (siehe [Tabelle 3.2](#)) dargestellt.

Bedingung 1	Operator	Bedingung 2	gesamte Bedingung
wahr	and	wahr	wahr
wahr	and	falsch	falsch
falsch	and	wahr	falsch
falsch	and	falsch	falsch
wahr	or	wahr	wahr
wahr	or	falsch	wahr
falsch	or	wahr	wahr
falsch	or	falsch	falsch
-	not	wahr	falsch
-	not	falsch	wahr

Tabelle 3.2 Einsatz von logischen Operatoren

Ein Beispiel:

```
x = 12
y = 15
z = 20

print("x:", x)
print("y:", y)
print("z:", z)

# Bedingung 1
if x<y and x<z:
    print("x ist die kleinste Zahl")

# Bedingung 2
if y>x or y>z:
    print("y ist nicht die kleinste Zahl")
# Bedingung 3
if not y<x:
    print("y ist nicht kleiner als x")
```

Listing 3.9 Datei operator\_logisch.py

Die Ausgabe des Programms:

```
x:12
y:15
```

```

z: 20
x ist die kleinste Zahl
y ist nicht die kleinste Zahl
y ist nicht kleiner als x

```

Zur Erläuterung:

- and** ► Bedingung 1 ergibt *wahr*, wenn *x* kleiner als *y* *und* kleiner als *z* ist. Dies trifft bei den gegebenen Anfangswerten zu.
- or** ► Bedingung 2 ergibt *wahr*, wenn *y* größer als *x* *oder* *y* größer als *z* ist. Die erste Bedingung trifft zu, also ist die gesamte Bedingung wahr: *y* ist nicht die kleinste der drei Zahlen.
- not** ► Bedingung 3 ist wahr, wenn *y* *nicht* kleiner als *x* ist. Dies trifft hier zu.
- Zu allen Verzweigungen kann es natürlich auch einen *else*-Zweig geben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

Mehrere  
Entscheidungen

### Übung u\_operator

Das Programm zur Berechnung des monatlich zu zahlenden Steuerbetrags soll wiederum verändert werden (Datei *u\_operator.py*). Die Steuertabelle sieht nun wie folgt aus:

Gehalt	Familienstand	Steuersatz
> 4.000 €	ledig	26 %
> 4.000 €	verheiratet	22 %
<= 4.000 €	ledig	22 %
<= 4.000 €	verheiratet	18 %

### 3.3.6 Mehrere Vergleichsoperatoren

- 1 < 2 < 3** Bedingungen können auch mehrere Vergleichsoperatoren enthalten. Manche Verzweigungen sind mithilfe von mehreren Vergleichsoperatoren anstelle von mehreren verknüpften Bedingungen verständlicher. Es folgt ein Beispiel:

```

x = 12
y = 15
z = 20

```

```

print("x:", x)
print("y:", y)
print("z:", z)

# Bedingung 1
if x < y < z:
    print("y liegt zwischen x und z")

```

**Listing 3.10** Datei operator\_vergleich.py

Die Ausgabe des Programms:

```

x: 12
y: 15
z: 20
y liegt zwischen x und z

```

Zur Erläuterung:

- ▶ Die Bedingung  $x < y < z$  entspricht dem Ausdruck  $x < y$  and  $y < z$ . In der Kurzform ist sie allerdings besser lesbar.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 3.3.7 Spiel, Version mit genauer Bewertung der Eingabe

Nun kann die Eingabe des Benutzers genauer bewertet werden:

- ▶ mithilfe einer mehrfachen Verzweigung
- ▶ anhand logischer Operatoren
- ▶ anhand von Bedingungen mit mehreren Vergleichsoperatoren

Das Programm wird wie folgt verändert:

```

# Zufallsgenerator
import random
random.seed()

# Werte und Berechnung
a = random.randint(1,10)
b = random.randint(1,10)
c = a + b

print("Die Aufgabe:", a, "+", b)

# Eingabe
print("Bitte eine Zahl eingeben:")
z = input()

```

```
zahl = int(z)

# Mehrfache Verzweigung, logische Operatoren
# Bedingungen mit mehreren Vergleichsoperatoren
if zahl == c:
    print(zahl, "ist richtig")
elif zahl < 0 or zahl > 100:
    print(zahl, "ist ganz falsch")
elif c-1 <= zahl <= c+1:
    print(zahl, "ist ganz nahe dran")
else:
    print(zahl, "ist falsch")

# Ende
print("Ergebnis: ", c)
```

**Listing 3.11** Datei spiel\_operator.py

Eine mögliche Ausgabe des Programms wäre:

```
Die Aufgabe: 2 + 1
Bitte eine Zahl eingeben:
4
4 ist ganz nahe dran
Ergebnis: 3
```

Zur Erläuterung:

- ▶ Insgesamt werden jetzt über if, zweimal elif und else vier Möglichkeiten geboten.
- or
  - ▶ Ist die Eingabe kleiner als 0 oder größer als 100, so ist sie ganz falsch. Dies wird mithilfe des logischen Operators or gelöst.
  - ▶ Unterscheidet sich die Zahl vom richtigen Ergebnis nur um den Wert 1, so ist sie ganz nahe dran. Dies wird mit einer Bedingung gelöst, die mehrere Vergleichsoperatoren enthält.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

### 3.3.8 Rangfolge der Operatoren

In vielen Ausdrücken treten mehrere Operatoren auf. Bisher sind dies Rechenoperatoren, Vergleichsoperatoren und logische Operatoren. Damit Python weiß, in welcher Reihenfolge die einzelnen Teilschritte bearbeitet werden sol-

len, gilt eine feste Rangfolge der Operatoren. Die Teilschritte, bei denen höherrangige Operatoren beteiligt sind, werden zuerst ausgeführt.

**Tabelle 3.3** gibt die Rangfolge der bisher verwendeten Operatoren in Python an, beginnend mit den Operatoren, die den höchsten Rang haben. Gleichrangige Operatoren stehen jeweils in einer Zeile. Teilschritte, in denen mehrere Operatoren gleichen Ranges stehen, werden von links nach rechts ausgeführt.

Höchster Rang oben

Operator	Bedeutung
+ -	positives Vorzeichen einer Zahl, negatives Vorzeichen einer Zahl
* / % //	Multiplikation, Division, Modulo, Ganzzahldivision
+ -	Addition, Subtraktion
< <= > >= !=	kleiner, kleiner oder gleich, größer, größer oder gleich, gleich, ungleich
not	logische Verneinung
and	logisches Und
or	logisches Oder

**Tabelle 3.3** Rangfolge der bisher genutzten Operatoren

## 3.4 Schleifen

Neben der Verzweigung gibt es eine weitere wichtige Struktur zur Steuerung von Programmen: die Schleife. Mithilfe einer Schleife ermöglichen Sie die wiederholte Ausführung eines Programmschritts.

Es muss zwischen zwei Typen von Schleifen unterschieden werden, der `for`-Schleife und der `while`-Schleife. Der jeweilige Anwendungsbereich der beiden Typen wird durch folgende Merkmale definiert:

Wiederholung

- ▶ Eine `for`-Schleife wird verwendet, wenn ein Programmschritt für eine regelmäßige, zum Zeitpunkt der Anwendung bekannte Folge von Werten wiederholt ausgeführt werden soll.
- ▶ Eine `while`-Schleife wird verwendet, wenn sich erst durch Eingaben des Anwenders ergibt, ob ein Programmschritt ausgeführt werden soll und wie oft er wiederholt wird.

for

while

Eine `for`-Schleife wird auch als *Zählschleife* bezeichnet, eine `while`-Schleife als *bedingungsgesteuerte Schleife*.

#### 3.4.1 for-Schleife

Die Anwendung der `for`-Schleife verdeutlicht das folgende Beispiel:

```
for i in 2, 7.5, -22:  
    print("Zahl:", i, ", Quadrat:", i*i)
```

**Listing 3.12** Datei `schleife_for.py`

Folgende Ausgabe wird erzeugt:

```
Zahl: 2 , Quadrat: 4  
Zahl: 7.5 , Quadrat: 56.25  
Zahl: -22 , Quadrat: 484
```

Zur Erläuterung:

##### Abfolge von Zahlen

- ▶ Die erste Zeile ist wie folgt zu lesen: Führe die folgenden eingerückten Anweisungen für jede Zahl in der Abfolge 2, 7.5 und -22 aus. Nenne diese Zahl in diesen Anweisungen `i`.
- ▶ Nach der Zahlenfolge muss, ebenso wie bei `if-else`, ein Doppelpunkt notiert werden.
- ▶ Anstelle von `i` können Sie natürlich auch eine andere Variable als Schleifenvariable nutzen.
- ▶ Die Zahlen werden zusammen mit einem kurzen Informationstext ausgegeben bzw. quadriert und ausgegeben.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### Hinweise

##### Iterable

1. Eine solche Abfolge von Objekten, die z. B. in einer `for`-Schleife durchlaufen werden kann, nennt man auch *iterierbares Objekt* oder kurz *Iterable*. Diese Iterables werden uns in Python noch häufig begegnen.
2. Den Vorgang des Durchlaufens nennt man auch *Iterieren*.

#### 3.4.2 Schleifenabbruch mit »break«

##### break

Die Anweisung `break` bietet eine weitere Möglichkeit zur Steuerung von Schleifen. Sie führt zu einem unmittelbaren Abbruch einer Schleife. Sie wird sinnvollerweise mit einer Bedingung verbunden und häufig bei Sonderfällen eingesetzt. Ein Beispiel:

```
for i in 12, -4, 20, 7:
    if i*i > 200:
        break
    print("Zahl:", i, ", Quadrat:", i*i)
print("Ende")
```

**Listing 3.13** Datei schleife\_break.py

Es wird die Ausgabe erzeugt:

Zahl: 12 , Quadrat: 144

Zahl: -4 , Quadrat: 16

Ende

Zur Erläuterung:

- ▶ Die Schleife wird unmittelbar verlassen, falls das Quadrat der aktuellen Zahl größer als 200 ist.
- ▶ Die Ausgabe innerhalb der Schleife erfolgt auch nicht mehr.
- ▶ Das Programm wird nach der Schleife fortgesetzt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 3.4.3 Geschachtelte Kontrollstrukturen

Wie das vorherige Programm verdeutlicht, können Kontrollstrukturen (also Verzweigungen und Schleifen) auch geschachtelt werden. Dies bedeutet, dass eine Kontrollstruktur eine weitere Kontrollstruktur enthält. Diese kann ihrerseits wiederum eine Kontrollstruktur enthalten usw. Ein weiteres Beispiel:

```
for x in -2, -1, 0, 1, 2:
    if x > 0:
        print(x, "positiv")
    else:
        if x < 0:
            print(x, "negativ")
        else:
            print(x, "gleich 0")
```

**Listing 3.14** Datei schachtelung.py

Es wird die Ausgabe erzeugt:

-2 negativ

-1 negativ

0 gleich 0

1 positiv  
2 positiv

Zur Erläuterung:

**Äußeres for** ► Die äußerste Kontrollstruktur ist eine `for`-Schleife. Alle einfach eingerückten Anweisungen werden – gemäß der Schleifensteuerung – mehrmals ausgeführt.

**Äußeres if** ► Mit der äußeren `if`-Anweisung wird die erste Verzweigung eingeleitet. Ist `x` größer 0, wird die folgende, zweifach eingerückte Anweisung ausgeführt.

**Innteres if** ► Trifft die Bedingung zur äußeren `if`-Anweisung nicht zu, so wird die innere `if`-Anweisung hinter der äußeren `else`-Anweisung untersucht.

► Trifft diese Bedingung zu, so werden die folgenden, dreifach eingerückten Anweisungen ausgeführt.

► Trifft die innere `if`-Anweisung nicht zu, so werden die dreifach eingerückten Anweisungen ausgeführt, die der inneren `else`-Anweisung folgen.

**Mehrfach einrücken** Zu beachten sind besonders die mehrfachen Einrückungen, damit die Tiefe der Kontrollstruktur von Python richtig erkannt werden kann.

Nach einem Doppelpunkt hinter dem Kopf einer Kontrollstruktur wird in der Entwicklungsumgebung IDLE automatisch mit einem Tabulatorsprung eingerückt. Dieser Sprung erzeugt standardmäßig vier Leerzeichen, dadurch werden die Kontrollstrukturen für den Entwickler klar erkennbar.

**Mindestens ein Leerzeichen** Falls Sie mit einem anderen Editor arbeiten, müssen Sie darauf achten, dass um mindestens ein Leerzeichen eingerückt wird, damit Python die Kontrollstruktur erkennt. Sinnvoller ist eine Einrückung um zwei oder mehr Leerzeichen, damit die Struktur auch für den Entwickler gut erkennbar ist.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 3.4.4 Spiel, Version mit `for`-Schleife und Abbruch

**Vier Versuche** Die `for`-Schleife wird nun dazu genutzt, die Eingabe und die Bewertung insgesamt viermal zu durchlaufen. Der Benutzer hat also vier Versuche, das richtige Ergebnis zu ermitteln.

Die Anweisung `break` dient zum Abbruch der Schleife, sobald der Benutzer das richtige Ergebnis eingegeben hat.

```
# Zufallsgenerator
import random
random.seed()
```

```

# Werte und Berechnung
a = random.randint(1,10)
b = random.randint(1,10)
c = a + b
print("Die Aufgabe:", a, "+", b)
# Schleife mit for
for i in 1, 2, 3, 4:
    # Eingabe
    print("Bitte eine Zahl eingeben:")
    z = input()
    zahl = int(z)
    # Verzweigung
    if zahl == c:
        print(zahl, "ist richtig")
        # Abbruch der Schleife
        break
    else:
        print(zahl, "ist falsch")
# Ende
print("Ergebnis: ", c)

```

**Listing 3.15** Datei spiel\_for.py

Folgende Ausgabe wird erzeugt:

```

Die Aufgabe: 7 + 9
Bitte eine Zahl eingeben:
12
12 ist falsch
Bitte eine Zahl eingeben:
16
16 ist richtig
Ergebnis: 16

```

Zur Erläuterung:

- ▶ Die Aufgabe wird einmal ermittelt und gestellt.
- ▶ Der Benutzer wird maximal viermal dazu aufgefordert, ein Ergebnis einzugeben. Jede seiner Eingaben wird bewertet.
- ▶ Wird bereits bei einem der ersten drei Versuche das richtige Ergebnis eingegeben, so wird die Schleife vorzeitig abgebrochen. **Vorzeitiger Abbruch**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

### 3.4.5 for-Schleife mit »range()«

`range()` Meist werden Schleifen für regelmäßige Abfolgen von Zahlen genutzt. Dabei erweist sich der Einsatz der Funktion `range()` als sehr nützlich. Ein Beispiel:

```
for i in range(3,11,2):
    print("Zahl:", i, "Quadrat:", i*i)
```

**Listing 3.16** Datei `range_drei.py`

Es wird die Ausgabe erzeugt:

```
Zahl: 3 Quadrat: 9
Zahl: 5 Quadrat: 25
Zahl: 7 Quadrat: 49
Zahl: 9 Quadrat: 81
```

**Ganze Zahlen** Der englische Begriff *range* bedeutet »Bereich«. Innerhalb der Klammern hinter `range` können bis zu drei ganze Zahlen, durch Kommata getrennt, eingetragen werden:

**Beginn** ▶ Die erste ganze Zahl (hier 3) gibt den Beginn des Bereichs an, für den die folgenden Anweisungen ausgeführt werden sollen.

**Ende** ▶ Die zweite ganze Zahl (hier 11) kennzeichnet das Ende des Bereichs. Es ist die erste Zahl, für die die Anweisungen *nicht* mehr ausgeführt werden.

**Schrittweite** ▶ Die dritte ganze Zahl (hier 2) gibt die Schrittweite für die Schleife an. Die Zahlen, für die die Anweisungen ausgeführt werden, stehen zueinander also jeweils im Abstand von +2.

Der Aufruf der Funktion `range()` mit den Zahlen 3, 11 und 2 ergibt somit die Abfolge: 3, 5, 7, 9.

**Schrittweite 1** Wird die Funktion `range()` nur mit zwei Zahlen aufgerufen, so wird eine Schrittweite von 1 angenommen. Ein Beispiel:

```
for i in range(5,9):
    print("Zahl:", i)
```

**Listing 3.17** Datei `range_zwei.py`

Es wird folgende Ausgabe erzeugt:

```
Zahl: 5
Zahl: 6
Zahl: 7
Zahl: 8
```

**Beginn bei 0** Wird die Funktion `range()` sogar nur mit einer Zahl aufgerufen, so wird diese Zahl einfach als die obere Grenze angesehen. Als untere Grenze wird 0 gesetzt. Außerdem wird wiederum eine Schrittweite von 1 angenommen. Ein Beispiel:

```
for i in range(3):
    print("Zahl:", i)
```

**Listing 3.18** Datei range\_eins.py

Die Ausgabe:

```
Zahl: 0
Zahl: 1
Zahl: 2
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### Hinweise

- Bei der Funktion `range()` können eine oder mehrere der drei Zahlen auch negativ sein. Achten Sie aber auf die Eingabe sinnvoller Zahlenkombinationen.
- Die Angabe `range(3,-11,2)` ist nicht sinnvoll, da man von der Zahl +3 in Schritten von +2 nicht zur Zahl –11 gelangt. Python fängt solche Schleifen ab und lässt sie nicht ausführen. Dasselbe gilt für die Zahlenkombination `range(3,11,-2)`.

Nicht sinnvoll

Soll ein regelmäßiger Ablauf von Zahlen mit Nachkommastellen erzeugt werden, so muss die Schleifenvariable entsprechend umgerechnet werden. Ein Beispiel:

Keine ganze Zahl

```
# 1. Version
for x in range(18,22):
    print(x/10)
print()
```

```
# 2. Version
x = 1.8
for i in range(4):
    print(x)
    x = x + 0.1
```

**Listing 3.19** Datei range\_nachkomma.py

Die Ausgabe lautet:

```
1.8
1.9
2.0
2.1
```

```
1.8  
1.9000000000000001  
2.0  
2.1
```

Zur Erläuterung:

- ▶ Es sollen jeweils die Zahlen von 1,8 bis 2,1 in Schritten von 0,1 erzeugt werden.
- Version 1 ▶ In der ersten Version werden dazu die ganzen Zahlen von 18 bis 21 erzeugt und durch 10 geteilt.
- Version 2 ▶ In der zweiten Version wird mit dem ersten Wert begonnen und innerhalb der Schleife jeweils um 0,1 erhöht. Vorher ist zu errechnen, wie häufig die Schleife durchlaufen werden muss.
  - ▶ Beide Versionen haben Vor- und Nachteile für den Entwickler. In der ersten Version erkennen Sie Anfang und Ende der Schleife gut, nicht aber die Schrittweite. In der zweiten Version ist es umgekehrt.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Division muss `x / 10.0` lauten, damit das Ergebnis richtig berechnet wird. Alternativ könnte sie auch `0.1 * x` lauten.

#### Schleife erkennen

#### Übung u\_range

Ermitteln Sie durch Überlegen (nicht durch einen einfachen Aufruf) die Ausgabe des folgenden Programms (Datei `u_range.py`).

```
print("Schleife 1")  
for i in 2, 3, 6.5, -7:  
    print(i)  
print("Schleife 2")  
for i in range(3,11,3):  
    print(i)  
print("Schleife 3")  
for i in range(-3,14,4):  
    print(i)  
print("Schleife 4")  
for i in range(3,-11,-3):  
    print(i)
```

**Übung u\_range\_inch**

Schreiben Sie ein Programm, das die folgende Ausgabe erzeugt (Datei *u\_range\_inch.py*).

```
15 inch = 38.1 cm
20 inch = 50.8 cm
25 inch = 63.5 cm
30 inch = 76.2 cm
35 inch = 88.9 cm
40 inch = 101.6 cm
```

Es handelt sich um eine regelmäßige Liste von Inch-Werten, für die der jeweilige Zentimeter-Wert durch Umrechnung mit dem Faktor 2,54 ermittelt werden soll. Beachten Sie, dass für dieses Programm keine Eingabe durch den Anwender notwendig ist.

### 3.4.6 Spiel, Version mit »range()«

Die Schleife zur Wiederholung der Eingabe wird nun mithilfe von `range()` gebildet. Gleichzeitig haben Sie damit einen Zähler, der die laufende Nummer des Versuchs enthält. Diesen Zähler können Sie verwenden, um dem Benutzer am Ende mitzuteilen, wie viele Versuche er benötigt hat.

Mit Zähler

```
# Zufallsgenerator
import random
random.seed()
# Werte und Berechnung
a = random.randint(1,10)
b = random.randint(1,10)
c = a + b
print("Die Aufgabe:", a, "+", b)
# Schleife mit range
for versuch in range(1,10):
    # Eingabe
    print("Bitte eine Zahl eingeben:")
    z = input()
    zahl = int(z)
    # Verzweigung
    if zahl == c:
        print(zahl, "ist richtig")
        # Abbruch der Schleife
        break
    else:
        print(zahl, "ist falsch")
```

```
# Anzahl ausgeben
print("Ergebnis: ", c)
print("Anzahl Versuche:", versuch)
```

#### **Listing 3.20 Datei spiel\_range.py**

Die Ausgabe lautet:

```
Die Aufgabe: 10 + 5
Bitte eine Zahl eingeben:
13
13 ist falsch
Bitte eine Zahl eingeben:
15
15 ist richtig
Ergebnis: 15
Anzahl Versuche: 2
```

Zur Erläuterung:

- ▶ Der Benutzer hat maximal neun Versuche: `range(1,10)`.
- Zähler** ▶ Die Variable `versuch` dient als Zähler für die Versuche.
- ▶ Nach Eingabe der richtigen Lösung (oder nach vollständigem Durchlauf der Schleife) wird dem Benutzer die Anzahl der Versuche mitgeteilt.

#### **Unterschiede in Python 2**

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

### **3.4.7 while-Schleife**

#### **Bedingte Schleife**

Die `while`-Schleife dient zur bedingungsgesteuerten Wiederholung einer Schleife. Mithilfe des folgenden Programms sollen zufällige Zahlen addiert und ausgegeben werden. Solange die Summe der Zahlen kleiner als 30 ist, wird der Vorgang wiederholt. Ist die Summe gleich oder größer als 30, wird das Programm beendet.

```
# Zufallsgenerator
import random
random.seed()
# Initialisierung
summe = 0
# while-Schleife
while summe < 30:
    zzahl = random.randint(1,8)
```

```
summe = summe + zzahl
print("Zahl:", zzahl, "Zwischensumme:", summe)
print("Ende")
```

**Listing 3.21** Datei schleife\_while.py

Eine mögliche Ausgabe des Programms sähe wie folgt aus:

```
Zahl: 3 Zwischensumme: 3
Zahl: 8 Zwischensumme: 11
Zahl: 5 Zwischensumme: 16
Zahl: 8 Zwischensumme: 24
Zahl: 7 Zwischensumme: 31
Ende
```

Zur Erläuterung:

- ▶ Zunächst wird die Variable für die Summe der Zahlen auf 0 gesetzt.
- ▶ Die while-Anweisung leitet die Schleife ein. Die wörtliche Übersetzung der Zeile lautet: *solange die Summe kleiner als 30 ist*. Dies bezieht sich auf die folgenden, eingerückten Anweisungen.
- ▶ Hinter dem Wort `while` steht (wie bei einer `if`-Anweisung) eine Bedingung, die mithilfe von Vergleichsoperatoren erstellt wird. Auch hier dürfen Sie den : (Doppelpunkt) am Ende der Zeile (wie bei `if-else` und `for`) nicht vergessen.
- ▶ Es wird eine zufällige Zahl ermittelt und zur bisherigen Summe addiert. Die neue Summe errechnet sich also aus der alten Summe plus der eingegebenen Zahl. Die neue Summe wird ausgegeben.
- ▶ Die Anweisung zur Ausgabe des Texts »Ende« wird erst erreicht, wenn die Summe den Wert 30 erreicht oder überschritten hat.

Einrücken

Doppelpunkt

Ende der Schleife

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 3.4.8 Spiel, Version mit while-Schleife und Zähler

Die `while`-Schleife wird nun zur Wiederholung der Eingabe genutzt. Der Benutzer hat unendlich viele Versuche, die Aufgabe zu lösen. Die Variable `versuch` als Zähler für die Versuche muss separat gesteuert werden; sie ergibt sich nicht mehr automatisch als Schleifenvariable.

Solange Eingabe falsch

```
# Zufallsgenerator
import random
random.seed()
```

```

# Werte und Berechnung
a = random.randint(1,10)
b = random.randint(1,10)
c = a + b
print("Die Aufgabe:", a, "+", b)

# Schleife initialisieren
zahl = c + 1

# Anzahl initialisieren
versuch = 0

# Schleife mit while
while zahl != c:
    # Anzahl Versuche
    versuch = versuch + 1

    # Eingabe mit Umwandlung
    print("Bitte eine Zahl eingeben:")
    z = input()
    zahl = int(z)

    # Verzweigung
    if zahl == c:
        print(zahl, "ist richtig")
    else:
        print(zahl, "ist falsch")
# Anzahl ausgeben
print("Ergebnis: ", c)
print("Anzahl Versuche:", versuch)

```

### [Listing 3.22 Datei spiel\\_while.py](#)

Die Ausgabe hat sich nicht geändert.

Zur Erläuterung:

- ▶ Der Benutzer hat unendlich viele Versuche. Die `while`-Schleife läuft, solange die richtige Lösung nicht ermittelt wurde.

**Schleife starten**

- ▶ Die Variable `zahl` wird mit einem Wert vorbesetzt, der dafür sorgt, dass die `while`-Schleife mindestens einmal läuft.

- ▶ Die Variable `versuch` wird mit 0 vorbesetzt und dient als laufende Nummer.

**Anzahl Versuche**

- ▶ Nach Eingabe der richtigen Lösung wird dem Benutzer die Anzahl der Versuche mitgeteilt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

### Übung u\_while

Schreiben Sie ein Programm (Datei `u_while.py`), das den Anwender wiederholt dazu auffordert, einen Wert in Inch einzugeben. Der eingegebene Wert soll anschließend in Zentimeter umgerechnet und ausgegeben werden. Das Programm soll beendet werden, falls der Anwender den Wert 0 eingibt.

Hinweis: Bei einer `while`-Schleife wird immer angegeben, unter welcher Bedingung wiederholt werden soll, und nicht, unter welcher Bedingung beendet werden soll. Daher ist für dieses Programm die umgekehrte Bedingung zu formulieren: *solange die Eingabe ungleich 0 ist.*

Ende mit 0

## 3.5 Entwicklung eines Programms

Bei der Entwicklung Ihrer eigenen Programme sollten Sie Schritt für Schritt vorgehen. Stellen Sie zuerst einige Überlegungen an, wie das gesamte Programm aufgebaut sein sollte, und zwar auf Papier. Aus welchen Teilen sollte es nacheinander bestehen? Versuchen Sie dann nicht, das gesamte Programm mit all seinen komplexen Bestandteilen auf einmal zu schreiben! Dies ist der größte Fehler, den Einsteiger (und manchmal auch Fortgeschrittenen) machen können.

Teile eines Programms

Schreiben Sie zunächst eine einfache Version des ersten Programmteils. Anschließend testen Sie sie. Erst nach einem erfolgreichen Test fügen Sie den folgenden Programmteil hinzu. Nach jeder Änderung testen Sie wiederum. Sollte sich ein Fehler zeigen, so wissen Sie, dass er aufgrund der letzten Änderung aufgetreten ist. Nach dem letzten Hinzufügen haben Sie eine einfache Version Ihres gesamten Programms.

Einfache Version

Nun ändern Sie einen Teil Ihres Programms in eine komplexere Version ab. Auf diese Weise machen Sie Ihr Programm Schritt für Schritt komplexer, bis Sie schließlich das gesamte Programm so erstellt haben, wie es Ihren anfänglichen Überlegungen auf Papier entspricht.

Komplexe Version

Manchmal ergibt sich während der praktischen Programmierung noch die eine oder andere Änderung gegenüber Ihrem Entwurf. Das ist kein Problem, solange sich nicht der gesamte Aufbau ändert. Sollte dies allerdings der Fall sein, so kehren Sie noch einmal kurz zum Papier zurück und überdenken den

Änderungen

Aufbau. Das bedeutet nicht, dass Sie die bisherigen Programmzeilen löschen müssen, sondern möglicherweise nur ein wenig ändern und anders anordnen.

- Einzelne Schritte** Schreiben Sie Ihre Programme übersichtlich. Falls Sie gerade überlegen, wie Sie drei, vier bestimmte Schritte Ihres Programms auf einmal machen können: Machen Sie daraus einfach einzelne Anweisungen, die der Reihe nach ausgeführt werden. Dies vereinfacht eine eventuelle Fehlersuche. Falls Sie (oder eine andere Person) Ihr Programm später einmal ändern oder erweitern möchten, dann gelingt der Einstieg in den Aufbau des Programms wesentlich schneller.
- Kontrolle** Sie können die Funktion `print()` zur Kontrolle von Werten und zur Suche von logischen Fehlern einsetzen. Zusätzlich können Sie einzelne Zeilen Ihres Programms als Kommentar kennzeichnen, um festzustellen, welcher Teil des Programms fehlerfrei läuft und welcher Teil demnach fehlerbehaftet ist.

## 3.6 Fehler und Ausnahmen

- Fehler** Macht der Anwender nach einer Eingabeaufforderung eine falsche Eingabe (z. B. keine Zahl, sondern einen Text), so wird das Programm mit einer Fehlermeldung beendet. Bisher wurde vereinfacht davon ausgegangen, dass der Anwender korrekte Eingaben vornimmt. In diesem Abschnitt beschreibe ich, wie Sie Fehler vermeiden oder abfangen.

### 3.6.1 Basisprogramm

- Abbruch möglich** Durch das Abfangen von falschen Eingaben wird die Benutzung eines Programms für den Anwender deutlich komfortabler. Im folgenden Programm soll eine ganze Zahl eingegeben werden. Da der Benutzer dieses Programm durch die Eingabe von Text oder Sonderzeichen zum Abbruch bringen kann, wird es im weiteren Verlauf des Abschnitts verbessert.

```
# Eingabe
print("Bitte geben Sie eine ganze Zahl ein")
z = input()

# Umwandlung
zahl = int(z)

# Ausgabe
print("Sie haben die ganze Zahl", zahl, "richtig eingegeben")
print("Ende des Programms")
```

**Listing 3.23** Datei `fehler_basis.py`

Gibt der Anwender eine Zahl (z. B. 12) ein, so läuft das Programm fehlerfrei bis zum Ende und erzeugt die folgende Ausgabe:

Bitte geben Sie eine ganze Zahl ein

12

Sie haben die ganze Zahl 12 richtig eingegeben

Ende des Programms

Richtige Eingabe

Macht der Anwender jedoch eine falsche Eingabe (z. B. »3a«), so bricht das Programm vorzeitig ab und erzeugt die folgende Ausgabe:

Bitte geben Sie eine ganze Zahl ein

3a

Traceback (most recent call last):

File "fehler\_basis.py", line 6, in <module>

    zahl = int(z)

ValueError: invalid literal for int() with base 10: '3a'

Falsche Eingabe

Diese Informationen weisen auf die Stelle im Programm hin, an der der Fehler bemerkt wurde (*Dateifehler\_basis.py*, Zeile 6). Außerdem wird die Art des Fehlers mitgeteilt (`ValueError`).

Fehlerstelle

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

## 3.6.2 Fehler abfangen

Der Fehler soll zunächst abgefangen werden, um einen Abbruch des Programms zu vermeiden. Zu diesem Zweck müssen Sie die Stelle herausfinden, an der der Fehler auftrat. An dieser Stelle müssen Sie das Programm verbessern.

Verbessern

Das Programm soll zukünftig an dieser Stelle alle Arten von Fehlern abfangen, die Python automatisch erkennen kann. Dies erreichen Sie in einem ersten Schritt durch die folgende Änderung:

Fehler abfangen

```
# Eingabe
print("Bitte geben Sie eine ganze Zahl ein")
z = input()

# Versuch der Umwandlung
try:
    zahl = int(z)
    print("Sie haben die ganze Zahl", zahl,
          "richtig eingegeben")
# Fehler bei Umwandlung
```

```
except:  
    print("Sie haben die ganze Zahl nicht"  
        " richtig eingegeben")  
  
print("Ende des Programms")
```

**Listing 3.24** Datei fehler\_abfangen.py

**Kein Abbruch mehr** Wenn der Anwender eine richtige ganze Zahl eingibt, läuft das Programm wie bisher. Bei einer falschen Eingabe erscheint nun eine entsprechende Meldung. Das Programm bricht jedoch nicht ab, sondern läuft bis zum Ende.

Bitte geben Sie eine ganze Zahl ein

3a

Sie haben die ganze Zahl nicht richtig eingegeben

Ende des Programms

Zur Erläuterung:

- try** ► Die Anweisung `try` leitet eine Ausnahmebehandlung ein. Ähnlich wie bei der `if`-Anweisung gibt es verschiedene Zweige, die das Programm durchlaufen kann. Das Programm versucht (englisch: `try`), die Anweisungen durchzuführen, die eingerückt hinter `try` stehen.
- Falls die Eingabe erfolgreich ist, wird der `except`-Zweig nicht ausgeführt, ähnlich wie beim `else`-Zweig der `if`-Anweisung.
- except** ► Ist die Eingabe dagegen nicht erfolgreich und handelt es sich um einen Fehler, so wird der Fehler oder die Ausnahme (englisch: `exception`) mit der Anweisung `except` abgefangen. Es werden dann alle eingerückten Anweisungen in diesem Zweig durchgeführt.
- Anschließend läuft das Programm ohne Abbruch zu Ende, da der Fehler zwar auftrat, aber abgefangen wurde.
- Nach `try` und `except` muss jeweils ein Doppelpunkt gesetzt werden, wie bei `if-else`, `for` oder `while`.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

#### Hinweise

##### Kritische Stelle

1. Nur die *kritische* Zeile wurde in die Ausnahmebehandlung eingebettet. Der Programmierer muss sich also Gedanken darüber machen, welche Stellen seines Programms fehlerträchtig sind.

2. Eine Eingabeaufforderung ist solch eine kritische Stelle. Andere Fehlermöglichkeiten sind z. B. die Bearbeitung einer Datei (die möglicherweise nicht existiert) oder die Ausgabe auf einen Drucker (der vielleicht nicht eingeschaltet ist).

Fehlermöglich-keiten

### 3.6.3 Eingabe wiederholen

In einem zweiten Schritt soll dafür gesorgt werden, dass der Anwender nach einer falschen Eingabe eine neue Eingabe machen kann. Der gesamte Eingabevorgang einschließlich der Ausnahmebehandlung wird so lange wiederholt, bis er erfolgreich war. Betrachten Sie dazu das folgende Programm:

Erneute Eingabe

```
# Initialisierung der while-Schleife
fehler = 1

# Schleife bei falscher Eingabe
while fehler == 1:
    # Eingabe
    print("Bitte geben Sie eine ganze Zahl ein")
    z = input()

    # Versuch der Umwandlung
    try:
        zahl = int(z)
        print("Sie haben die ganze Zahl", zahl,
              "richtig eingegeben")
        fehler = 0
    # Fehler bei Umwandlung
    except:
        print("Sie haben die ganze Zahl nicht"
              " richtig eingegeben")

print("Ende des Programms")
```

**Listing 3.25** Datei fehler\_eingabe\_neu.py

Nachfolgend wird eine mögliche Eingabe gezeigt – zunächst mit einem Fehler, anschließend fehlerfrei:

Test

```
Bitte geben Sie eine ganze Zahl ein
3a
Sie haben die ganze Zahl nicht richtig eingegeben
Bitte geben Sie eine ganze Zahl ein
12
Sie haben die ganze Zahl 12 richtig eingegeben
Ende des Programms
```

Zur Erläuterung:

- ▶ Die Variable `fehler` wird zunächst auf den Wert 1 gesetzt. Diese Variable ist notwendig, um die Eingabe gegebenenfalls wiederholen zu können.
- ▶ Mit der `while`-Anweisung wird eine Schleife formuliert, in der der Eingabevorgang einschließlich der Ausnahmebehandlung eingebettet ist. Die Schleife wird wiederholt, solange die Variable `fehler` den Wert 1 hat.
- Eingabe-wiederholung**
- ▶ Ist die Eingabe erfolgreich, so wird die Variable `fehler` auf 0 gesetzt. Dies führt dazu, dass die Schleife beendet wird und das Programm regulär fortfahren kann.
- Richtige Eingabe**
- ▶ Ist die Eingabe nicht erfolgreich, so hat `fehler` nach wie vor den Wert 1. Dies führt dazu, dass der Eingabevorgang wiederholt wird.
- Falsche Eingabe**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### Hinweis

Beachten Sie die doppelte Einrückung: einmal nach `while` und noch einmal nach `try/except`.

### Übung u\_fehler

Verbessern Sie das Programm zur Eingabe und Umrechnung eines beliebigen Inch-Wertes in Zentimeter. Eingabefehler des Anwenders sollen abgefangen werden. Das Programm soll den Anwender so lange zur Eingabe auffordern, bis sie erfolgreich war (Datei `u_fehler.py`).

### 3.6.4 Exkurs: Schleifenfortsetzung mit »continue«

- break** An dieser Stelle möchte ich auf ein Thema zurückkommen, das bereits in Abschnitt 3.4, »Schleifen«, behandelt wurde. Sie kennen aus diesem Abschnitt die Anweisung `break`, die zum unmittelbaren Abbruch einer Schleife führt.
- continue** Die Anweisung `continue` dient zum unmittelbaren Abbruch des aktuellen Durchlaufs einer Schleife. Die Schleife wird anschließend mit dem nächsten Durchlauf fortgesetzt. Betrachten Sie hierzu das folgende Programm:

```
for i in range (1,7):
    print("Zahl:", i)
    if 3 <= i <= 5:
        continue
    print("Quadrat:", i*i)
```

**Listing 3.26** Datei schleife\_continue.py

Die Ausgabe dieses Programms wäre:

```
Zahl: 1
Quadrat: 1
Zahl: 2
Quadrat: 4
Zahl: 3
Zahl: 4
Zahl: 5
Zahl: 6
Quadrat: 36
```

Zur Erläuterung:

- ▶ Die Schleife durchläuft alle Zahlen von 1 bis 6. Die Zahlen werden auch alle ausgegeben.
- ▶ Liegt die aktuelle Zahl zwischen 3 und 5, so wird der Rest der Schleife über-  
gangen und unmittelbar der nächste Schleifendurchlauf begonnen. Ande-  
renfalls wird das Quadrat der Zahl ausgegeben.

**Rest übergehen**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 3.6.5 Spiel, Version mit Ausnahmebehandlung

Die Ausnahmebehandlung und die Anweisung `continue` werden im folgenden Programm dazu eingesetzt, einen Eingabefehler abzufangen und das Programm regulär fortzusetzen.

**Fehler abfangen**

```
# Zufallsgenerator
import random
random.seed()

# Werte und Berechnung
a = random.randint(1,10)
b = random.randint(1,10)
c = a + b

print("Die Aufgabe:", a, "+", b)
```

```

# Schleife und Anzahl initialisieren
zahl = c + 1
versuch = 0

# Schleife mit while
while zahl != c:
    # Anzahl Versuche
    versuch = versuch + 1

    # Eingabe
    print("Bitte eine ganze Zahl eingeben:")
    z = input()

    # Versuch einer Umwandlung
    try:
        zahl = int(z)
    except:
        # Falls Umwandlung nicht erfolgreich
        print("Sie haben keine ganze Zahl eingegeben")
        # Schleife unmittelbar fortsetzen
        continue

    # Verzweigung
    if zahl == c:
        print(zahl, "ist richtig")
    else:
        print(zahl, "ist falsch")

# Anzahl Versuche
print("Ergebnis: ", c)
print("Anzahl Versuche:", versuch)

```

**Listing 3.27 Datei spiel\_ausnahme.py**

Es wird folgende Ausgabe erzeugt:

```

Die Aufgabe: 8 + 3
Bitte eine ganze Zahl eingeben:
12
12 ist falsch
Bitte eine ganze Zahl eingeben:
11a
Sie haben keine ganze Zahl eingegeben
Bitte eine ganze Zahl eingeben:
11
11 ist richtig
Ergebnis: 11
Anzahl Versuche: 3

```

Zur Erläuterung:

- ▶ Die Umwandlung der Eingabe steht in einem try-except-Block.
- ▶ Falls die Umwandlung aufgrund einer falschen Eingabe nicht gelingt, erscheint eine entsprechende Meldung.
- ▶ Der Rest der Schleife wird übergangen, und die nächste Eingabe wird unmittelbar angefordert.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

## 3.7 Funktionen und Module

Die Modularisierung, also die Zerlegung eines Programms in selbst geschriebene Funktionen, bietet besonders bei größeren Programmen unübersehbare Vorteile:

- ▶ Programmteile, die mehrmals benötigt werden, müssen nur einmal definiert werden. Modularisierung
- ▶ Nützliche Programmteile können in mehreren Programmen verwendet werden. Mehrfach verwenden
- ▶ Umfangreiche Programme können in übersichtliche Teile zerlegt werden. Übersichtlicher
- ▶ Pflege und Wartung von Programmen wird erleichtert.
- ▶ Der Programmcode ist für den Programmierer selbst (zu einem späteren Zeitpunkt) und für andere Programmierer leichter zu verstehen. Verständlicher

Neben den selbst geschriebenen Funktionen gibt es in Python, wie bei jeder anderen Programmiersprache auch, zahlreiche vordefinierte Funktionen, die dem Entwickler viel Arbeit abnehmen können. Diese Funktionen sind entweder fest eingebaut oder über die Einbindung spezieller Module verfügbar.

Als Beispiel für eine fest eingebaute Funktion wurde bereits `input()` eingesetzt. Jede Funktion hat eine spezielle Aufgabe. So hält beispielsweise die Funktion `input()` das Programm an und nimmt eine Eingabe entgegen.

Wie viele (aber nicht alle) Funktionen hat `input()` einen sogenannten Rückgabewert, liefert also ein Ergebnis an die Stelle des Programms zurück, von der sie aufgerufen wurde: die eingegebene Zeichenkette.

Die Entwicklung selbst geschriebener Funktionen erläutert der folgende Abschnitt.

### 3.7.1 Einfache Funktionen

**Immer gleich** Einfache Funktionen führen bei Aufruf stets dieselbe Aktion aus. Im folgenden Beispiel führt jeder Aufruf der Funktion `stern()` dazu, dass eine optische Trennung auf dem Bildschirm ausgegeben wird:

```
# Definition der Funktion
def stern():
    print("-----")
    print("*** Trennung ***")
    print("-----")

# Hauptprogramm
x = 12
y = 5
stern()                      # 1. Aufruf
print("x =", x, ", y =", y)
stern()                      # 2. Aufruf
print("x + y =", x + y)
stern()                      # 3. Aufruf
print("x - y =", x - y)
stern()                      # 4. Aufruf
```

**Listing 3.28** Datei `funktion_einfach.py`

Folgende Ausgabe wird erzeugt:

```
-----
*** Trennung ***
-----
x = 12 , y = 5
-----
*** Trennung ***
-----
x + y = 17
-----
*** Trennung ***
-----
x - y = 7
-----
*** Trennung ***
```

**Definition** Zur Erläuterung:

Im oberen Teil des Programms wird die Funktion `stern()` definiert:

- def**
- ▶ Nach der Anweisung `def` folgt der Name der Funktion (hier `stern`), anschließend runde Klammern und der bereits bekannte Doppelpunkt. Innerhalb der Klammern könnten Werte an die Funktion übergeben werden. Hierzu

erfahren Sie mehr in [Abschnitt 3.6.2](#), »Funktionen mit einem Parameter«, und [Abschnitt 3.6.3](#), »Funktionen mit mehreren Parametern«.

- ▶ Die folgenden, eingerückten Anweisungen werden jedes Mal durchgeführt, wenn die Funktion aufgerufen wird.
- ▶ Die eingebaute Funktion `print()` zur Erzeugung einer optischen Trennung wird dreimal aufgerufen.

Eine Funktion wird zunächst nur definiert und nicht durchgeführt. Sie steht sozusagen zum späteren Gebrauch bereit. Im unteren Teil beginnt das eigentliche Programm:

- ▶ Es werden verschiedene Rechenoperationen mit zwei Variablen durchgeführt.
- ▶ Die Ausgabezeilen werden mithilfe der Funktion `stern()` optisch voneinander getrennt.
- ▶ Die Funktion `stern()` wird insgesamt viermal aufgerufen.
- ▶ Nach Bearbeitung der Funktion `stern()` fährt das Programm jedes Mal mit der Anweisung fort, die dem Aufruf der Funktion folgt.

**Bereitstellung****Aufruf****Nach Funktion weiter****Klammern****Name**

Eine Funktion wird aufgerufen, indem Sie ihren Namen, gefolgt von den runden Klammern, notieren. Sollen Informationen an die Funktion geliefert werden, so geben Sie diese innerhalb der runden Klammern an.

Den Namen einer Funktion können Sie weitgehend frei wählen – es gelten die gleichen Regeln wie bei den Namen von Variablen:

- ▶ Der Name kann aus den Buchstaben a bis z, A bis Z, aus Ziffern und dem Zeichen `_` (Unterstrich) bestehen.
- ▶ Der Name darf nicht mit einer Ziffer beginnen.
- ▶ Er darf keinem reservierten Wort von Python entsprechen.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 3.7.2 Funktionen mit einem Parameter

Bei einem Aufruf können auch Informationen an Funktionen übermittelt werden, sogenannte *Parameter*. Dies führt dazu, dass diese Informationen innerhalb der Funktion ausgewertet werden können und bei jedem Aufruf zu unterschiedlichen Ergebnissen führen. Ein Beispiel:

**Parameter**

```
# Definition der Funktion
def quadrat(x):
    q = x*x
    print("Zahl:", x, "Quadrat:", q)
```

```
# Hauptprogramm
quadrat(4.5)
a = 3
quadrat(a)
quadrat(2*a)
```

**Listing 3.29** Datei parameter.py

Die Ausgabe lautet:

Zahl: 4.5 Quadrat: 20.25

Zahl: 3 Quadrat: 9

Zahl: 6 Quadrat: 36

Zur Erläuterung:

- Wertübermittlung** Die Definition der Funktion `quadrat()` enthält in den Klammern eine Variable. Dies bedeutet, dass beim Aufruf der Funktion ein Wert übermittelt und dieser Variablen zugewiesen wird. Im vorliegenden Programm werden der Funktion die folgenden Werte geliefert:

1. Die Zahl 4,5 – die Variable `x` erhält in der Funktion den Wert 4.5.
2. Die Variable `a` – die Variable `x` erhält in der Funktion den aktuellen Wert von `a`, nämlich 3.
3. Das Ergebnis einer Berechnung – die Variable `x` erhält in der Funktion den aktuellen Wert von 2 mal `a`, also 6.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### Hinweis

##### Anzahl Werte

Die Funktion erwartet genau einen Wert. Sie darf also nicht ohne einen Wert oder mit mehr als einem Wert aufgerufen werden, sonst bricht das Programm mit einer Fehlermeldung ab.

#### Übung u\_parameter

Es soll für verschiedene Bruttogehälter der Steuerbetrag berechnet werden (Datei `u_parameter.py`). Liegt das Gehalt über 2.500 €, so sind 22 % Steuern zu zahlen, ansonsten 18 %. Die Berechnung und die Ausgabe des Steuerbetrags sollen innerhalb einer Funktion mit dem Namen `steuer()` stattfinden. Die Funktion soll für die folgenden Gehälter aufgerufen werden: 1.800 €, 2.200 €, 2.500 €, 2.900 €.

### 3.7.3 Funktionen mit mehreren Parametern

Eine Funktion kann noch vielseitiger werden, falls Sie ihr mehrere Parameter übermitteln. Dabei ist auf die übereinstimmende Anzahl und die richtige Reihenfolge der Parameter zu achten. Ein Beispiel:

Anzahl,  
Reihenfolge

```
# Definition der Funktion
def berechnung(x,y,z):
    ergebnis = (x+y) * z
    print("Ergebnis:", ergebnis)

# Hauptprogramm
berechnung(2,3,5)
berechnung(5,2,3)
```

**Listing 3.30** Datei parameter\_mehrere.py

Die Ausgabe lautet:

Ergebnis: 25  
Ergebnis: 21

Zur Erläuterung:

Es werden genau drei Parameter erwartet, bei beiden Aufrufen werden auch drei Werte übermittelt. Wie Sie am Ergebnis erkennen, ist die Reihenfolge der Parameter wichtig.

Reihenfolge

- ▶ Beim ersten Aufruf erhält  $x$  den Wert 2,  $y$  den Wert 3 und  $z$  den Wert 5. Dies ergibt die Rechnung:  $(2 + 3) \times 5 = 25$ .
- ▶ Beim zweiten Aufruf werden zwar die gleichen Zahlen übergeben, aber in anderer Reihenfolge. Es ergibt sich die Rechnung:  $(5 + 2) \times 3 = 21$ .

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 3.7.4 Funktionen mit Rückgabewert

Funktionen werden häufig zur Berechnung von Ergebnissen eingesetzt. Zu diesem Zweck können Funktionen ihre Ergebnisse als sogenannte *Rückgabewerte* zurückliefern.

Ergebnis

Im Unterschied zu vielen anderen Programmiersprachen können Funktionen in Python mehr als einen Rückgabewert liefern. In diesem Abschnitt sollen allerdings zunächst nur Funktionen betrachtet werden, die genau einen Rückgabewert zur Verfügung stellen.

Rückgabewert

Im folgenden Beispiel wird eine Funktion, die einen Rückgabewert liefert, auf zwei verschiedene Arten im Hauptprogramm eingesetzt und aufgerufen.

```
# Definition der Funktion
def mittelwert(x,y):
    ergebnis = (x+y) / 2
    return ergebnis
# Hauptprogramm
c = mittelwert(3, 9)
print("Mittelwert:", c)
x = 5
print("Mittelwert:", mittelwert(x,4))
```

**Listing 3.31** Datei rueckgabewert.py

Es wird die Ausgabe erzeugt:

**Mittelwert: 6.0**

**Mittelwert: 4.5**

Zur Erläuterung:

- |                           |  |
|---------------------------|--|
| <b>return</b>             | ► Innerhalb der Funktion wird das Ergebnis zunächst berechnet. Anschließend wird es mithilfe der Anweisung <code>return</code> an die aufrufende Stelle zurückgeliefert. Die Anweisung <code>return</code> beendet außerdem unmittelbar den Ablauf der Funktion. |
| <b>Rückgabe speichern</b> | ► Beim ersten Aufruf wird der Rückgabewert in der Variablen <code>c</code> zwischengespeichert. Er kann im weiteren Verlauf des Programms an beliebiger Stelle verwendet werden.   |
| <b>Rückgabe ausgeben</b>  | ► Beim zweiten Aufruf geschehen zwei Dinge gleichzeitig: Die Funktion <code>mittelwert()</code> wird aufgerufen und liefert ein Ergebnis. Dieses Ergebnis wird unmittelbar ausgegeben.   |

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. In der Funktion muss durch 2.0 geteilt werden, damit das Divisionsergebnis richtig berechnet wird.

### 3.7.5 Spiel, Version mit Funktionen

#### Mit Funktionen

Das Programm umfasst nun zwei Funktionen; die erste Funktion dient zur Ermittlung der Aufgabe, die zweite zur Bewertung der Eingabe:

```
# Aufgabe
def aufgabe():
    a = random.randint(1,10)
    b = random.randint(1,10)
```

```

erg = a + b
print("Die Aufgabe:", a, "+", b)
return erg

# Kommentar
def kommentar(eingabezahl, ergebnis):
    if eingabezahl == ergebnis:
        print(eingabezahl, "ist richtig")
    else:
        print(eingabezahl, "ist falsch")
# Zufallsgenerator
import random
random.seed()

# Aufgabe
c = aufgabe()

# Schleife und Anzahl initialisieren
zahl = c + 1
versuch = 0

# Schleife mit while
while zahl != c:
    # Anzahl Versuche
    versuch = versuch + 1

    # Eingabe
    print("Bitte eine Zahl eingeben:")
    z = input()

    # Versuch einer Umwandlung
    try:
        zahl = int(z)
    except:
        # Falls Umwandlung nicht erfolgreich
        print("Sie haben keine Zahl eingegeben")
        # Schleife unmittelbar fortsetzen
        continue

    # Kommentar
    kommentar(zahl,c)

# Anzahl Versuche
print("Ergebnis: ", c)
print("Anzahl Versuche: ", versuch)

```

### **Listing 3.32 Datei spiel\_funktion.py**

Die Ausgabe hat sich nicht geändert.

Zur Erläuterung:

- aufgabe()** ► In der Funktion `aufgabe()` werden die beiden Zufallszahlen ermittelt, und die Aufgabe wird auf dem Bildschirm ausgegeben. Außerdem wird das Ergebnis der Aufgabe als Rückgabewert an das Hauptprogramm zurückgeliefert.
- kommentar()** ► Der Funktion `kommentar()` werden zwei Zahlen als Parameter übermittelt: die Lösung des Anwenders und das richtige Ergebnis. Innerhalb der Funktion wird die eingegebene Lösung untersucht, und ein entsprechender Kommentar wird ausgegeben. Die Funktion hat keinen Rückgabewert.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

#### Übung `u_rueckgabewert`

Das Programm aus Übung `u_parameter` soll umgeschrieben werden. Innerhalb der Funktion `steuer()` soll der Steuerbetrag nur berechnet und an das Hauptprogramm zurückgeliefert werden. Die Ausgabe des ermittelten Werts soll im Hauptprogramm stattfinden (Datei `u_rueckgabewert.py`).

## 3.8 Das fertige Spiel

**Anzahl bestimmen** Zum Abschluss des Programmierkurses erweitern wir das Spiel noch ein wenig – dabei nutzen Sie die Programmiermittel, die Ihnen inzwischen zur Verfügung stehen. Bei der erweiterten Version des Spiels sollen bis zu zehn Aufgaben nacheinander gestellt werden. Der Benutzer kann dabei die Anzahl selbst bestimmen.

- Alle Grundrechenarten** ► Zusätzlich zur Addition kommen nun auch die anderen drei Grundrechenarten zum Einsatz: Subtraktion, Multiplikation und Division.

- Zahlenbereiche** ► Die Bereiche, aus denen die zufälligen Zahlen gewählt werden, hängen von der Rechenart ab. Bei der Multiplikation wird z.B. mit kleineren Zahlen gerechnet als bei der Addition.
- Der Benutzer hat maximal drei Versuche pro Aufgabe.
- Die Anzahl der richtig gelösten Aufgaben wird gezählt.

Die einzelnen Abschnitte des Programms sind nummeriert. Diese Nummern finden sich in der Erläuterung wieder.

In späteren Kapiteln kommen weitere Ergänzungen hinzu. Das Programm sieht nun wie folgt aus:

```
# 1: Zufallsgenerator
import random
random.seed()
# 2: Anzahl Aufgaben
anzahl = -1
while anzahl<0 or anzahl>10:
    try:
        print("Wie viele Aufgaben (1 bis 10):")
        anzahl = int(input())
    except:
        continue

# 3: Anzahl richtige Ergebnisse
richtig = 0

# 4: Schleife mit "anzahl" Aufgaben
for aufgabe in range(1,anzahl+1):

    # 5: Operatorauswahl
    opzahl = random.randint(1,4)

    # 6: Operandenauswahl
    if(opzahl == 1):
        a = random.randint(-10,30)
        b = random.randint(-10,30)
        op = "+"
        c = a + b
    elif(opzahl == 2):
        a = random.randint(1,30)
        b = random.randint(1,30)
        op = "-"
        c = a - b
    elif(opzahl == 3):
        a = random.randint(1,10)
        b = random.randint(1,10)
        op = "*"
        c = a * b
    elif(opzahl == 4):
        c = random.randint(1,10)
        b = random.randint(1,10)
        op = "/"
        a = c * b

    # 7: Sonderfall Division
```

```

# 8: Aufgabenstellung
print("Aufgabe", aufgabe, "von",
      anzahl, ":", a, op, b)
# 9: Schleife mit 3 Versuchen
for versuch in range(1,4):

    # 10: Eingabe
    try:
        print("Bitte eine Zahl eingeben:")
        zahl = int(input())
    except:
        # Falls Umwandlung nicht erfolgreich
        print("Sie haben keine Zahl eingegeben")
        # Schleife unmittelbar fortsetzen
        continue

    # 11: Kommentar
    if zahl == c:
        print(zahl, "ist richtig")
        richtig = richtig + 1
        break
    else:
        print(zahl, "ist falsch")

    # 12: Richtiges Ergebnis der Aufgabe
    print("Ergebnis: ", c)

# 13: Anzahl richtige Ergebnisse
print("Richtig:", richtig, "von", anzahl)

```

### **Listing 3.33 Datei spiel\_fertig.py**

Es wird die folgende Ausgabe erzeugt:

**Wie viele Aufgaben (1 bis 10):**

**2**

**Aufgabe 1 von 2 : 26 + 18**

**Bitte eine Zahl eingeben:**

**44**

**44 ist richtig**

**Ergebnis: 44**

**Aufgabe 2 von 2 : 27 – 2**

**Bitte eine Zahl eingeben:**

**24**

**24 ist falsch**

**Bitte eine Zahl eingeben:**

**23**

**23 ist falsch**

**Bitte eine Zahl eingeben:**

22

22 ist falsch

Ergebnis: 25

Richtig: 1 von 2

Zur Erläuterung:

- ▶ Nach der Initialisierung des Zufallsgenerators (1) wird die gewünschte Anzahl der Aufgaben eingelesen (2). Da der Benutzer einen Fehler bei der Eingabe machen könnte, findet eine Ausnahmebehandlung statt. Fehler abfangen
- ▶ Der Rückgabewert der Funktion `input()` wird unmittelbar als Parameter der Funktion `int()` genutzt. So können zwei Schritte auf einmal erledigt werden. Eingabe,  
Umwandlung
- ▶ Der Zähler für die Anzahl der richtig gelösten Aufgaben (`richtig`) wird auf 0 gestellt (3).
- ▶ Es wird eine äußere `for`-Schleife mit der gewünschten Anzahl gestartet (4). Zufälliger Operator
- ▶ Der Operator wird per Zufallsgenerator ermittelt (5).
- ▶ Für jeden Operator gibt es andere Bereiche, aus denen die Zahlen ausgewählt werden (6). Der Operator selbst und das Ergebnis werden gespeichert. Nur ganze Zahlen
- ▶ Eine Besonderheit ist bei der Division zu beachten (7): Es sollen nur ganze Zahlen vorkommen. Die beiden zufälligen Operanden (`a` und `b`) werden daher aus dem Ergebnis einer Multiplikation ermittelt.
- ▶ Die Aufgabe wird gestellt (8). Dabei werden zur besseren Orientierung des Benutzers auch die laufende Nummer und die Gesamtanzahl der Aufgaben ausgegeben. Maximal drei  
Versuche
- ▶ Es wird eine innere `for`-Schleife für maximal drei Versuche gestartet (9).
- ▶ Die Eingaben des Benutzers (10) werden kommentiert (11). Nach maximal drei Versuchen wird das richtige Ergebnis ausgegeben (12).
- ▶ Als Endergebnis wird die Anzahl der richtig gelösten Aufgaben ausgegeben (13). Endergebnis

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen `\` für den Umbruch von langen Programmzeilen eingesetzt.



# Kapitel 4

## Datentypen

Dieses Kapitel beschäftigt sich mit den Eigenschaften und Vorteilen der verschiedenen Objekttypen. Es werden Operationen, Funktionen und Operatoren für die jeweiligen Datentypen vorgestellt. Ein eigener Abschnitt über Objektreferenzen und Objektidentität vervollständigt die Objektbetrachtung.

Alle Daten werden in Python als Objekte gespeichert. Man kann dabei zwei Arten von Objekttypen unterscheiden, nämlich zum einen die,

Alles ist ein Objekt

- ▶ die einzelne Objekte enthalten, wie z. B. Zahlen oder Zeichen, und zum anderen die,
- ▶ die eine zusammengehörige Gruppe von Objekten enthalten, wie z. B. Strings (= Zeichenketten), Listen, Tupel, Dictionary und Sets.

In diesem Kapitel geht es zunächst um Zahlen. Später folgen die anderen Objekttypen. Dabei stelle ich auch die Gemeinsamkeiten und Unterschiede der Objekttypen vor.

### 4.1 Zahlen

Ganze Zahlen, Zahlen mit Nachkommastellen, Brüche und Operationen mit Zahlen sind Thema dieses Abschnitts.

#### 4.1.1 Ganze Zahlen

Es gibt in Python 3 einen Objekttyp für ganze Zahlen: den Typ `int` (von englisch *integer* für *ganzahlig*). Zahlen dieses Typs sind unendlich genau.

Typ `int`

Zahlen des Typs `int` können Sie mithilfe von vier verschiedenen Zahlensystemen direkt verarbeiten. Üblicherweise wird das dezimale Zahlensystem mit der Basis 10 benutzt. Außerdem stehen in Python die folgenden Zahlensysteme zur Verfügung:

Zahlensysteme

- ▶ das duale Zahlensystem (mit der Basis 2)
- ▶ das oktale Zahlensystem (mit der Basis 8)
- ▶ das hexadezimale Zahlensystem (mit der Basis 16)

Ein Beispiel:

```
a = 27
print("Dezimal:", a)
print("Hexadezimal:", hex(a))
```

```

print("Oktal:", oct(a))
print("Dual:", bin(a))

b = 0x1a + 12 + 0b101 + 0o67
print("Summe:", b)

```

**Listing 4.1** Datei `zahl_ganz.py`

Folgende Ausgabe wird erzeugt:

```

Dezimal: 27
Hexadezimal: 0x1b
Oktal: 0o33
Dual: 0b11011
Summe: 98

```

Zur Erläuterung:

- ▶ Die dezimale Zahl 27 wird in die drei anderen Zahlensysteme umgerechnet und ausgegeben.

- |              |  |
|--------------|--|
| <b>hex()</b> | ▶ Die Funktion <code>hex()</code> dient zur Umrechnung und Ausgabe der Zahl in das hexadezimale System. Dieses System hat neben den Ziffern 0 bis 9 die Ziffern a bis f für die Werte von 10 bis 15. Die Zahl 0x1b entspricht dem Wert (in Worten) 1 mal 16 hoch 1 + b (= 11) mal 16 hoch 0. |
| <b>oct()</b> | ▶ Zur Umrechnung und Ausgabe der Zahl in das oktale System dient die Funktion <code>oct()</code> . Das oktale System hat nur die Ziffern 0 bis 7. Die Zahl 0o33 entspricht dem Wert (in Worten) 3 mal 8 hoch 1 + 3 mal 8 hoch 0.   |
| <b>bin()</b> | ▶ Die Funktion <code>bin()</code> dient zur Umrechnung und Ausgabe der Zahl in das duale System. Dieses System hat nur die Ziffern 0 und 1. Die Zahl 0b11011 entspricht dem Wert (in Worten) 1 mal 2 hoch 4 + 1 mal 2 hoch 3 + 0 mal 2 hoch 2 + 1 mal 2 hoch 1 + 1 mal 2 hoch 0.             |

Sie können auch direkt mit Zahlen in anderen Zahlensystemen rechnen. Die Berechnung der Variablen `b` ergibt:

$$\begin{aligned}
 & 0x1a + 0b101 + 0o67 = \\
 & (1 \text{ mal } 16 \text{ hoch } 1 + a (= 10) \text{ mal } 16 \text{ hoch } 0) + \\
 & (1 \text{ mal } 10 \text{ hoch } 1 + 2 \text{ mal } 10 \text{ hoch } 0) + \\
 & (1 \text{ mal } 2 \text{ hoch } 2 + 0 \text{ mal } 2 \text{ hoch } 1 + 1 \text{ mal } 2 \text{ hoch } 0) + \\
 & (6 \text{ mal } 8 \text{ hoch } 1 + 7 \text{ mal } 8 \text{ hoch } 0) = \\
 & (16 + 10) + 10 + 2 + (4 + 0 + 1) + (48 + 7) = 98
 \end{aligned}$$

**Erkennung** Bei der Eingabe oder Zuweisung muss `0x`, `0b` bzw. `0o` vor der eigentlichen Zifferfolge stehen, damit das zugehörige Zahlensystem erkannt wird.

**Bits und Bytes** Zahlen setzen sich auf der niedrigsten Ebene aus Bits und Bytes zusammen. In Abschnitt 4.1.6, »Bitoperatoren«, werden Sie noch ein wenig intensiver mit Dualzahlen, der Funktion `bin()` und den sogenannten Bitoperatoren arbeiten, die Ihnen den Zugriff auf Bitebene erleichtern.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Oktalzahl wird ohne das kleine `o` ausgegeben, also nur `033`. Bei der Eingabe oder Zuweisung einer Oktalzahl sind beide Versionen möglich, also `0o33` und `033`.

## 4.1.2 Zahlen mit Nachkommastellen

Der Datentyp für Zahlen mit Nachkommastellen heißt `float`. Diese sogenannten Fließkommazahlen werden mithilfe des Dezimalpunkts und gegebenenfalls der Exponentialschreibweise angegeben. Dazu ein kleines Beispiel:

```
a = 7.5
b = 2e2
c = 3.5E3
d = 4.2e-3
```

```
print(a, b, c, d)
```

**Listing 4.2** Datei `zahl_nachkomma.py`

Die Ausgabe lautet:

`7.5 200.0 3500.0 0.0042`

Zur Erläuterung:

- ▶ Die Variable `a` erhält den Wert `7.5`. Beachten Sie, dass Nachkommastellen mit einem Dezimalpunkt abgetrennt werden. Dies gilt auch für die Eingabe einer Zahl mit Nachkommastellen mithilfe der Funktion `input()`.
- ▶ Die Variable `b` erhält den Wert `200` ( $= 2 \text{ mal } 10 \text{ hoch } 2 = 2 \text{ mal } 100$ ).
- ▶ Die Variable `c` erhält den Wert `3.500` ( $= 3,5 \text{ mal } 10 \text{ hoch } 3 = 3,5 \text{ mal } 1.000$ ).
- ▶ Die Variable `d` erhält den Wert `0,0042` ( $= 4,2 \text{ mal } 10 \text{ hoch } 3 = 4,2 \text{ mal } 0,001$ ).

**Typ float**

**Dezimalpunkt**

Bei der Zuweisung in Exponentialschreibweise wird mithilfe des `e` (oder `E`) ausgedrückt, um wie viele Stellen und in welche Richtung der Dezimalpunkt innerhalb der Zahl verschoben wird. Diese Schreibweise eignet sich z. B. für sehr große oder sehr kleine Zahlen, da sie die Eingabe vieler Nullen erspart.

**Exponentialschreibweise**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 4.1.3 Operator \*\*

**Operator \*\*** Neben den bereits behandelten Rechenoperatoren + (Addition), – (Subtraktion), \* (Multiplikation), / (Division) und % (Modulo, Rest einer Ganzzahldivision) wird der Operator \*\* (Potenz) eingesetzt.

Ein Beispiel:

```
z = 5 ** 3
print("5 hoch 3 =", z)
z = 5.2 ** 3
print("5.2 hoch 3 =", z)
z = -5.2 ** 3
print("-5.2 hoch 3 =", z)
z = 5.2 ** 3.8
print("5.2 hoch 3.8 =", z)
```

**Listing 4.3** Datei `zahl_hoch.py`

Es wird die Ausgabe erzeugt:

```
5 hoch 3 = 125
5.2 hoch 3 = 140.608
-5.2 hoch 3 = -140.608
5.2 hoch 3.8 = 525.7904646699526
```

Zur Erläuterung:

- ▶ Der Variablen `z` wird nacheinander das Ergebnis verschiedener Exponentialrechnungen zugewiesen. Anschließend wird der jeweils aktuelle Wert von `z` mit Kommentar ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 4.1.4 Rundung und Konvertierung

Es gibt eine Reihe fest eingebauter Funktionen für Zahlen. Eine Liste mit vielen fest eingebauten Funktionen inklusive einer kurzen Beschreibung finden Sie in Abschnitt 5.8, »Eingebaute Funktionen«.

**round()** Als Anwendungsbeispiel für die fest eingebauten Funktionen soll die Funktion `round()` zur Rundung einer Zahl dienen – im Unterschied zu der bereits bekannten Funktion `int()` zur Konvertierung (Umwandlung) in eine ganze Zahl.

Das Programm:

```
# Positive Zahl
x = 12/7
print("x:", x)
```

```

# Rundung und Konvertierung
rx = round(x,3)
print("x gerundet auf drei Stellen: ", rx)
rx = round(x)
print("x gerundet auf null Stellen: ", rx)
ix = int(x)
print("int(x):", ix)
print()

# Negative Zahl
x = -12/7
print("x:", x)

# Rundung und Konvertierung
rx = round(x,3)
print("x gerundet auf drei Stellen: ", rx)
rx = round(x)
print("x gerundet auf null Stellen: ", rx)
ix = int(x)
print("int(x):", ix)

```

#### **Listing 4.4 Datei zahl\_umwandeln.py**

Es wird folgende Ausgabe erzeugt:

```

x: 1.7142857142857142
x gerundet auf drei Stellen: 1.714
x gerundet auf null Stellen: 2
int(x): 1

x: -1.7142857142857142
x gerundet auf drei Stellen: -1.714
x gerundet auf null Stellen: -2
int(x): -1

```

Zur Erläuterung:

- ▶ Es wird die Division  $12/7$  ausgeführt. Das Rechenergebnis wird anschließend auf drei verschiedene Arten umgewandelt:

- Mithilfe der eingebauten Funktion `round()` wird das Ergebnis auf drei Stellen nach dem Komma gerundet. Runden auf drei Stellen
- Mit der gleichen Funktion wird das Ergebnis auf die nächsthöhere oder nächstniedrigere ganze Zahl gerundet. Runden ohne Kommastellen
- Mithilfe der eingebauten Funktion `int()` wird das Ergebnis in eine ganze Zahl umgewandelt. Dabei werden – im Unterschied zum Runden – die Stellen nach dem Komma einfach abgeschnitten. Stellen abschneiden

- ▶ Die gleichen Operationen werden mit einer negativen Zahl mit Nachkommastellen durchgeführt.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Division muss `12.0/7` oder `12/7.0` lauten, damit das Ergebnis richtig berechnet wird. Nach der Rundung auf null Stellen wird daher `2.0` bzw. `-2.0` ausgegeben.

### 4.1.5 Modul »math«

**Mathematische Funktionen** Das Modul `math` enthält eine Reihe von mathematischen Funktionen für Zahlen.

**`sin()`, `cos()`, `tan()`** Als Beispiel für eine dieser Funktionen dienen im Folgenden die trigonometrischen Funktionen `sin()`, `cos()` und `tan()`. Das Modul `math` enthält außerdem die mathematischen Konstanten `pi` und `e` (Euler'sche Zahl). Das Programm lautet:

```
# Modul math
import math

# Trigonom. Funktionen und Konstanten
x = 30
xbm = x / 180 * math.pi
print("Sinus ", x, "Grad:", math.sin(xbm))
print("Cosinus", x, "Grad:", math.cos(xbm))
print("Tangens", x, "Grad:", math.tan(xbm))
```

**Listing 4.5** Datei `zahl_math.py`

Es wird die Ausgabe erzeugt:

```
Sinus 30 Grad: 0.4999999999999994
Cosinus 30 Grad: 0.8660254037844387
Tangens 30 Grad: 0.5773502691896257
```

Zur Erläuterung:

- ▶ Das Modul `math` wird importiert.
- ▶ Sinus, Kosinus und Tangens des Winkels 30 Grad werden berechnet.
- ▶ Alle Funktionen beziehen sich auf eine Angabe des Winkels im Bogenmaß, daher wird der Winkel mithilfe der Konstanten `math.pi` in das Bogenmaß umgewandelt.
- ▶ Die drei Funktionen werden angewendet, das Ergebnis wird ausgegeben.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Division muss `x / 180.0 * math.pi` lauten, damit das Divisionsergebnis richtig berechnet wird.

Es ist häufig nützlich zu wissen, ob es sich bei einer Zahl um eine ganze Zahl (Datentyp `int`) oder eine Fließkommazahl (Datentyp `float`) handelt. Die Funktion `type()` gibt den Typ (die Klasse) eines Objekts aus. Hierzu ein Programmbeispiel:

```
a = 2
print("Typ:", type(a))
b = 12/6
print("Typ:", type(b))
print("Modulo liefert:", 12 %6==0)
```

**Listing 4.6** Datei `zahl_type.py`

Das Programm erzeugt die Ausgabe:

```
Typ: <class 'int'>
Typ: <class 'float'>
Modulo liefert: True
```

Zur Erläuterung:

- ▶ Die Variable `a` enthält den Wert 2 und ist vom Typ `int`.
- ▶ Die Variable `b` enthält den gleichen Wert, allerdings als Ergebnis von `12/6`. Es handelt sich um ein Objekt vom Typ `float`.
- ▶ Die Information, dass ein Ergebnis ganzzahlig ist, erhalten Sie mithilfe des Operator % Modulo-Operators (%).

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion `type()` liefert `type` statt `class`. Der Ausdruck `12/6` ist vom Typ `int`, da es sich um eine ganzzahlige Division handelt. Der Ausdruck `12.0/6` oder `12/6.0` ist vom Typ `float`, wie in Python 3.

## 4.1.6 Bitoperatoren

Sämtliche Daten, ob nun Zahlen oder Zeichenketten, setzen sich auf der Hardware-Ebene aus Bits und Bytes zusammen. In diesem Zusammenhang können Sie mit Dualzahlen, der Funktion `bin()` und den sogenannten Bitoperatoren arbeiten, die Ihnen den Zugriff auf Bitebene erleichtern. Bitoperatoren benötigen Sie meist nur dann, wenn Sie Operationen auf der Ebene der Hardware durchführen möchten.

**Bits und Bytes**

Ein Beispiel zu Bitoperatoren:

```
# Nur 1 Bit gesetzt
bit0 = 1          # 0000 0001
bit3 = 8          # 0000 1000
```

```

print(bin(bit0), bin(bit3))

# Bitweise AND
a = 5          # 0000 0101
erg = a & bit0  # 0000 0001
if erg:
    print(a, "ist ungerade")

# Bitweise OR
erg = 0          # 0000 0000
erg = erg | bit0 # 0000 0001
erg = erg | bit3 # 0000 1001
print("Bits nacheinander gesetzt:", erg, bin(erg))

# Bitweise Exclusive-OR
a = 21         # 0001 0101
b = 19         # 0001 0011
erg = a ^ b    # 0000 0110
print("Ungleiche Bits:", erg, bin(erg))

# Bitweise Inversion, aus x wird -(x+1)
a = 11         # 0000 1011
erg = ~a        # 1111 0100
print("Bitweise Inversion:", erg, bin(erg))

# Bitweise schieben
a = 11         # 0000 1011
erg = a >> 1   # 0000 0101
print("Um 1 nach rechts geschoben:", erg, bin(erg))
erg = a << 2   # 0010 1100
print("Um 2 nach links geschoben:", erg, bin(erg))

```

#### **Listing 4.7 Datei bit\_operator.py**

Es folgt die Ausgabe:

```

0b1 0b1000
5 ist ungerade
Bits nacheinander gesetzt: 9 0b1001
Ungleiche Bits: 6 0b110
Bitweise Inversion: -12 -0b1100
Um 1 nach rechts geschoben: 5 0b101
Um 2 nach links geschoben: 44 0b101100

```

Zur Erläuterung:

- Bits gesetzt** ► Zunächst werden die beiden Variablen `bit0` und `bit3` eingeführt, die bei einigen der nachfolgenden Berechnungen benötigt werden. Sie haben die Werte 1 und 8. Am Ende der Programmzeile sehen Sie sie als Dualzahl, also mithilfe von 8 Bit (= 1 Byte) notiert. Das letzte Bit eines Byte wird Bit 0 genannt, das

vorletzte Bit ist Bit 1 usw. Die Werte der beiden Variablen `bit0` und `bit3` sind so gewählt, dass jeweils nur ein Bit gesetzt ist (= 1), die restlichen Bit sind nicht gesetzt (= 0).

- ▶ Sie können sich auch eine Reihe von acht Leuchtdioden (LED) vorstellen. Sie sind entweder *an* oder *aus*. Diese Information kann innerhalb eines Byte gespeichert werden. Falls eines seiner Bits gesetzt ist, dann ist die betreffende LED *an*, ansonsten *aus*.
- ▶ Zur Verdeutlichung werden die beiden Variablen mithilfe der Funktion `bin()` als Dualzahl ausgegeben.
- ▶ Sie können den Bitoperator `&` zur bitweisen Und-Verknüpfung zweier Zahlen nutzen. Ähnlich wie beim logischen Operator `and` (siehe [Abschnitt 3.3.5](#)) wird ein bestimmtes Bit im Ergebnis nur dann gesetzt, wenn dieses Bit in beiden Zahlen gesetzt ist. Diese Operation wird für jedes einzelne Bit durchgeführt.
- ▶ Falls Sie wissen möchten, ob ein bestimmtes Bit innerhalb einer Zahl gesetzt ist, dann verknüpfen Sie diese Zahl mithilfe des Bitoperators `&` mit einer anderen Zahl, in der nur dieses eine gesuchte Bit gesetzt ist. Falls es sich um das Bit 0 handelt, dann wissen Sie darüber hinaus, ob die Zahl gerade (Bit 0 = 0) oder ungerade (Bit 0 = 1) ist.
- ▶ Der Bitoperator `|` dient zur bitweisen Oder-Verknüpfung zweier Zahlen. Ähnlich wie beim logischen Operator `or` (siehe ebenfalls [Abschnitt 3.3.5](#)) wird ein bestimmtes Bit im Ergebnis gesetzt, wenn dieses Bit in einer der beiden Zahlen oder in beiden Zahlen gesetzt ist. Diese Operation wird auch für jedes einzelne Bit durchgeführt.
- ▶ Falls Sie einzelne Bits einer Zahl setzen möchten, dann verknüpfen Sie diese Zahl mithilfe des Bitoperators `|` mit einer anderen Zahl, in der nur dieses eine gesuchte Bit gesetzt ist.
- ▶ Der Bitoperator `^` dient zur bitweisen Exklusiv-Oder-Verknüpfung zweier Zahlen. Ein bestimmtes Bit im Ergebnis wird nur dann gesetzt, wenn dieses Bit in einer der beiden Zahlen gesetzt ist. Falls das Bit in beiden Zahlen gesetzt ist, dann wird das Ergebnis-Bit nicht gesetzt. Diese Operation wird ebenfalls für jedes einzelne Bit durchgeführt.
- ▶ Sie können den Bitoperator `~` zur bitweisen Inversion einer Zahl nutzen. Dabei wird aus der Zahl  $x$ , die Zahl  $-(x+1)$ , also z. B. aus 11 wird -12.
- ▶ Die beiden Bitoperatoren `>>` und `<<` dienen zum Schieben von Bits innerhalb einer Zahl.
- ▶ Mithilfe von `>>` werden alle Bits um eine bestimmte Anzahl von Stellen nach rechts geschoben. Die Bits, die dann nach rechts »hinausfallen«, sind verloren. Eine Verschiebung um  $n$  Bit nach rechts entspricht einer ganzzahligen Division der Zahl durch  $2$  hoch  $n$ . Eine Verschiebung um 1 Bit nach rechts entspricht also einer ganzzahligen Division durch 2.

- Mithilfe von `<<` werden alle Bits um eine bestimmte Anzahl von Stellen nach links geschoben. Eine Verschiebung um  $n$  Bit nach links entspricht einer Multiplikation der Zahl mit  $2$  hoch  $n$ . Eine Verschiebung um 1 Bit nach links entspricht also einer Multiplikation mit  $2$ .

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 4.1.7 Brüche

`fractions` Python kann auch mit Brüchen rechnen bzw. Informationen über Brüche zur Verfügung stellen. Dazu wird das Modul `fractions` (deutsch: Brüche) genutzt. Ein Beispiel:

```
# Import des Moduls
import fractions

# Bruch
z = 12
n = 28
print("Bruch:", z, "/", n)

# als Fraction
b1 = fractions.Fraction(z, n)
print("Fraction:", b1)
print("Z, N:", b1.numerator, b1.denominator)
wert = b1.numerator / b1.denominator
print("Wert:", wert)
print()

# Umrechnen
x = 2.375
print("Zahl:", x)
b2 = fractions.Fraction(x)
print("Fraction:", b2)
print()
# ggT: groesster gemeinsamer Teiler
print("Bruch:", z, "/", n)
print("ggT:", fractions.gcd(z,n))
```

**Listing 4.8** Datei `zahl_bruch.py`

Die Ausgabe lautet:

```
Bruch: 12 / 28
Fraction: 3/7
Z, N: 3 7
```

Wert: 0.42857142857142855

Zahl: 2.375

Fraction: 19/8

Bruch: 12 / 28

ggT: 4

Zur Erläuterung:

- ▶ Zunächst wird ein Beispielbruch in der bekannten Form dargestellt. Er wird gebildet aus zwei Zahlen: Zähler und Nenner. Zähler, Nenner
- ▶ Die Funktion `Fraction()` aus dem Modul `fractions` bietet verschiedene Möglichkeiten, einen Bruch zu erzeugen. Genauer gesagt handelt es sich bei `Fraction()` um den Konstruktor der Klasse `Fraction`. Damit wird eine Instanz (ein Objekt) der Klasse erzeugt und eine Referenz auf dieses Objekt zurückgeliefert. Klassen, Instanzen, Konstruktoren und andere Aspekte der objektorientierten Programmierung werden in [Kapitel 6](#), »Objektorientierte Programmierung«, genauer erläutert. `Fraction()`
- ▶ Der Bruch `b1`, der aus 12/28 gebildet wurde, wird bei der Erzeugung automatisch auf 3/7 gekürzt. Bruch kürzen
- ▶ Zähler und Nenner des Bruchs stehen in den Eigenschaften `numerator` und `denominator` einzeln zur Verfügung. Der Wert eines Bruchs lässt sich über diese beiden Eigenschaften berechnen:  $3/7 = 0,428\dots$  numerator, denominator
- ▶ Umgekehrt können Sie auch eine Zahl mit Nachkommastellen in einen Bruch umrechnen. Dazu übergeben Sie die Zahl der Konstruktormethode `Fraction()`: Aus 2.375 wird 19/8. Float-Zahl
- ▶ Die Methode `gcd()` berechnet den größten gemeinsamen Teiler (ggT; engl.: *greatest common divisor*) zweier ganzer Zahlen. Dies ist die größte Zahl, durch die sich die beiden ganzen Zahlen ohne Rest teilen lassen. Für die Zahlen 12 und 28 ist dies die Zahl 4. Mit dem größten gemeinsamen Teiler kann ein Bruch gekürzt werden: Aus 12/28 wird 3/7. `gcd()`

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Division muss `1.0 * b1.numerator / b1.denominator` lauten, damit das richtige Ergebnis berechnet wird.

Brüche können auch dazu dienen, eine Zahl mit Nachkommastellen zu approximieren, also anzunähern. Dazu dient die Methode `limit_denominator()`. Ein Beispiel:

`limit_ denominator()`

```
# Import des Moduls
import fractions

# untersuchte Zahl
x = 1.84953
print("Zahl:", x)

# als Bruch
b3 = fractions.Fraction(x)
print("Fraction:", b3)

# approximiert
b4 = b3.limit_denominator(100)
print("Approximiert auf Nenner max. 100:", b4)

# Genauigkeit
wert = b4.numerator / b4.denominator
print("Wert:", wert)
print("rel. Fehler:", abs((x-wert)/x))
```

**Listing 4.9 Datei zahl\_bruch\_naehern.py**

Die Ausgabe:

**Zahl: 1.84953**  
**Fraction: 8329542618810553/4503599627370496**  
**Approximiert auf Nenner max. 100: 172/93**  
**Wert: 1.8494623655913978**  
**rel. Fehler: 3.656843014286614e-05**

Zur Erläuterung:

- ▶ Es soll die Zahl 1,84953 untersucht werden. Diese entspricht dem Bruch 184953/100000.

**Nenner begrenzen**

- ▶ Mit der Methode `limit_denominator()` wird der Nenner auf die Zahl 100 begrenzt. Es wird dann der Bruch gesucht, der
  - einen Nenner mit dem maximalen Wert 100 hat und
  - der Zahl 1,84953 am nächsten kommt.
- ▶ Im vorliegenden Fall ist der gesuchte Bruch 172/93.
- ▶ Dieser Bruch hat den Wert 1,8494623655913978 und kommt der ursprünglichen Zahl recht nahe.

**Relativer Fehler**

- ▶ Der relative Fehler zwischen diesem Wert und der untersuchten Zahl beträgt  $3.65684301429 \cdot 10^{-5}$ .

**Betrag, `abs()`**

- ▶ Der relative Fehler wird mithilfe der eingebauten Funktion zur Berechnung des Betrags ermittelt (`abs()`). Der Betrag ist bekanntlich der Absolutwert einer Zahl, also die Zahl ohne das Vorzeichen.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Division muss `1.0 * b4.numerator / b4.denominator` lauten, damit das richtige Ergebnis berechnet wird.

## 4.2 Zeichenketten

Zeichenketten sind Sequenzen von einzelnen Zeichen – also Texte. Auch andere Objekttypen gehören zu den Sequenzen. Anhand von Zeichenketten wird im Folgenden eine Einführung in die Sequenzen geboten.

Sequenzen

### 4.2.1 Eigenschaften

Zeichenketten (Strings) sind Objekte des Datentyps `str`. Strings bestehen aus mehreren Zeichen oder Wörtern. Sie werden gekennzeichnet, indem man sie in einfache, doppelte oder dreimal doppelte Hochkommata setzt. Ein Beispiel:

Typ `str`

```
t1 = "Hallo Welt"
t2 = 'Auch das ist eine Zeichenkette'
t3 = """Diese Zeichenkette
    geht ueber
    mehrere Zeilen"""
t4 = 'Hier sind "doppelte Hochkommata" gespeichert'
print("Bitte geben Sie einen Text ein")
t5 = input()
print("t1:", t1)
print("t2:", t2)
print("t3:", t3)
print("t4:", t4)
print("t5:", t5)

print("Typ:", type(t1))
```

**Listing 4.10** Datei `text_eigenschaft.py`

Es wird die Ausgabe erzeugt:

```
Bitte geben Sie einen Text ein
Das ist meine Eingabe
t1: Hallo Welt
t2: Auch das ist eine Zeichenkette
t3: Diese Zeichenkette
    steht in
    mehreren Zeilen
t4: Hier sind "doppelte Hochkommata" gespeichert
```

`t5: Das ist meine Eingabe`

`Typ: <class 'str'>`

Zur Erläuterung:

- ▶ Die Zeichenkette `t1` ist in doppelten Hochkommata gespeichert.
- ▶ Die Zeichenkette `t2` ist in einfachen Hochkommata gespeichert.
- ▶ Die Zeichenkette `t3` ist in dreifachen Hochkommata gespeichert, sie darf sich deshalb über mehrere Zeilen erstrecken und wird auch in mehreren Zeilen ausgegeben.
- ▶ Die Zeichenkette `t4` verdeutlicht den Vorteil, den das Vorhandensein mehrerer Alternativen bietet: Die doppelten Hochkommata sind hier Bestandteil des Texts und werden auch ausgegeben.
- ▶ Die eingebaute Funktion `input()` ist bereits bekannt. Sie dient zur Eingabe von Zeichenketten. Sie liefert als Ergebnis den eingegebenen Text zurück. Er wird hier in der Variablen `t5` gespeichert.
- ▶ Mithilfe der Funktion `type()` wird für die Zeichenkette `t1` der Objekttyp (`str`) ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Die Funktion `type()` liefert `type` statt `class`.

## 4.2.2 Operatoren

Operatoren  
+, \*, in

Die Operatoren `+` und `*` dienen zur Verkettung mehrerer Sequenzen bzw. zur Vervielfachung einer Sequenz. Mithilfe des Operators `in` stellen Sie fest, ob ein bestimmtes Element in einer Sequenz enthalten ist. Betrachten Sie das folgende Beispiel für diese Operatoren, angewendet für Strings:

```
# Operatoren + und *
t1 = "Teil 1"
t2 = "Teil 2"
tgesamt = t1 + ", " + t2

t3 = "-ooooo-"
t4 = "***"
tlinie = t4 + t3 * 3 + t4

print(tgesamt)
print(tlinie)

# Operator in
tname = "Robinson Crusoe"
```

```

print("Text:", tname)

if "b" in tname:
    print("b: ist enthalten")

if "p" not in tname:
    print("p: ist nicht enthalten")

```

**Listing 4.11** Datei `text_operator.py`

Die Ausgabe lautet:

```

Teil 1, Teil 2
***-oooo--oooo--oooo-***
Text: Robinson Crusoe
b: ist enthalten
p: ist nicht enthalten

```

Zur Erläuterung:

- ▶ Die Zeichenkette `tgesamt` wird mithilfe des Verkettungsoperators `+` aus drei Teilen zusammengesetzt: den beiden Zeichenketten `t1` und `t2` und dem Text mit Komma und Leerzeichen. **Verkettung**
- ▶ Die Zeichenkette `tlinie` wird mithilfe des Verkettungsoperators `+` und des Vervielfachungsoperators `*` zusammengesetzt. Dabei wird der Ausdruck `"-oooo-` dreimal hintereinander in `tlinie` gespeichert. **Vervielfachung**
- ▶ Mithilfe des Operators `in` wird festgestellt, ob das Element `b` in der Sequenz enthalten ist. **Operator in**
- ▶ Der logische Operator `not` dient (zusammen mit `in`) zur Feststellung, ob das Element `p` nicht enthalten ist. **Operator not**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 4.2.3 Operationen

Teilbereiche von Sequenzen werden als *Slices* bezeichnet. Am Beispiel von Strings soll der Einsatz von Slices verdeutlicht werden. Auf die hier beschriebene Weise sind sie auch auf andere Sequenzen anwendbar.

**Teilbereiche von Sequenzen**

Als Beispiel für eine Sequenz wird wiederum die Zeichenkette `Robinson Crusoe` in der Variablen `tname` gespeichert. Tabelle 4.1 stellt die einzelnen Elemente von `tname` mit dem zugehörigen Index dar. Die Nummerierung beginnt bei 0; alternativ können Sie auch eine negative Nummerierung nutzen, die mit 1 endet (siehe unterste Zeile der Tabelle).

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Element	R	o	b	i	n	s	o	n		C	r	u	s	o	e
negativer Index	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Tabelle 4.1 Sequenz mit Index

- Index** Ein Slice wird durch die Angabe eines Bereichs in eckigen Klammern ([ ]) hinter der sequentiellen Variablen erzeugt. Er beginnt mit einem Startindex, gefolgt von einem Doppelpunkt und einem Endindex. Ein Slice, der nur aus einem einzelnen Zeichen besteht, wird durch die Eingabe eines einzelnen Index erzeugt.
- len()** Die eingebaute Funktion `len()` ermittelt die Anzahl der Elemente einer Sequenz. Im Fall eines Strings spricht man hierbei auch von der Länge der Zeichenkette.

```
# Beispiel-Sequenz, hier Zeichenkette
tname = "Robinson Crusoe"
print("Text:", tname)
# Anzahl der Elemente
lg = len(tname)
print("Anzahl Elemente:", lg)
# Teilebereiche, Elemente
ts = tname[5:8]
print("[5:8]:", ts)
ts = tname[:8]
print("[::8]:", ts)
ts = tname[9:]
print("[9:]:", ts)
ts = tname[9:-3]
print("[9:-3]:", ts)

# Elemente einzeln
for zeichen in tname[5:8]:
    print(zeichen)
```

Listing 4.12 Datei text\_operation.py

Es wird die folgende Ausgabe erzeugt:

```
Text: Robinson Crusoe
Anzahl Elemente: 15
[5:8]: son
[::8]: Robinson
[9:]: Crusoe
[9]: C
```

```
[9:-3]: Cru
s
o
n
```

Zur Erläuterung:

- ▶ Die Länge der Sequenz, also die Anzahl der Zeichen in der Zeichenkette, wird mithilfe der Funktion `len()` ermittelt.
- ▶ Slice [5:8]: Der Bereich erstreckt sich von dem Element, das durch den Startindex gekennzeichnet wird, bis *vor* das Element, das durch den Endindex gekennzeichnet wird (Ergebnis: `son`). Slice von ... bis
- ▶ Slice [:8]: Wenn der Startindex weggelassen wird, beginnt der Bereich bei 0 (Ergebnis: `Robinson`). Slice bis
- ▶ Slice [9:]: Wenn der Endindex weggelassen wird, endet der Bereich am Ende der Zeichenkette (Ergebnis: `Crusoe`). Slice von
- ▶ Slice [9]: Wird nur ein Index angegeben, so besteht der Bereich nur aus einem einzelnen Element (Ergebnis: `C`). Element
- ▶ Slice [9:-3]: Wird ein Index mit einer negativen Zahl angegeben, so wird vom Ende aus gemessen, beginnend bei 1 (Ergebnis: `Cru`).
- ▶ Eine `for`-Schleife kann genutzt werden, um eine ganze Zeichenkette oder einen Teil der Zeichenkette Zeichen für Zeichen durchzugehen. for

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

Strings sind nicht veränderbar. Es können keine einzelnen Zeichen oder Bereiche aus Strings durch andere Zeichen oder Slices ersetzt werden.

**Unveränderbar**

Ein Beispiel:

```
tname = "Robinson Crusoe"
```

```
try:
    tname[3] = "e"
except:
    print("Fehler")

try:
    tname[3:5] = "el"
except:
    print("Fehler")
```

**Listing 4.13** Datei `text_unveraenderbar.py`

Die Ausgabe lautet:

```
Fehler
Fehler
```

Zur Erläuterung:

- ▶ Die einzige Möglichkeit zur Veränderung eines Strings ist die Erzeugung eines neuen Objekts.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 4.2.4 Funktionen

Neben der eingebauten Funktion `len()` gibt es für Objekte des Datentyps `str` eine Reihe von nützlichen Funktionen zur Bearbeitung und Analyse von Zeichenketten.

An folgendem Beispiel sollen einige Funktionen zum Suchen von Teilstücken verdeutlicht werden:

```
# Beispiel
test = "Das ist ein Beispielsatz"
print("Text:", test)

# Anzahl Suchtexte
such = "ei"
anz = test.count(such)
print("count: Der String", such, "kommt", anz, "mal vor")

# Erste Position des Suchtextes
anfpos = test.find(such)
print("find 1: Zum ersten Mal an Position", anfpos)
# Weitere Position des Suchtextes
nextpos = test.find(such, anfpos+1)
print("find 2: Ein weiteres Mal an Position", nextpos)
# Letzte Position des Suchtextes
endpos = test.rfind(such)
print("rfind: Zum letzten Mal an Position", endpos)

# Suchtext nicht gefunden
such = "am"
pos = test.find(such)
if pos==-1:
    print("find 3:", such, "wurde nicht gefunden")
else:
    print("find 3:", such, "an Position", pos, "gefunden")
```

```
# Ersetzen von Text
test = test.replace("ist", "war")
print("replace:", test)
```

**Listing 4.14** Datei text\_suchen.py

Folgende Ausgabe wird erzeugt:

```
Text: Das ist ein Beispielsatz
count: Der String ei kommt 2 mal vor
find 1: Zum ersten Mal an Position 8
find 2: Ein weiteres Mal an Position 13
rfind: Zum letzten Mal an Position 13
find 3: am wurde nicht gefunden
replace: Das war ein Beispielsatz
```

Zur Erläuterung:

- ▶ Die Funktion `count()` ergibt die Anzahl der Vorkommen eines Suchtextes (hier in der Variablen `such`) innerhalb des analysierten Textes. `count()`
- ▶ Die Funktion `find()` ergibt die Position, an der ein Suchtext innerhalb eines analysierten Texts vorkommt. Zur Erinnerung: Das erste Element einer Sequenz hat die Position 0. `find()`
- ▶ Bei der Funktion `find()` können Sie optional einen zweiten Parameter angeben. Dieser Parameter bestimmt die Position, ab der gesucht wird. Im vorliegenden Fall ist dies die Position des Zeichens hinter der ersten Fundstelle. Diese Technik wird häufig verwendet, um mithilfe einer Schleife alle Vorkommen eines Suchtextes zu finden. Startpunkt für Suche
- ▶ Die Funktion `rfind()` ergibt die Position des letzten Vorkommens des Suchtextes innerhalb des analysierten Textes. `rfind()`
- ▶ Falls der gesuchte Text nicht vorkommt, liefern `find()` bzw. `rfind()` den Wert `-1` zurück.
- ▶ Die Funktion `replace()` ersetzt einen gesuchten Teiltext durch einen anderen und liefert den geänderten Text zurück. `replace()`

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

Das folgende Beispiel zeigt hauptsächlich Funktionen zum Zerlegen von Texten in Teilstexte:

```
# Beispiel
test = "Das ist ein Beispielsatz"
print("Text:", test)
```

```

# Beginn, Ende
if test.startswith("Das"):
    print("Text beginnt mit Das")
if not test.endswith("Das"):
    print("Text endet nicht mit Das")

# Zerlegung
teile = test.partition("ei")
print("vor der 1. Teilung:", teile[0])
print("hinter der 1. Teilung:", teile[2])

teile = test.rpartition("ei")
print("vor der 2. Teilung:", teile[0])
print("hinter der 2. Teilung:", teile[2])

# Zerlegung in Liste
wliste = test.split()
for i in range(0, 3):
    print("Element:", i, wliste[i])

```

**Listing 4.15** Datei `text_zerlegen.py`

Die Ausgabe lautet:

```

Text: Das ist ein Beispielsatz
Text beginnt mit Das
Text endet nicht mit Das
vor der 1. Teilung: Das ist
hinter der 1. Teilung: n Beispielsatz
vor der 2. Teilung: Das ist ein B
hinter der 2. Teilung: spielsatz
Element: 0 Das
Element: 1 ist
Element: 2 ein

```

Zur Erläuterung:

#### `startswith()`, `endswith()`

- ▶ Mithilfe der Funktionen `startswith()` und `endswith()` untersuchen Sie, ob eine Zeichenkette mit einem bestimmten Text beginnt oder endet. Beide Funktionen liefern einen Wahrheitswert (`True` oder `False`), daher kann der Rückgabewert direkt als Bedingung genutzt werden.

#### `partition()`, `rpartition()`

- ▶ Die Funktionen `partition()` und `rpartition()` zerlegen eine Zeichenkette in drei Teile anhand eines Teilungstexts. Diese drei Teile werden in einem Tupel geliefert.
  - Das erste Element des Tupels (Element 0) enthält den Text bis zum Teilungstext, das dritte Element (Element 2) enthält den Text hinter dem Teilungstext.

- Die Funktion `partition()` sucht den Teilungstext ausgehend vom Beginn der Zeichenkette, die Funktion `rpartition()` sucht ihn ausgehend vom Ende der Zeichenkette.
  - Falls der Teilungstext nicht gefunden wird, steht die gesamte Zeichenkette im ersten Element des Tupels; das dritte Element ist leer.
  - Weitere Informationen zu den Tupeln erhalten Sie in [Abschnitt 4.4, »Tupel«](#).
- Die Funktion `split()` zerlegt einen Text in einzelne Teile, die in einer Liste gespeichert werden. Das Leerzeichen wird dabei als Trennzeichen angesehen. Zur Verdeutlichung werden im vorliegenden Beispiel die drei ersten Elemente der Liste, zusammen mit der laufenden Nummer innerhalb der Liste, ausgegeben (für weitere Informationen zu Listen siehe [Abschnitt 4.3, »Listen«](#)).
- Falls bei der Funktion `split()` ein anderes Trennzeichen verwendet werden soll, wie z. B. das Semikolon oder das Rautezeichen, dann muss dieses Zeichen als Parameter an die Funktion übergeben werden, z. B.: `split(";"")`.

`split()`

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 4.2.5 Umwandlung von einer Zeichenkette in eine Zahl

Enthält eine Zeichenkette eine Zahl, etwa eine vom Benutzer eingegebene Zeichenkette oder einen Kommandozeilenparameter (siehe [Abschnitt 5.11, »Parameter der Kommandozeile«](#)), so kann diese Zeichenkette konvertiert werden. Dazu dienen die bereits bekannten Funktionen `int()` und `float()` zur Umwandlung in eine ganze Zahl oder in eine Zahl mit Nachkommastellen. Anschließend kann mit dieser Zahl gerechnet werden.

Das folgende Beispiel soll den Unterschied in der Behandlung von Zahlen und Strings verdeutlichen:

```
# Erste Zeichenkette
x = "15.3"
```

```
ergebnis = x * 2
print(ergebnis)
```

```
x = float(x)
ergebnis = x * 2
print(ergebnis)
```

```
# Zweite Zeichenkette
x = "17"
```

```
ergebnis = x * 2
print(ergebnis)
```

```
x = int(x)
ergebnis = x * 2
print(ergebnis)
```

**Listing 4.16** Datei text\_in\_zahl.py

Die Ausgabe des Programms ist:

```
15.315.3
30.6
1717
34
```

Zur Erläuterung:

- `float()` ▶ In der ersten Zeichenkette steht 15.3. Wird diese Zeichenkette mit 2 »multipliziert«, so ergibt sich die Zeichenkette 15.315.3. Wird die Zeichenkette mithilfe der Funktion `float()` hingegen in eine Zahl mit Nachkommastellen verwandelt, so kann mit ihr gerechnet werden. Eine Multiplikation mit 2 ergibt mathematisch korrekt die Ausgabe 30.6.
- `int()` ▶ Ähnlich sieht es bei der Umwandlung einer ganzen Zahl aus. In der zweiten Zeichenkette steht 17. Wenn diese Zeichenkette mit 2 »multipliziert« wird, ergibt sich die Zeichenkette 1717. Wird die Zeichenkette mithilfe der Funktion `int()` in eine ganze Zahl verwandelt, so kann mit ihr gerechnet werden. Eine Multiplikation mit 2 ergibt in diesem Fall mathematisch korrekt 34.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

Versucht man, eine Zeichenkette, die keine gültige Zahl enthält, in eine Zahl umzuwandeln, so tritt eine Ausnahme auf. Daher sollte die Umwandlung von Benutzereingaben oder Kommandozeilenparametern in eine Ausnahmebehandlung eingebettet werden:

```
# Fehler abfangen
x = "15.3p"

try:
    x = float(x)
    print(x*2)
except:
    print("Zeichenkette konnte nicht umgewandelt werden")
```

**Listing 4.17** Datei text\_keine\_zahl.py

Die Ausgabe des Programms lautet:

**Zeichenkette konnte nicht umgewandelt werden**

Zur Erläuterung:

- In der Zeichenkette steht ein nicht gültiges Zeichen (hier das p). Bei der Umwandlung tritt eine Ausnahme auf, und es erscheint eine Fehlermeldung.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 4.2.6 Umwandlung von einer Zahl in eine Zeichenkette

Muss eine Zahl in eine Zeichenkette umgewandelt werden, etwa zur Ausgabe der Zahl in eine Datei (siehe [Abschnitt 8.3.1](#), »Sequentielles Schreiben«), so können Sie die Funktion `str()` verwenden. Ein Beispiel:

```
a = 23
b = 7.5
c = a + b

# 1. Ausgabe
print(a, "+", b, "=", c)

# 2. Ausgabe
print(str(a) + "+" + str(b) + "=" + str(c))
```

**Listing 4.18** Datei `text_von_zahl.py`

Die Ausgabe lautet:

```
23 + 7.5 = 30.5
23+7.5=30.5
```

Zur Erläuterung:

- Die erste Ausgabe erfolgt wie gewohnt. Sie setzt sich aus den einzelnen Variablen und den verbindenden Texten zusammen. Nach jedem Teil der Ausgabe wird automatisch ein Leerzeichen eingefügt.
- Für die zweite Ausgabe werden die Zahlen zunächst mithilfe der Funktion `str()` in Zeichenketten umgewandelt. Abschließend werden die verschiedenen Zeichenketten mit dem Verkettungsoperator `+` verbunden. In der Ausgabe kommen keine Leerzeichen mehr vor.

Eine weitere Möglichkeit zur Gestaltung von Ausgaben zeige ich Ihnen in [Abschnitt 5.2.2](#), »Formatierte Ausgabe mit `format()`«.

**Operator +**

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 4.2.7 Datentyp »bytes«

**Erzeugen** Objekte des Datentyps `bytes` bestehen aus Zeichen, deren Zeichencode im Bereich von 0 bis 255 liegt. Jedes Zeichen kann mithilfe eines Bytes gespeichert werden. Sie können `bytes`-Objekte mithilfe von Byte-Literalen erzeugen oder mithilfe der eingebauten Funktion `bytes()`. Im Unterschied zu `bytes`-Objekten werden die üblichen Zeichenketten, also `str`-Objekte, aus Unicode-Zeichen gebildet.

**Byte-Literal** Byte-Literale beginnen mit einem »`b`« oder einem »`B`«. Dies ist bei Eingabe oder Zuweisung zu beachten. Bei der Ausgabe wird ein `b` vorangestellt.

Es folgt ein Beispielprogramm:

```
# Datentyp str
st = "Hallo"
print(st, type(st))

# Datentyp bytes
by = b'Hello'
print(by, type(by))

# Umwandlung von str in bytes
by = bytes("Hello", "UTF-8")
print(by, type(by))

# Umwandlung von bytes in str
by = b'Hello'
st = by.decode()
print(st, type(st))
```

**Listing 4.19** Datei `bytes.py`

Das Programm erzeugt die folgende Ausgabe:

```
Hallo <class 'str'>
b'Hello' <class 'bytes'>
b'Hello' <class 'bytes'>
Hello <class 'str'>
```

Zur Erläuterung:

- ▶ Zunächst werden ein `str`-Objekt und ein `bytes`-Objekt per Zuweisung gebildet und ausgegeben. Beachten Sie das vorangestellte `b` beim Byte-Literal.

- Zur Umwandlung eines str-Objekts in ein bytes-Objekt wird die eingebaute Funktion bytes() genutzt. Dabei wird die Codierung des str-Objekts angegeben, hier UTF-8.
- Die Methode decode() wird zur Umwandlung eines bytes-Objekts in ein str-Objekt verwendet werden.

### Unterschiede in Python 2

In Python 2 gibt es den Datentyp bytes nicht. Zeichenketten sind vom Typ str.

## 4.3 Listen

Eine Liste ist eine Sequenz von Objekten in ihrer allgemeinsten Form. Sie kann Elemente unterschiedlichen Objekttyps enthalten. Eine Liste bietet vielfältige Möglichkeiten, unter anderem die Funktionalität von ein- und mehrdimensionalen Feldern (Arrays), wie man sie aus anderen Programmiersprachen kennt.

### 4.3.1 Eigenschaften

Eine Liste ist im Gegensatz zu einem String veränderbar. Von diesem Unterschied abgesehen, ist ein String, vereinfacht gesagt, nur eine spezialisierte Form einer Liste zur Speicherung von einzelnen Zeichen. Einige Beispiele für Listen:

Veränderbar

```
# Liste von Zahlen
z = [3, 6, 12.5, -8, 5.5]
print(z)          # gesamte Liste
print(z[0])       # ein Element
print(z[0:3])     # Slice

# Liste von Zeichenketten
s = ["Hamburg", "Augsburg", "Berlin"]
print(s)

# Anzahl Elemente
print("Anzahl:", len(s))
```

**Listing 4.20** Datei liste\_eigenschaft.py

Es wird die Ausgabe erzeugt:

```
[3, 6, 12.5, -8, 5.5]
3
[3, 6, 12.5]
```

```
[‘Hamburg’, ‘Augsburg’, ‘Berlin’]
```

Anzahl: 3

Zur Erläuterung:

- Eckige Klammern** ► Listen werden innerhalb von eckigen Klammern angegeben.
- Kommata** ► Innerhalb dieser Klammern listen Sie die einzelnen Elemente durch Komma trennt auf.
- Die Variable `z` enthält eine Liste von Zahlen mit und ohne Nachkommastellen.
- Index** ► Wie bei Strings ermitteln Sie ein einzelnes Element einer Liste durch Angabe eines Index (hier: `z[0]`).
- Slice** ► Einen Teilbereich einer Liste ermitteln Sie mithilfe eines Slices (hier: `z[0:3]`).
- Die Variable `s` enthält eine Liste von Zeichenketten.
- len()** ► Die Länge einer Sequenz, also auch einer Liste, ermitteln Sie mit der Funktion `len()`.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

- Mehrdimensionale Listen** Eine Liste kann Elemente unterschiedlicher Objekttypen enthalten. Diese Elemente können wiederum Listen sein. Auf diese Weise werden mehrdimensionale Listen erzeugt. Ein Beispiel:

```
# mehrdimensionale Liste, unterschiedliche Objekte
x = [[“Paris”, “Fr”, 3500000], [“Rom”, “It”, 4200000]]
print(x)

# Teilliste
print(x[0])

# einzelne Elemente
print(x[0][0], “hat”, x[0][2], “Einwohner”)
print(x[1][0], “hat”, x[1][2], “Einwohner”)

# Teile von Elementen
print(x[0][1][:1])
```

**Listing 4.21** Datei `liste_mehrdimensional.py`

Die Ausgabe lautet:

```
[['Paris', 'Fr', 3500000], ['Rom', 'It', 4200000]]
```

```
['Paris', 'Fr', 3500000]
```

```
Paris hat 3500000 Einwohner
```

```
Rom hat 4200000 Einwohner
```

```
F
```

Zur Erläuterung:

- ▶ Die Variable `x` enthält zwei Listen. Innerhalb jeder Teilliste (oder eingebetteten Liste) sind zwei Zeichenketten und eine Zahl gespeichert.
- ▶ Eingebettete Listen ermitteln Sie durch Angabe eines einzelnen Index oder eines Slices (hier: `x[0]`).
- ▶ Einzelne Elemente von eingebetteten Listen sprechen Sie durch Angabe mehrerer Indizes oder Slices an (hier z. B.: `x[0][2]`). Die erste Angabe in eckigen Klammern kennzeichnet hier die eingebettete Liste mit dem Index 0, die zweite Angabe in eckigen Klammern kennzeichnet das Element mit dem Index 2 innerhalb dieser eingebetteten Liste.
- ▶ Einzelne Elemente der Liste sind wiederum Sequenzen, falls es sich um Zeichenketten handelt. Daher können Sie das erste Zeichen des Elements `x[0][1]` mithilfe von `x[0][1][:1]` ermitteln.

**Eingebettete Liste**

**Mehrere Indizes**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 4.3.2 Operatoren

Die Operatoren `+` und `*` können Sie bei Sequenzen, also auch bei Listen, zur Verkettung oder Vervielfachung einsetzen. Außerdem werden Listen, zusammen mit dem Operator `in`, häufig zur Erstellung einer `for`-Schleife genutzt. Ein gemeinsames Beispiel für die beiden genannten Fälle:

```
# zwei Listen
fr = ["Paris", "Lyon", "Marseille"]
it = ["Rom", "Pisa"]

# Listen zusammensetzen
stadtliste = fr + it * 2
print(stadtliste)

# Liste teilweise durchlaufen
for stadt in stadtliste[3:6]:
    print(stadt)
```

**Operatoren `+`, `*`, `in`**

**Listing 4.22** Datei `liste_operator.py`

Es wird die folgende Ausgabe erzeugt:

```
[‘Paris’, ‘Lyon’, ‘Marseille’, ‘Rom’, ‘Pisa’, ‘Rom’, ‘Pisa’]
Rom
Pisa
Rom
```

Zur Erläuterung:

- ▶ In den beiden Listen `fr` und `it` werden jeweils einige Zeichenketten gespeichert.
- ▶ Die Liste `stadtliste` ist eine neue Liste. Sie enthält die Elemente der Liste `fr` und zweimal nacheinander die Elemente der Liste `it`.
- ▶ Die Liste `stadtliste` wird als vollständige Liste ausgegeben.

#### for-Schleife

- ▶ Mithilfe einer `for`-Schleife und eines Slices wird ein Teil der Liste Element für Element ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 4.3.3 Funktionen und Operationen

**Veränderlich** Listen können, im Gegensatz zu Strings, verändert werden. Sie können also nicht nur einzelne Elemente oder Teilbereiche auswählen, sondern Sie können außerdem

- Liste ändern**
- ▶ zusätzliche Elemente oder Teilbereiche am Ende oder am Anfang hinzufügen,
  - ▶ vorhandene Elemente oder Teilbereiche verändern,
  - ▶ Elemente oder Teilbereiche innerhalb der Liste einfügen und
  - ▶ Elemente oder Teilbereiche löschen.

Sie sollten allerdings beachten, dass Sie nicht aus Versehen ein einzelnes Element durch einen Teilbereich ersetzen. Dadurch würde statt des Austauschs eines Elements eine ganze Liste als eingebettete Liste erzeugt.

- del** Im Folgenden werden zunächst einige Listenoperationen mit Elementen, Teilbereichen und der Anweisung `del` zum Löschen von Elementen aufgeführt:

```
# Originalliste
fr = ["Paris", "Lyon", "Marseille", "Bordeaux"]
print("Original:")
print(fr)
```

```
# Ersetzen eines Elementes durch ein Element
fr[2] = "Lens"
```

```

print("Element ersetzt:")
print(fr)

# Ersetzen eines Teilstücks durch eine Liste
fr[1:3] = ["Nancy", "Metz", "Gap"]
print("Teil ersetzt:")
print(fr)
# Entfernen eines Teilstücks
del fr[3:]
print("Teil entnommen:")
print(fr)

# Ersetzen eines Elementes durch eine Liste
fr[0] = ["Paris-Nord", "Paris-Sud"]
print("Element durch Liste ersetzt:")
print(fr)

```

**Listing 4.23** Datei liste\_element.py

Die Ausgabe der Liste in den verschiedenen Zuständen:

Original:

`['Paris', 'Lyon', 'Marseille', 'Bordeaux']`

Element ersetzt:

`['Paris', 'Lyon', 'Lens', 'Bordeaux']`

Teil ersetzt:

`['Paris', 'Nancy', 'Metz', 'Gap', 'Bordeaux']`

Teil entnommen:

`['Paris', 'Nancy', 'Metz']`

Element durch Liste ersetzt:

`[['Paris-Nord', 'Paris-Sud'], 'Nancy', 'Metz']`

Zur Erläuterung:

- ▶ Die Originalliste, bestehend aus vier Zeichenketten, wird erstellt.
- ▶ Ein einzelnes Element wird durch eine andere Zeichenkette ersetzt. Ersetzen
- ▶ Ein Teilstück wird durch eine Liste ersetzt.
- ▶ Ein Teilstück wird mithilfe von `del` aus der Liste gelöscht. Löschen
- ▶ Ein Element wird durch eine Liste ersetzt; dadurch wird eine eingebettete Liste erzeugt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

Es gibt eine Reihe von weiteren Funktionen zur Analyse und Bearbeitung von Listen. Einige davon werden im folgenden Programm verdeutlicht:

```

# Originalliste
fr = ["Paris", "Lyon", "Marseille"]
print("Original:")
print(fr)
# Einsetzen eines Elements
fr.insert(0, "Nantes")
print("Nach Einsetzen:")
print(fr)

# Sortieren der Elemente
fr.sort()
print("Nach Sortieren:")
print(fr)

# Umdrehen der Liste
fr.reverse()
print("Nach Umdrehen:")
print(fr)

# Entfernen eines Elements
fr.remove("Nantes")
print("Nach Entfernen:")
print(fr)

# Ein Element am Ende hinzu
fr.append("Paris")
print("Ein Element hinzu:")
print(fr)

# Anzahl bestimmter Elemente
print("Anzahl Elemente Paris:", fr.count("Paris"))

# Suchen bestimmter Elemente
print("Erste Position Paris:", fr.index("Paris"))

```

**Listing 4.24** Datei `liste_aendern.py`

Die Ausgabe der Liste in den verschiedenen Zuständen:

```

Original:
['Paris', 'Lyon', 'Marseille']
Nach Einsetzen:
['Nantes', 'Paris', 'Lyon', 'Marseille']
Nach Sortieren:
['Lyon', 'Marseille', 'Nantes', 'Paris']
Nach Umdrehen:
['Paris', 'Nantes', 'Marseille', 'Lyon']
Nach Entfernen:
['Paris', 'Marseille', 'Lyon']

```

**Ein Element hinzufügen:**

`['Paris', 'Marseille', 'Lyon', 'Paris']`

Anzahl Elemente Paris: 2

Erste Position Paris: 0

Zur Erläuterung:

- ▶ Die Originalliste, bestehend aus drei Zeichenketten, wird erstellt.
- ▶ Ein Element wird mit der Funktion `insert()` an der Position 0 eingefügt, also zu Beginn der Liste. `insert()`
- ▶ Die Liste wird mit der Funktion `sort()` intern sortiert. Falls es sich um eine Liste von Zeichenketten handelt, wird alphabetisch sortiert. Eine Liste von Zahlen wird nach Größe sortiert. Bei anderen Listenelementen oder bei gemischten Listen ist der Einsatz der Funktion `sort()` nur bedingt sinnvoll. `sort()`
- ▶ Die Liste wird mit der Funktion `reverse()` intern umgedreht. `reverse()`
- ▶ Ein bestimmtes Element (hier: `Nantes`) wird innerhalb der Liste gesucht. Falls es vorhanden ist, wird das erste Vorkommen dieses Elements mit der Funktion `remove()` gelöscht. Falls es nicht vorhanden ist, wird eine Ausnahme ausgelöst. `remove()`
- ▶ Ein Element wird am Ende der Liste mit `append()` angefügt. `append()`
- ▶ Die Anzahl der Vorkommen eines bestimmten Elements (hier: `Paris`) wird mit der Funktion `count()` ermittelt. `count()`
- ▶ Die Position des ersten Vorkommens eines bestimmten Elements (hier: `Paris`) wird mit der Funktion `index()` ermittelt. Ist das Element nicht vorhanden, wird eine Ausnahme ausgelöst. `index()`

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 4.4 Tupel

In diesem Abschnitt werden Tupel und ihre besonderen Eigenschaften sowie Operationen mit Tupeln erläutert.

### 4.4.1 Eigenschaften

Ein Tupel unterscheidet sich von einer Liste im Wesentlichen durch eine einzige Eigenschaft: Ein Tupel kann nicht verändert werden. Ansonsten gelten die gleichen Regeln, und es können die gleichen Operationen und Funktionen auf Tupel wie auf Listen angewendet werden, sofern sie keine Veränderung des Tupels hervorrufen.

Unveränderlich

### 4.4.2 Operationen

Einige Beispiele und Besonderheiten:

```
# Tupel mit und ohne Klammer
z = (3, 6, -8, 5.5)
print("Tupel 1:", z)

z = 6, 8, -3
print("Tupel 2:", z)
# mehrdimensionales Tupel, unterschiedliche Objekte
x = (("Paris", "Fr", 3500000), ["Rom", "It", 4200000])
print("mehrdim. Tupel:")
print(x)

# Ersetzen
try:
    x[0][0] = "Lyon" # nicht erlaubt, weil Tupel
except:
    print("Fehler")
x[1][0] = "Pisa" # erlaubt, weil Liste
print("Listenelement ersetzt:", x[1])

# Tupel bei for-Schleife
for i in 4, 5, 12:
    print("i:", i)

# Zuweisung mit Tupel
x,y = 2,18
print("x:", x, "y:", y)
```

**Listing 4.25** Datei `tupel_operation.py`

Es wird die Ausgabe erzeugt:

```
Tupel 1: (3, 6, -8, 5.5)
Tupel 2: (6, 8, -3)
mehrdim. Tupel:
([('Paris', 'Fr', 3500000), ['Rom', 'It', 4200000]])
Fehler
Listenelement ersetzt: ['Pisa', 'It', 4200000]
i: 4
i: 5
i: 12
x: 2 y: 18
```

Zur Erläuterung:

#### Runde Klammern

- ▶ Tupel können mit runden Klammern (statt eckiger Klammern bei Listen) oder ganz ohne Klammern erzeugt werden. Sie können gleichzeitig Zahlen, Zeichenketten und andere Objekte enthalten.

- ▶ Durch Einbettung können Sie mehrdimensionale Tupel erzeugen. Hier ist dies das Tupel `x`, bestehend wiederum aus einem Tupel und einer Liste.
- ▶ Die versuchte Veränderung des inneren Tupels erzeugt eine Ausnahme. Erlaubt ist dagegen die Veränderung der inneren Liste, die in einem Tupel eingebettet ist (hier für `x[1][0]`).
- ▶ Die ersten `for`-Schleifen in diesem Buch wurden bereits mithilfe von Tupeln geschrieben, ohne dass dieser Begriff gesondert erwähnt wurde.
- ▶ Mithilfe eines Tupels können Sie mehrere Werte gleichzeitig zuweisen. Im vorliegenden Beispiel werden die Werte 2 und 18 den Einzelvariablen `x` bzw. `y` zugewiesen. Mehr zu diesem Thema erfahren Sie im nächsten Abschnitt.

Mehrdimensionale  
Tupel

Mehrfa-  
che  
Zuweisung

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 4.4.3 Tupel entpacken

Innerhalb eines Tupels sind mehrere, unveränderliche Werte gespeichert. Mithilfe eines Tupels kann eine mehrfache Zuweisung erfolgen. Dabei werden in einer Anweisung gleichzeitig mehreren Variablen Werte zugewiesen.

Dieses Vorgehen kann Ihnen Schreibarbeit ersparen. Allerdings müssen Sie dabei einige Besonderheiten beachten, die das folgende Programm verdeutlicht.

Schreibabkürzung

```
# 1: Mehrfache Zuweisung
x, y, z = 3, 5.2, "hallo"
print("Mehrf. Zuweisung:", x, y, z)

# 2: Auswirkungen erst danach
a = 12
b = 15
c = 22
a, b, c = c, a, a+b
print("Auswirkung:", a, b, c)

# 3: Verpacken eines Tupels
p = 3, 4
print("Verpackt:", p)

# 4: Entpacken eines Tupels
m, n = p
print("Entpackt: m:", m, "n:", n)

# 5: Falsche Zuweisung eines Tupels
try:
```

```

s, t = 3, 4, 12
print(s, t)
except:
    print("Fehler")

# 6: Rest in Liste
print()
x, *y, z = 3, 5.2, "hallo", 7.3, 2.9
print(x)
print(y)
print(z)

# kein Rest, Liste leer
print()
x, *y, z = 3, 5.2
print(x)
print(y)
print(z)

```

**Listing 4.26** Datei tupel\_entpacken.py

Die Ausgabe sieht wie folgt aus:

**Mehrf. Zuweisung:** 3 5.2 hallo

**Auswirkung:** 22 12 27

**Verpackt:** (3, 4)

**Entpackt:** m: 3 n: 4

**Fehler**

```

3
[5.2, 'hallo', 7.3]
2.9
3
[]
5.2

```

Zur Erläuterung der mehrfachen Zuweisung (1):

**Mehrfa  
che  
Zuweisung**

- ▶ Die Variable `x` bekommt den Wert `3`, die Variable `y` den Wert `5.2` und die Variable `z` den Wert `hallo` zugewiesen.
- ▶ Die folgenden Anweisungen bewirken dasselbe, beanspruchen aber drei Programmzeilen statt einer.

```

x = 3
y = 5.2
z = "hallo"

```

Zur Erläuterung der Auswirkungen (2):

- ▶ Sie erkennen, dass die Änderung eines Variablenwerts keine Auswirkungen innerhalb der gleichen Mehrfachzuweisung hat:

- Die Variable `a` erhält den alten (und neuen) Wert von `c` (= 22).
- Die Variable `b` erhält den alten Wert von `a` (= 12).
- Die Variable `c` erhält den alten Wert von `a`, erhöht um den alten Wert von `b` (= 27).

- ▶ Die Änderung der Variablen wirkt sich erst in der nächsten Anweisung aus.
- ▶ Die folgenden Anweisungen hätten zu ganz anderen Ergebnissen geführt, sie sind nicht (!) gleichzusetzen mit der Anweisung: `a, b, c = c, a, a+b`:

```
a = c
b = a
c = a+b
```

Auswirkung später

Zur Erläuterung der Verpackung (3):

- ▶ Steht auf der linken Seite der Zuweisung nur eine Variable (im Beispiel: `p`), während auf der rechten Seite mehrere Werte oder Ausdrücke stehen (im Beispiel: `3` und `4`), so handelt es sich um die Erzeugung (Verpackung) eines Tupels.

Verpacken

Zur Erläuterung der Entpackung (4):

- ▶ Ein Tupel kann wieder entpackt werden. Dabei sollte die Anzahl der Variablen auf der linken Seite der Zuweisung (im Beispiel: die beiden Variablen `m` und `n`) der Anzahl der Elemente des Tupels (im Beispiel: `p`) auf der rechten Seite entsprechen.

Entpacken

Zur Erläuterung der falschen Zuweisung (5):

- ▶ Es werden mehrere Werte zugewiesen, aber es steht nicht die gleiche Anzahl an Variablen oder nur eine einzelne Variable zur Verfügung. Es resultiert eine Ausnahme, da die Werte weder einem Tupel noch einer einzelnen Variablen eindeutig zugeordnet werden können.

Anzahl passt nicht

Zur Erläuterung der Zuweisung an die gesternnte Variable (6):

- ▶ Falls die Anzahl der Tupelwerte nicht mit der Anzahl der Variablen übereinstimmt, können Sie (seit Python 3) einer der Variablen einen Stern voranstellen. In dieser gestertennten (englisch: *starred*) Variablen wird der Rest des Tupels gespeichert, der nicht einer einzelnen Variablen zugewiesen werden konnte. Im Programmbeispiel werden der erste und der letzte Wert einer einzelnen Variablen zugewiesen. Die mittleren Werte landen als Liste in der mittleren Variablen.

Variable mit \*

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Gesternnte Variable gibt es erst seit Python 3. In Python 2 muss die Anzahl der Tupelwerte mit der Anzahl der Variablen übereinstimmen. Daher müssen Sie Teil 6 des Programms weglassen, ansonsten käme es zu einem Programmabbruch.

## 4.5 Dictionarys

**Wörterbuch** Ein Dictionary ist mit einem Wörterbuch zu vergleichen. In einem Wörterbuch finden Sie unter einem Schlüsselbegriff die zugeordnete Information. So steht etwa in einem englisch-deutschen Wörterbuch unter dem Eintrag *house* der zugeordnete deutsche Begriff *Haus*.

### 4.5.1 Eigenschaften

**Schlüssel, Wert** In Python stellen Dictionarys veränderliche Objekte dar und bestehen aus mehreren Paaren. Jedes Paar besteht aus einem eindeutigen Schlüssel und einem zugeordneten Wert. Über den Schlüssel greifen Sie auf den Wert zu. Als Schlüssel werden meistens Strings verwendet, es können aber auch andere unveränderliche Objekte (Zahlen, Tupel) benutzt werden. Die Schlüssel sind ungeordnet, daher ist auch die Reihenfolge bei der Ausgabe eines gesamten Dictionarys nicht festgelegt.

Im folgenden Beispiel sollen mehrere Personen und ihre Altersangaben in einem Dictionary erfasst und bearbeitet werden. Der Name der jeweiligen Person dient als Schlüssel. Über den Namen kann auf das Alter der Person (auf den Wert des Schlüssels) zugegriffen werden.

```
# Erzeugung eines Dictionarys
alter = {"Peter":31, "Julia":28, "Werner":35}
print(alter)
# Ersetzen eines Werts
alter["Julia"] = 27
print(alter)
# Ein Element hinzufügen
alter["Moritz"] = 22
print(alter)
# Ausgabe
print("Julia:", alter["Julia"])
```

**Listing 4.27** Datei dictionary\_eigenschaft.py

Folgende Ausgabe wird erzeugt:

```
{'Peter': 31, 'Julia': 28, 'Werner': 35}
{'Peter': 31, 'Julia': 27, 'Werner': 35}
{'Moritz': 22, 'Peter': 31, 'Julia': 27, 'Werner': 35}
Julia: 27
```

Zur Erläuterung:

- Geschweifte Klammern**
- ▶ Es wird das Dictionary `alter` mit drei Informationspaaren erzeugt und ausgegeben. Dictionarys werden mithilfe von geschweiften Klammern (`{}`) erzeugt. Die Paare werden durch Kommata voneinander getrennt, ein Paar wird in der folgenden Form notiert: *Schlüssel:Wert*.

- ▶ Auf ein Element greifen Sie über die Angabe des Schlüssels in eckigen Klammern zu. Dies wird im vorliegenden Beispiel für Zuweisung und Ausgabe genutzt.
- ▶ Elemente sind veränderlich. Hier wird die Altersangabe von "Julia" verändert.
- ▶ Elemente können hinzugefügt werden. Hier wird dem Dictionary das Paar "Moritz":22 hinzugefügt, da kein Element mit dem Schlüssel "Moritz" gefunden wurde.

Eckige Klammern

Veränderlich

Hinzufügen

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 4.5.2 Funktionen

Es gibt eine Reihe von Funktionen zur Bearbeitung von Dictionaries. Einige werden im folgenden Programm verdeutlicht:

```
# Erzeugung
alter = {"Peter":31, "Julia":28, "Werner":35}
print(alter)

# Element enthalten?
if "Julia" in alter:
    print(alter["Julia"])

# Entfernen eines Elementes
del alter["Julia"]

# Element enthalten?
if "Julia" not in alter:
    print("Julia ist nicht enthalten")

# Anzahl Elemente
print("Anzahl: ", len(alter))

# Aktualisierung mit zweitem Dictionary
ualter = {'Moritz': 18, 'Werner': 29}
alter.update(ualter)
print(alter)
```

**Listing 4.28** Datei `dictionary_funktion.py`

Das Programm erzeugt die Ausgabe:

```
{'Peter': 31, 'Julia': 28, 'Werner': 35}
28
```

```
Julia ist nicht enthalten
Anzahl: 2
{'Moritz': 18, 'Peter': 31, 'Werner': 29}
```

Zur Erläuterung:

- del** ► Ein einzelnes Element wird mithilfe der Anweisung `del` aus der Liste gelöscht.
- Operator in** ► Die Existenz eines Elements prüfen Sie mithilfe des Operators `in`.
- len()** ► Die Anzahl der Elemente ermitteln Sie mithilfe der Funktion `len()`.
- update()** ► Ein Dictionary aktualisieren Sie mithilfe der Funktion `update()` mit einem anderen Dictionary. Dabei erhalten vorhandene Elemente gegebenenfalls einen neuen Wert, neue Elemente werden angehängt. Die beiden Dictionaries werden also zusammengeführt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 4.5.3 Views

**Unmittelbare Änderung** Die Funktionen `keys()`, `items()` und `values()` erzeugen sogenannte Views eines Dictionaries. Diese Views verändern sich seit Python 3 unmittelbar, falls sich das zugeordnete Dictionary verändert. Views sind also seit Python 3 dynamisch, vorher waren sie statisch. Ein Beispiel:

```
# Erzeugung
alter = {"Peter":31, "Julia":28, "Werner":35}

# Werte
w = alter.values()
print("Anzahl Werte:", len(w))
for x in w:
    print(x)
if 31 in w:
    print("31 ist enthalten")
alter["Peter"] = 41
if 31 not in w:
    print("31 ist nicht enthalten")
print()

# Keys
k = alter.keys()
print("Anzahl Keys:", len(k))
for x in k:
    print(x)
```

```

if "Werner" in k:
    print("Werner ist enthalten")
del alter["Werner"]
if "Werner" not in k:
    print("Werner ist nicht enthalten")
print()

# Items
i = alter.items()
alter["Franz"] = 35
print("Anzahl Items:", len(i))
for x in i:
    print(x)
if ("Julia", 28) in i:
    print("Julia, 28 ist enthalten")

```

**Listing 4.29** Datei dictionary\_view.py

Folgende Ausgaben werden durch den ersten Teil des Programms erzeugt:

**Anzahl Werte: 3**

31

28

35

31 ist enthalten

31 ist nicht enthalten

Zur Erläuterung der Werte-View:

- ▶ Mithilfe der Funktion `values()` wurde eine (seit Python 3 dynamische) View (w) der Werte des Dictionarys erzeugt.
- ▶ Den Inhalt der View können Sie mithilfe einer `for`-Schleife und des Operators `in` ausgeben. Sie können – wiederum mithilfe des Operators `in` – prüfen, ob ein bestimmter Wert in der View existiert.
- ▶ Der Wert eines Dictionary-Elements wird verändert. Dies hat seit Python 3 auch Auswirkungen auf die zugehörige View. Diese muss also nicht mehr neu erzeugt werden. Der ursprüngliche Wert wird nach der Änderung nicht mehr gefunden.

**Werte-View**

**Unmittelbare Änderung**

Folgende Ausgaben werden durch den zweiten Teil des Programms erzeugt:

**Anzahl Keys: 3**

Peter

Julia

Werner

Werner ist enthalten

Werner ist nicht enthalten

Zur Erläuterung der Schlüssel-View:

**Schlüssel-View**

- ▶ Mithilfe der Funktion `keys()` wurde eine (seit Python 3 dynamische) View (`k`) der Keys des Dictionarys erzeugt.
- ▶ Den Inhalt der Views können Sie mithilfe einer `for`-Schleife und des Operators `in` ausgeben. Sie können – wiederum mithilfe des Operators `in` – prüfen, ob ein bestimmter Key in der View existiert.

**Lösung unmittelbar**

- ▶ Ein Dictionary-Element wird gelöscht. Dies hat seit Python 3 auch Auswirkungen auf die zugehörige View. Das ursprünglich vorhandene Dictionary-Element wird nach dem Löschen nicht mehr gefunden.

Folgende Ausgaben werden durch den dritten und letzten Teil des Programms erzeugt:

```
Anzahl Items: 3
('Franz', 35)
('Peter', 41)
('Julia', 28)
Julia, 28 ist enthalten
```

Zur Erläuterung der Elemente-View:

**Elemente-View**

- ▶ Mithilfe der Funktion `items()` wurde eine (seit Python 3 dynamische) View (`i`) der Elemente des Dictionarys erzeugt.
- ▶ Den Inhalt der View können Sie mithilfe einer `for`-Schleife und `in` ausgeben. Sie können – wiederum mit `in` – prüfen, ob ein bestimmtes Element, also eine Schlüssel-Wert-Kombination, in der View existiert.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Da es sich um statische Views handelt, werden Veränderungen nicht unmittelbar bemerkt. Am Ende des Programms ist der Wert `31` nicht mehr in der Werte-View und der Key `Werner` nicht mehr in der Key-View enthalten. Das Programm liefert daher andere Ausgaben.

#### 4.5.4 Vergleiche

**Alle Elemente gleich**

Dictionarys können miteinander verglichen werden. Mithilfe des Operators `==` stellen Sie fest, ob alle Elemente, also alle Schlüssel-Wert-Kombinationen, übereinstimmen. Allerdings können Sie nicht prüfen, ob ein Dictionary kleiner oder größer als ein anderes Dictionary ist. Ein Beispiel:

```
# Zwei Dictionarys
alter1 = {"Julia":28, "Peter":30}
alter2 = {"Peter":30, "Julia":28}
```

```
# Vergleich
if alter1 == alter2:
    print("Gleich")
try:
    if alter1 < alter2:
        print("1 < 2")
    else
        print("nicht 1 < 2")
except:
    print("Fehler")
```

**Listing 4.30** Datei dictionary\_vergleich.py

Die Ausgabe lautet:

**Gleich**  
**Fehler**

Zur Erläuterung:

- ▶ Die beiden Dictionarys werden in unterschiedlicher Reihenfolge erstellt. Mithilfe von `==` wird festgestellt, dass sie dennoch den gleichen Inhalt haben.
- ▶ Der Vergleich mit `<` (oder `>`) ist nicht möglich; er führt zu einem Fehler.

**Operator `==`**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Ein Vergleich mit `<` oder `>` ist möglich, allerdings hat das Ergebnis keine definierte Aussage.

## 4.6 Mengen, Sets

Mengen (engl.: *sets*) unterscheiden sich von Listen und Tupeln dadurch, dass jedes Element nur einmal existiert. Außerdem sind Mengen ungeordnet, daher ist auch die Reihenfolge bei der Ausgabe eines gesamten Sets nicht festgelegt. Einzelne Elemente können also nicht anhand eines Slices bestimmt werden. Allerdings können Sie mit Mengen einige Operationen durchführen, die aus der Mengenlehre bekannt sind.

**Einmalig,  
ungeordnet**

### 4.6.1 Eigenschaften

Zunächst erzeugen wir ein Set und eine Liste (zum Vergleich) und betrachten einige Eigenschaften:

```
# Liste
li = [8, 2, 5, 5, 5]
print("Liste:", li)
```

```
# Set
s1 = set([8, 2, 5, 5, 5])
print("Set:", s1)
print("Anzahl:", len(s1))
# Elemente
for x in s1:
    print("Element:", x)
if 5 in s1:
    print("5 ist enthalten")
```

**Listing 4.31** Datei `set_eigenschaft.py`

Die Ausgabe lautet:

```
Liste: [8, 2, 5, 5, 5]
Set: {8, 2, 5}
Anzahl: 3
Element: 8
Element: 2
Element: 5
5 ist enthalten
```

Zur Erläuterung:

- set()** ▶ Eine Menge erzeugen Sie mithilfe der Funktion `set()`. Als einziger Parameter wird der Funktion `set()` eine Liste (wie hier) oder ein anderes Objekt übergeben, das durchlaufen werden kann.
- Einmalig** ▶ Sie erkennen den Unterschied zur Liste: In der Liste kann ein Objekt mehrmals vertreten sein, in der Menge nur einmal.
- len()** ▶ Die Funktion `len()` ergibt erwartungsgemäß die Anzahl der Elemente der Menge.
- for, in** ▶ Mithilfe einer `for`-Schleife und des Operators `in` können Sie die Menge durchlaufen. Wiederum mit `in` prüfen Sie, ob ein bestimmtes Element in der Menge enthalten ist.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Ausgabe des Sets lautet `set([...])` statt `{...}`.

## 4.6.2 Funktionen

Betrachten wir einige Funktionen, die Sie auf Mengen anwenden können:

- ▶ Kopieren einer Menge mit `copy()`
- ▶ Hinzufügen von Elementen mit `add()`

- ▶ Entfernen von Elementen mit `discard()`
- ▶ Leeren einer Menge mit `clear()`

Ein Beispiel:

```
# Set
s1 = set([8, 15, "x"])
print("Original:", s1)

# Kopie
s2 = s1.copy()
print("Kopie:", s2)

# Element hinzufügen
s1.add("abc")
print("Element hinzufügt:", s1)

# Element entfernen
s1.discard("x")
print("Element entfernt:", s1)

# Leeren
s1.clear()
print("geleert:", s1)
```

#### **Listing 4.32 Datei `set_funktion.py`**

Es wird die Ausgabe erzeugt:

```
Original: {8, 'x', 15}
Kopie: {8, 'x', 15}
Element hinzufügt: {8, 'x', 'abc', 15}
Element entfernt: {8, 'abc', 15}
geleert: set()
```

Zur Erläuterung:

- |   |                        |
|---|------------------------|
| ▶ Die Funktion <code>copy()</code> erzeugt eine neue Menge als Kopie der alten Menge. | <code>copy()</code>    |
| ▶ Mithilfe der Funktion <code>add()</code> fügen Sie ein Element hinzu.               | <code>add()</code>     |
| ▶ Die Funktion <code>discard()</code> dient zum Löschen eines Elements.               | <code>discard()</code> |
| ▶ Die Funktion <code>clear()</code> leert die Menge von allen Elementen.              | <code>clear()</code>   |

#### **Unterschiede in Python 2**

Die Klammern bei der Anweisung `print` entfallen. Die Ausgabe des Sets lautet `set(...)` statt `{...}`.

### 4.6.3 Operatoren

- < <= > = > Mit den vier Operatoren <, <=, > und >= stellen Sie fest, ob eine Menge eine Teilmenge oder eine echte Teilmenge einer anderen Menge ist. Ein Beispiel:

```
# Sets
s1 = set([8, 2, 5])
s2 = set([2, 8])
s3 = set([2, 5, 8])

print("s1:", s1)
print("s2:", s2)
print("s3:", s3)

# Teilmenge, echte Teilmenge
if s2 < s1:
    print("s2 ist echte Teilmenge von s1")
if s3 <= s1:
    print("s3 ist Teilmenge von s1")
```

**Listing 4.33** Datei `set_teilmenge.py`

Folgende Ausgabe wird erzeugt:

```
s1: {8, 2, 5}
s2: {8, 2}
s3: {8, 2, 5}
s2 ist echte Teilmenge von s1
s3 ist Teilmenge von s1
```

Zur Erläuterung:

- |                        |   |
|------------------------|---|
| <b>Echte Teilmenge</b> | ► Die Menge <code>s2</code> ist eine echte Teilmenge der Menge <code>s1</code> , denn alle Elemente von <code>s2</code> sind in <code>s1</code> enthalten, und <code>s2</code> hat weniger Elemente als <code>s1</code> .   |
| <b>Teilmenge</b>       | ► Die Menge <code>s3</code> ist nur eine <i>normale</i> Teilmenge der Menge <code>s1</code> , denn alle Elemente von <code>s3</code> sind in <code>s1</code> enthalten, aber <code>s3</code> hat ebenso viele Elemente wie <code>s1</code> .<br>► Die Reihenfolge der Elemente bei der Erzeugung ist unerheblich. |

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Ausgabe des Sets lautet `set([...])` statt `{...}`.

- | & - ^ Mithilfe der Operatoren | (oder), & (und), - (minus) und ^ (hoch) können Sie einige Mengenoperationen durchführen. Ein Beispiel:

```

# Sets
s1 = set([8, 15, "x"])
s2 = set([4, "x", "abc", 15])

print("s1:", s1)
print("s2:", s2)

# Vereinigungsmenge
s3 = s1 | s2
print("Vereinigungsmenge:", s3)
# Schnittmenge
s4 = s1 & s2
print("Schnittmenge:", s4)

# Differenzmengen
s5 = s1 - s2
print("Differenzmenge s1-s2:", s5)
s6 = s2 - s1
print("Differenzmenge s2-s1:", s6)

s7 = s2 ^ s1
print("symm. Differenzmenge:", s7)

```

**Listing 4.34** Datei `set_operator.py`

Die Ausgabe lautet:

```

s1: {8, 'x', 15}
s2: {'x', 'abc', 4, 15}
Vereinigungsmenge: {'abc', 4, 8, 'x', 15}
Schnittmenge: {'x', 15}
Differenzmenge s1-s2: {8}
Differenzmenge s2-s1: {'abc', 4}
symm. Differenzmenge: {8, 'abc', 4}

```

Zur Erläuterung:

- ▶ Der Operator `|` (oder) dient zur Vereinigung zweier Mengen. Die entstehende Menge enthält alle Elemente, die in der ersten oder in der zweiten Menge enthalten sind. Auch in der neuen Menge ist jedes Element nach wie vor nur einmal enthalten. **Vereinigungsmenge**
- ▶ Alle Elemente, die in der ersten und in der zweiten Menge enthalten sind, bilden die Schnittmenge. Dies gelingt mithilfe des Operators `&` (und). **Schnittmenge**
- ▶ Bei einer Differenzmenge ist es wichtig zu betrachten, welche Menge von welcher anderen Menge abgezogen wird. Mithilfe des Operators `-` (minus) werden zwei verschiedene Differenzmengen erstellt. Die Operation `s1-s2` zieht von der Menge `s1` alle Elemente ab, die auch in `s2` enthalten sind. Bei der Operation `s2-s1` verhält es sich umgekehrt. **Differenzmenge**

### Symmetrische Differenzmenge

- Bei der symmetrischen Differenzmenge werden mithilfe des Operators `^` (hoch) die Elemente ermittelt, die nur in einer der beiden Mengen enthalten sind.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Ausgabe des Sets lautet `set([...])` statt `{...}`.

#### 4.6.4 Frozenset

##### Unveränderlich

Ein Sonderfall eines Sets ist ein Frozenset. Ein Frozenset ist im Unterschied zum Set *eingefroren* (englisch: *frozen*), also unveränderlich. Ein Beispiel:

```
# Set
s = set([8, 15, "x", 8])
print("Set:", s)

# Frozenset
fs = frozenset([8, 15, "x", 8])
print("Frozenset:", fs)
for x in fs:
    print(x)
try:
    fs.discard("x")
except:
    print("Fehler")
```

**Listing 4.35** Datei `set_frozenset.py`

- Die Ausgabe sieht wie folgt aus:

```
Set: {8, 'x', 15}
Frozenset: frozenset({8, 'x', 15})
8
x
15
Fehler
```

Zur Erläuterung:

- |                          |  |
|--------------------------|--|
| <code>frozenset()</code> | ► Bei der Erzeugung des Frozensets mithilfe der Funktion <code>frozenset()</code> wird (wie beim Set) darauf geachtet, dass jedes Element nur einmal vorkommt.   |
| <code>frozenset</code>   | ► Bei der Ausgabe wird das Frozenset besonders gekennzeichnet: durch den Begriff <code>frozenset</code> und zusätzliche Klammern.<br>► Die einzelnen Elemente geben Sie mithilfe von <code>for</code> und <code>in</code> wie gewohnt aus.<br>► Allerdings führt der Versuch, ein Frozenset zu verändern, zu einem Fehler. |

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Ausgabe des Sets ist `set([...])` statt `{...}`. Die Ausgabe des Frozensets ist `frozenset([...])` statt `frozenset({...})`.

## 4.7 Wahrheitswerte und Nichts

Objekte und Ausdrücke können wahr oder falsch sein, außerdem gibt es auch das Nichts-Objekt. Der folgende Abschnitt erläutert die Zusammenhänge.

### 4.7.1 Wahrheitswerte True und False

Besonders im Zusammenhang mit Bedingungsprüfungen (`if`, `while`) wird der Wahrheitswert eines Ausdrucks benötigt.

Beispiel: Wenn eine Zahl größer als 10 ist, sollen bestimmte Anweisungen ausgeführt werden. Der dabei benötigte Ausdruck  $x > 10$  ist wahr, wenn  $x$  einen Zahlenwert größer als 10 hat. Er ist falsch, wenn  $x$  einen Zahlenwert kleiner oder gleich 10 hat.

Die Ausdrücke liefern eines der beiden Schlüsselworte `True` (wahr) oder `False` (falsch). Dies sind die einzigen Objekte des Datentyps `bool`.

Es gibt außerdem die Funktion `bool()`, die den Wahrheitswert eines Ausdrucks oder eines Objekts zurückgibt. Neben diesen Ausdrücken haben in Python auch Objekte einen Wahrheitswert:

► Folgende Objekte sind wahr (liefern `True`):

- eine Zahl ungleich 0, also größer als 0 oder kleiner als 0
- eine nicht leere Sequenz (String, Liste, Tupel)
- ein nicht leeres Dictionary
- eine nicht leere Menge

**Wahre Objekte**

► Folgende Objekte sind falsch (liefern `False`):

- eine Zahl, die den Wert 0 hat
- eine leere Sequenz (String: "", Liste [], Tupel ())
- ein leeres Dictionary: {}
- eine leere Menge: `set()`, `frozenset()`
- die Konstante `None` (siehe [Abschnitt 4.7.2](#), »Nichts, None«)

**Falsche Objekte**

Endlosschleifen, die nur mit einem `break` verlassen werden können, werden gerne mit `while(1)` konstruiert. Diese Bedingung ist immer wahr. Sie können natürlich auch `while(True)` schreiben.

**Endlosschleife**

**Länge gleich 0** Gilt für eine Objektsammlung `len(x) == 0`, so ist das Objekt `x` falsch. Im folgenden Programm wird der Wahrheitswert der genannten Objekte an Beispielen dargestellt, überprüft und ausgegeben.

```
True, False # True und False
W = True
print("Wahrheitswert:", W)
W = False
print("Wahrheitswert:", W)
W = 5>3
print("5>3:", W)
W = 5<3
print("5<3:", W)
print()
# Datentyp
W = 5>3
print("Typ von 5>3: ", type(W))
print()
```

```
Zahl # wahre Zahl
Z = 5 + 0.001 - 5
print("Zahl:", Z)
if Z:
    print("Zahl ist", bool(Z))

# nicht wahre Zahl
Z = 5.75 - 5.75
print("Zahl:", Z)
if not Z:
    print("Zahl ist", bool(Z))
print()
```

```
Zeichenkette # String
S = "Kurt"
print("String:", S)
if S:
    print("String ist nicht leer, also", bool(S))
print()
```

```
Liste # Liste
L = [3,4]
print("Liste vorher:", L)
del L[0:2]
print("Liste nachher:", L)
if not L:
    print("Liste ist leer, also", bool(L))
print()
```

```

# Tupel                                     Tupel
T = (5,8,2)
print("Tupel:", T)
if T:
    print("Tupel ist nicht leer, also", bool(T))
print()

# Dictionary                                Dictionary
D = {"Julia":28, "Werner":32}
print("Dictionary vorher:", D)
del D["Julia"]
del D["Werner"]
print("Dictionary nachher:", D)
if not D:
    print("Dictionary ist leer, also", bool(D))
print()

# Set                                         Set
S = set([5, 7.5, "abc"])
print("Set vorher:", S)
S.clear()
print("Set nachher:", S)
if not S:
    print("Set ist leer, also", bool(S))
print()

```

**Listing 4.36** Datei wahrheitswert.py

Das Programm erzeugt die Ausgabe:

```

Wahrheitswert: True
Wahrheitswert: False
5>3: True
5<3: False
Typ von 5>3: <class 'bool'>
Zahl: 0.001000000000000334
Zahl ist True
Zahl: 0.0
Zahl ist False

String: Kurt
String ist nicht leer, also True
Liste vorher: [3, 4]
Liste nachher: []
Liste ist leer, also False
Tupel: (5, 8, 2)
Tupel ist nicht leer, also True
Dictionary vorher: {'Werner': 32, 'Julia': 28}

```

```
Dictionary nachher: {}
Dictionary ist leer, also False
Set vorher: {7.5, 'abc', 5}
Set nachher: set()
Set ist leer, also False
```

Zur Erläuterung:

- |                       |  |
|-----------------------|--|
| Typ <code>bool</code> | ► Der Variablen <code>W</code> werden Wahrheitswerte bzw. die Ergebnisse von Vergleichsausdrücken, also auch Wahrheitswerte, zugewiesen. Der Datentyp der Wahrheitswerte ist <code>bool</code> .             |
|                       | ► Sobald das Ergebnis einer Berechnung von 0 abweicht, ergibt sich der Wahrheitswert <code>True</code> .   |
| Leere Objekte         | ► String, Liste, Tupel, Dictionary und Set ergeben <code>False</code> , falls sie leer sind, und <code>True</code> , falls sie nicht leer sind. Dies können Sie zur Prüfung der betreffenden Objekte nutzen. |

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Ausgabe des Sets lautet `set([...])` statt `{...}`. Die Funktion `type()` liefert `type` statt `class`.

## 4.7.2 Nichts, `None`

- |                       |  |
|-----------------------|--|
| <code>NoneType</code> | Das Schlüsselwort <code>None</code> bezeichnet das Nichts-Objekt. <code>None</code> ist das einzige Objekt des Datentyps <code>NoneType</code> . |
| Rückgabewert          | Funktionen ohne Rückgabewert liefern <code>None</code> zurück. Dies kann ein Hinweis darauf sein,  |
|                       | ► dass Sie eine Funktion falsch eingesetzt haben, bei der Sie einen Rückgabewert erwartet haben, oder  |
|                       | ► dass eine Funktion kein Ergebnis liefert hat, obwohl dies erwartet wurde.  |

Ein Beispiel:

```
# Funktion
def quotient(a, b):
    try:
        c = a/b
        return c
    except:
        print("Funktion meldet Fehler")

# liefert Ergebnis
erg = quotient(7,4)
```

```

if erg:
    print("Ergebnis:", erg)
print()

# liefert Fehler
erg = quotient(7,0)
if not erg:
    print("Programm meldet Fehler")
print("Ergebnis:", erg)
print("Typ des Ergebnisses:", type(erg))
print()
# Konstante None
Z = None
print("Z:", Z)
if Z is None:
    print("Objekt ist das Nichts, also", bool(Z))

```

**Listing 4.37** Datei nichts.py

Die Ausgabe lautet:

**Ergebnis: 1.75**  
**Funktion meldet Fehler**  
**Programm meldet Fehler**  
**Ergebnis: None**  
**Typ des Ergebnisses: <class 'NoneType'>**  
**Z: None**  
**Objekt ist das Nichts, also False**

Zur Erläuterung:

- ▶ Zunächst wird die Funktion `quotient()` definiert. Diese berechnet den Quotient aus zwei Zahlen. Sie kann nicht den Wert 0 als Ergebnis liefern.
- ▶ Wenn der Quotient regulär berechnet werden kann, wird das Ergebnis mit Hilfe von `return` zurückgeliefert. Dies ist beim ersten Aufruf der Funktion der Fall.
- ▶ Tritt ein Fehler auf, so wird nichts zurückgeliefert. Die Variable `erg` erhält also den Wert `None`. Dies können Sie abfragen und damit feststellen, dass die Funktion kein nutzbares Ergebnis geliefert hat.
- ▶ Das Nichts-Objekt hat den Wahrheitswert `False`.

**Funktion liefert  
None**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion `type()` liefert `type` statt `class`. Die Funktion `quotient()` wird mit 7.0 als erstem Parameter aufgerufen, damit das richtige Ergebnis berechnet wird.

## 4.8 Referenz, Identität und Kopie

In diesem Abschnitt erläutere ich den Zusammenhang zwischen Objekten und Referenzen. Wir untersuchen die Identität von Objekten und erzeugen Kopien von Objekten.

### 4.8.1 Referenz und Identität

<b>Referenz</b>	Der Name eines Objekts ist im Grunde nur eine Referenz auf ein Objekt.
<b>Operator is</b>	Die Zuweisung dieser Referenz an einen anderen Namen erzeugt eine zweite Referenz auf das gleiche Objekt. Mithilfe des Identitätsoperators <code>is</code> können Sie feststellen, dass die beiden Referenzen auf das gleiche Objekt verweisen.  Wird das Objekt über die zweite Referenz geändert, so zeigt sich eines der beiden folgenden Verhalten:
<b>Zweites Objekt</b>	<ul style="list-style-type: none"> <li>▶ Im Fall eines einfachen Objekts wie Zahl oder String wird ein zweites Objekt erzeugt, in dem der neue Wert gespeichert wird. Die beiden Referenzen verweisen dann auf zwei verschiedene Objekte.</li> </ul>
<b>Zweite Referenz</b>	<ul style="list-style-type: none"> <li>▶ Im Fall eines nicht einfachen Objekts wie Liste, Dictionary usw. wird das Originalobjekt geändert. Es gibt nach wie vor ein Objekt mit zwei verschiedenen Referenzen.</li> </ul> <p>Mithilfe des Operators <code>==</code> stellen Sie fest, ob zwei Objekte den gleichen Inhalt haben, ob also z. B. zwei Listen die gleichen Elemente enthalten.</p>
<b>Operatoren is, ==</b>	Im folgenden Beispiel werden nacheinander eine Zahl, ein String und eine Liste erzeugt und zweimal referenziert. Anschließend wird der zweiten Referenz jeweils ein neuer Inhalt zugewiesen. Identität und Inhalt werden anhand der beiden Operatoren <code>is</code> und <code>==</code> festgestellt.
	<pre># Kopie einer Zahl print("Zahl:") x = 12.5 y = x print("gleiches Objekt:", x is y) y = 15.8 print("gleiches Objekt:", x is y) print("gleicher Inhalt:", x == y) print()  # Kopie eines Strings print("String:") x = "Robinson" y = x print("gleiches Objekt:", x is y) y = "Freitag"</pre>

```

print("gleiches Objekt:", x is y)
print("gleicher Inhalt:", x == y)
print()

# Zweite Referenz auf eine Liste
print("Liste:")
x = [23, "hallo", -7.5]
y = x
print("gleiches Objekt:", x is y)
y[1] = "welt"
print("gleiches Objekt:", x is y)

```

**Listing 4.38** Datei referenz.py

Es wird die folgende Ausgabe erzeugt:

**Zahl:**

```

gleiches Objekt: True
gleiches Objekt: False
gleicher Inhalt: False

```

**String:**

```

gleiches Objekt: True
gleiches Objekt: False
gleicher Inhalt: False

```

**Liste:**

```

gleiches Objekt: True
gleiches Objekt: True

```

Zur Erläuterung:

- ▶ Die Ausgabe zeigt, dass die Objekte zunächst jeweils identisch sind.
- ▶ Nach der Zuweisung eines neuen Werts zeigt sich, dass im Fall von Zahl und String ein zweites Objekt erzeugt wird. Da die beiden neuen Objekte durch Zuweisung eines anderen Werts entstanden sind, sind natürlich auch die Inhalte unterschiedlich. Zweites Objekt
- ▶ Die Liste (hier stellvertretend auch für andere Objekte) existiert insgesamt nur einmal, auch wenn einzelne Elemente der Liste verändert wurden. Sie können über beide Referenzen auf die Liste zugreifen. Zweite Referenz

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 4.8.2 Ressourcen sparen

### Speicherplatz sparen

Python spart gerne Ressourcen. Dies kann zu einem ungewöhnlichen Verhalten führen: Wenn einem Objekt über eine Referenz ein Wert zugewiesen wird und auf denselben Wert bereits von einer anderen Referenz verwiesen wird, so kann es geschehen, dass die beiden Referenzen anschließend auf dasselbe Objekt verweisen. Python spart also Speicherplatz.

### Referenz löschen

Die Anweisung `del` dient zur Löschung von nicht mehr benötigten Referenzen. Ein Objekt, auf das zwei Referenzen verweisen, wird durch das Löschen der ersten Referenz nicht gelöscht. Ein Beispiel:

```
# Ein Objekt, zwei Referenzen
x = 42
y = 42
print("x:", x, "y:", y, "identisch:", x is y)

# Zweites Objekt
y = 56
print("x:", x, "y:", y, "identisch:", x is y)

# Ressourcen sparen
y = 42
print("x:", x, "y:", y, "identisch:", x is y)
# Entfernen, Schritt 1
del y
print("x:", x)
# Entfernen, Schritt 2
del x
try:
    print("x:", x)
except:
    print("Fehler")
```

**Listing 4.39** Datei `ressourcen.py`

Es wird die folgende Ausgabe erzeugt:

```
x: 42 y: 42 identisch: True
x: 42 y: 56 identisch: False
x: 42 y: 42 identisch: True
x: 42
Fehler
```

Zur Erläuterung:

### Ein Objekt

- ▶ Zunächst erhalten die Referenzen `x` und `y` den gleichen Wert. Sie stellen fest: Es handelt sich nur um ein Objekt mit zwei Referenzen.

### Zwei Objekte

- ▶ Der Wert von `y` wird geändert. Damit gibt es nun zwei Referenzen auf zwei verschiedene Objekte.

- ▶ Der Wert von `y` wird wieder auf den früheren Wert zurückgesetzt. Nun gibt es wieder nur noch ein Objekt.
- ▶ Die Referenz `y` wird gelöscht. Das Objekt existiert weiterhin und kann über die Referenz `x` erreicht werden.
- ▶ Die Referenz `x` wird gelöscht. Nun führt die Ausgabe über diese Referenz zu einem Fehler, da der Name nicht mehr existiert.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 4.8.3 Objekte kopieren

Echte Kopien von nicht einfachen Objekten können Sie durch Erzeugen eines leeren Objekts und Anhängen oder Hinzufügen der einzelnen Elemente erzeugen. Für umfangreiche Objekte, die wiederum andere Objekte enthalten, können Sie sich auch der Funktion `deepcopy()` aus dem Modul `copy` bedienen. Beide Vorgehensweisen zeigt das folgende Programm.

`copy.deepcopy()`

```
# Modul copy
import copy

# Kopie einer Liste, Methode 1
x = [23,"hallo",-7.5]
y = []
for i in x:          # Elemente einzeln kopieren
    y.append(i)
print("gleiches Objekt:", x is y)
print("gleicher Inhalt:", x == y)
print()

# Kopie einer Liste, Methode 2
x = [23,[{"Berlin","Hamburg"},-7.5,12,67]]
y = copy.deepcopy(x)      # Funktion zur Tiefenkopie
print("gleiches Objekt:", x is y)
print("gleicher Inhalt:", x == y)
```

**Listing 4.40** Datei kopieren.py

Das Programm erzeugt die Ausgabe:

**gleiches Objekt: False**  
**gleicher Inhalt: True**

**gleiches Objekt: False**  
**gleicher Inhalt: True**

Zur Erläuterung:

- Die Ausgabe zeigt, dass in beiden Fällen jeweils ein neues Objekt erzeugt wurde. Die Inhalte der beiden Objekte sind allerdings gleich.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

# Kapitel 5

## Weiterführende Programmierung

In diesem Kapitel werden die Kenntnisse aus dem Programmierkurs im Zusammenhang mit den verschiedenen Objekttypen durch nützliche Praxistipps erweitert.

### 5.1 Allgemeines

Im ersten Teil des Kapitels erläutere ich einige nützliche Techniken, die keinem bestimmten Thema zuzuordnen sind.

#### 5.1.1 Kombinierte Zuweisungsoperatoren

Neben der einfachen Zuweisung eines Werts zu einer Variablen gibt es auch die kombinierten Zuweisungsoperatoren. Diese verbinden die normalen Operatoren für Zahlen oder Zeichenketten mit der Zuweisung eines Werts. Die betreffende Variable wird also unmittelbar um den genannten Wert verändert. Dies ist besonders bei umfangreichen Ausdrücken oder bei längeren Variablennamen sinnvoll. Ein Beispiel:

Kürzere  
Schreibweise

Der Ausdruck `TemperaturInCelsius += 5` ist überschaubarer als der Ausdruck `TemperaturInCelsius = TemperaturInCelsius + 5`. Beide Ausdrücke erhöhen den Wert der Variablen `TemperaturInCelsius` um 5.

Es folgt ein Beispiel, in dem kombinierte Zuweisungsoperatoren für Zahlen und für Zeichenketten eingesetzt werden.

```
# Kombinierte Zuweisungsoperatoren fuer Zahlen
x = 12
print(x)
x += 3      # Erhöhen von x
print(x)
x -= 9      # Vermindern von x
print(x)
x **= 2     # Quadrieren von x
print(x)
x *= 3      # Verdreifachen von x
print(x)

x //= 7     # ganzzahliges Teilen von x
print(x)

x /= 4      # Teilen von x
print(x)
```

```
x %= 2      # Dividieren, Rest berechnen
print(x)

# Kombinierte Zuweisungsoperatoren fuer Zeichenketten
t = " hallo"
print(t)

t += " python"  # Anhaengen an t
print(t)

t *= 3        # Verdreifachen von t
print(t)
```

**Listing 5.1** Datei zuweisung\_kombiniert.py

Die Ausgabe:

```
12
15
6
36
108
15
3.75
1.75
hallo
hallo python
hallo python hallo python hallo python
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die kombinierte Division muss `x /= 4.0` lauten, damit das richtige Ergebnis berechnet wird.

Zur Erläuterung:

- ▶ Die Variable `x` erhält zunächst den Wert 12. Durch die anschließenden Anweisungen wird der Wert von `x` jedes Mal verändert.
- `+== **= *=` ▶ Die Variable `x` wird um 3 erhöht (= 15), anschließend um 9 vermindert (= 6), dann hoch 2 gerechnet (= 36) und mit 3 multipliziert (= 108).
- `//= /= %=` ▶ Es folgt eine Ganzzahldivision:  $108 / 7 = 15.428\dots$  Die Nachkommastellen werden dabei abgeschnitten. Die verbleibende Zahl (15) wird durch 4 geteilt (= 3,75). Zuletzt wird der Rest der Division durch 2 berechnet (= 1,75).
- ▶ Die Variable `t` erhält zunächst den Wert `hallo`.
- `+*=` ▶ Sie wird anschließend verlängert und vervielfacht.

Bei all diesen Operationen ist darauf zu achten, dass die betreffende Variable vorher bereits einen Wert hat, sonst tritt ein Fehler auf.

### 5.1.2 Programmzeile in mehreren Zeilen

In Abschnitt 2.3.6, »Lange Ausgaben«, wurde bereits erläutert, wie Sie eine lange Zeichenkette bei Einsatz der Funktion `print()` über mehrere Zeilen verteilen. In diesem Abschnitt zeige ich, wie Sie lange Programmzeilen generell zerlegen können, unter anderem mithilfe des Zeichens \. Ein Beispiel:

```
print("Umrechnung von Celsius in Fahrenheit")

# Trennung einer Zeichenkette
print("Bitte geben Sie eine"
      " Temperatur in Celsius ein: ")
TemperaturInCelsius = float(input())

# Trennung eines Ausdrucks
TemperaturInFahrenheit = TemperaturInCelsius * 9 / 5 + 32

# Trennung nach einem Komma
print(TemperaturInCelsius, "Grad Celsius entsprechen",
      TemperaturInFahrenheit, "Grad Fahrenheit")
```

Zeichen \

**Listing 5.2 Datei zeile\_lang.py**

Die Ausgabe lautet:

```
Umrechnung von Celsius in Fahrenheit
Bitte geben Sie eine Temperatur in Celsius ein:
5.2
5.2 Grad Celsius entsprechen 41.36 Grad Fahrenheit
```

Zur Erläuterung:

- ▶ Zunächst wird eine Zeichenkette in gewohnter Weise zerlegt und in zwei Zeilen geschrieben. Jeder Teil der Zeichenkette wird in Anführungszeichen gesetzt. Ein trennendes Leerzeichen zwischen den beiden Teilen muss von Hand eingegeben werden.
- ▶ Eine längere Programmzeile mit einer Berechnung wird mithilfe des Zeichens \ zerlegt. Dieses Zeichen zeigt an, dass die Programmzeile in der nächsten Zeile fortgesetzt wird.
- ▶ Einfacher ist die Zerlegung der Programmzeile, wenn darin Kommata auftreten, wie z. B. beim Aufruf einer Funktion mit mehreren Parametern oder bei der Zuweisung einer Liste. Hier können Sie einfach nach einem Komma trennen.

Zeichen \

Nach Komma trennen

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch aller langen Programmzeilen eingesetzt.

#### 5.1.3 Eingabe mit Hilfestellung

- `input()` Die eingebaute Funktion `input()` zur Eingabe von Zeichenketten hat einen optionalen Parameter. Dabei handelt es sich um eine Zeichenkette, die eine hilfreiche Information für die Eingabe enthalten sollte. Dies spart eine Zeile mit der Funktion `print()`.

Im folgenden Beispiel wird zunächst die Summe aus drei eingegebenen Zahlen berechnet. Anschließend wird der Benutzer aufgefordert, den jeweiligen Namen der Hauptstadt eines von insgesamt drei Ländern einzugeben.

```
# Berechnung einer Summe
summe = 0
for i in range (1,4):
    fehler = True
    while fehler:
        zahl = input(str(i) + ". Zahl eingeben: ")
        try:
            summe += float(zahl)
            fehler = False
        except:
            print("Das war keine Zahl")
            fehler = True

print("Summe:", summe)
print()

# Geografiespiel
hauptstadt = {"Italien":"Rom", "Spanien":"Madrid",
              "Portugal":"Lissabon"}
hs = hauptstadt.items()

for land, stadt in hs:
    eingabe = input("Bitte die Hauptstadt von "
                  + land + " eingeben: ")
    if eingabe==stadt:
        print("Richtig")
    else:
        print("Falsch, richtig ist:", stadt)
```

**Listing 5.3** Datei `eingabe_hilfe.py`

Die Ausgabe lautet:

```
1. Zahl eingeben: 4
2. Zahl eingeben: 6,7
Das war keine Zahl
2. Zahl eingeben: 6.7
3. Zahl eingeben: 2
Summe: 12.7
```

Bitte die Hauptstadt von Spanien eingeben: Madrid

Richtig

Bitte die Hauptstadt von Italien eingeben: Neapel

Falsch, richtig ist: Rom

Bitte die Hauptstadt von Portugal eingeben: Lissabon

Richtig

Zur Erläuterung:

- ▶ Bei der Eingabe zur Berechnung einer Summe wird als Hilfestellung die laufende Nummer der einzugebenden Zahl ausgegeben. Es ist zu beachten, dass diese Nummer in eine Zeichenkette umgewandelt und mit dem restlichen Kommentar verkettet werden muss.
- ▶ Aufgrund der Ausnahmebehandlung muss eine fehlerhafte Eingabe wiederholt werden. In diesem Fall wird wieder die gleiche laufende Nummer als Hilfestellung ausgegeben.
- ▶ Bei der Zeichenketteneingabe wird das erste Element des jeweiligen Tupel-Elements, also das Land, mit ausgegeben. Der Benutzer gibt die Hauptstadt ein und erhält eine Rückmeldung, ob seine Eingabe richtig oder falsch ist.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe (auch mit Hilfestellung) heißt `raw_input()`.

#### 5.1.4 Anweisung »pass«

Die Anweisung `pass` bewirkt, dass nichts ausgeführt wird. Wozu existiert sie dann? Einige mögliche Einsatzzwecke sind:

Nichts ausführen

- ▶ Sie entwickeln ein Programm, in dem unter anderem eine Funktion aufgerufen wird. In der ersten Entwicklungsphase soll das Hauptprogramm geschrieben werden. Der Funktionsaufruf soll bereits an der richtigen Stelle platziert werden, aber noch keine Auswirkungen haben. Die Funktionsdefinition enthält in diesem Fall nur die Anweisung `pass`.

Funktions-Dummy

- Das Programm enthält eine einfache oder mehrfache Verzweigung, bei der in einem bestimmten Zweig nichts ausgeführt werden soll. Dieser Zweig soll aber trotzdem erscheinen, um den Programmablauf klarer zu machen.

Ein Programm mit Beispielen zu diesen Fällen sähe wie folgt aus:

```
# Funktions-Dummy
def QuadraturDesKreises():
    pass

# Funktionsaufruf
QuadraturDesKreises()

# Nur else-Zweig interessant
a = -12
b = 6
c = 6.2

if a >= 0 and b >= 0 and c >= 0:
    pass
else:
    print("Eine der Zahlen ist negativ")
# Ein Zweig nicht interessant
if a == 1:
    print("Fall 1")
elif a == 2:
    print("Fall 2")
elif a < 0:
    pass
else:
    print("Ansonsten")
```

**Listing 5.4** Datei `anweisung_pass.py`

Die Ausgabe lautet:

**Eine der Zahlen ist negativ**

Zur Erläuterung:

- Die Funktion `QuadraturDesKreises()` dient vorerst nur als Dummy und wird erst zu einem späteren Zeitpunkt mit Inhalten gefüllt. Sie ist aber bereits im Programm eingebaut und kann aufgerufen werden.
- Bei der einfachen Verzweigung soll nur im `else`-Zweig eine Ausgabe erfolgen.
- Bei der mehrfachen Verzweigung soll nur eine der drei möglichen Ausgaben erfolgen, falls der untersuchte Wert größer oder gleich 0 ist.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 5.1.5 Funktionen »eval()« und »exec()«

Die Funktionen `eval()` und `exec()` dienen zum Zusammensetzen von Python-Code. Mit diesen Funktionen lassen sich Anweisungen dynamisch aus Zeichenketten bilden.

- ▶ Die Funktion `eval()` evaluiert den zusammengesetzten Ausdruck, ermittelt also den Wert (engl.: *value*) des Ausdrucks. Auswerten
- ▶ Die Funktion `exec()` führt eine zusammengesetzte Anweisung aus. Ausführen

Ein Beispiel:

```
import math

# Zwei Funktionen
def mw1(a,b):
    c = (a+b)/2
    return c

def mw2(a,b):
    c = math.sqrt(a*b)
    return c

# eval
for i in 1,2:
    t = "mw" + str(i) + "(3,4)"
    c = eval(t)
    print(c)
print()

# exec
for i in 1,2:
    t = "print(mw" + str(i) + "(3,4))"
    exec(t)
```

**Listing 5.5** Datei `eval_exec.py`

Folgende Ausgabe wird erzeugt:

3.5  
3.4641016151377544

3.5  
3.4641016151377544

Zur Erläuterung:

- ▶ Es werden zunächst die Funktion `mw1()` zur Ermittlung des arithmetischen Mittelwerts und die Funktion `mw2()` zur Ermittlung des geometrischen Mittelwerts zweier Zahlen definiert.
- ▶ Die Namen der beiden Funktionen, `mw1` und `mw2`, unterscheiden sich nur geringfügig.
- ▶ Für den Aufruf von `eval()` werden zwei Ausdrücke in Zeichenketten zusammengesetzt: "`mw1(3,4)`" und "`mw2(3,4)`". Mithilfe dieser Zusammensetzung können die beiden Aufrufe erfolgen: `c = mw1(3,4)` und `c = mw2(3,4)`. Der Rückgabewert wird jeweils in der Variablen `c` gespeichert, die anschließend ausgegeben wird.
- ▶ Für den Aufruf von `exec()` werden zwei Anweisungen in Zeichenketten zusammengesetzt: "`print(mw1(3,4))`" und "`print(mw2(3,4))`". Diese beiden Anweisungen werden ausgeführt. Der Rückgabewert wird direkt ausgegeben.

Ausdruck  
zusammensetzen

Anweisung  
zusammensetzen

Ein weiteres Beispiel für die Anwendung der Funktion `eval()` finden Sie in [Abschnitt 11.3.2, »Ein einfacher Taschenrechner«](#), und [Abschnitt 11.3.3, »Methode grid\(\)«](#).

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Bei `exec` handelt es sich ebenfalls um eine Anweisung und nicht um eine Funktion, daher entfallen auch hier die Klammern. In der Funktion `mw1()` müssen Sie durch `2.0` statt durch `2` dividieren, damit das richtige Ergebnis berechnet wird.

## 5.2 Ausgabe und Formatierung

In diesem Abschnitt erläutere ich weitere Möglichkeiten zur Ausgabe mithilfe der Funktion `print()` und zur Formatierung von Ausgaben.

### 5.2.1 Funktion »print()«

Die Funktion `print()` haben wir bereits mehrfach eingesetzt. Sie bietet noch weitere Möglichkeiten:

- ▶ **Separator** Der Separator, der die ausgegebenen Objekte voneinander trennt, kann verändert werden. Er wird mit `sep` bezeichnet. Normalerweise wird ein Leerzeichen zur Trennung ausgegeben.
- ▶ **Zeilenende** Das Zeilenende, das normalerweise nach einer Ausgabe folgt, kann verändert werden. Es wird mit `end` bezeichnet.

Ein Beispiel:

```
# Berechnung
a = 23
b = 7.5
c = a + b

# normale Ausgabe
print("Ergebnis:", a, "+", b, "=", c)

# Ausgabe ohne Zeilenende und Leerzeichen
print("Ergebnis: ", end="")
print(a, "+", b, "=", c, sep="")

# Neue Zeile
print()

# Liste
stadt = ["Hamburg", "Berlin", "Augsburg"]
for x in stadt:
    print(x)
for x in stadt:
    print("Stadt", x, sep="=>", end=" # ")
```

**Listing 5.6** Datei funktion\_print.py

Die Ausgabe sieht wie folgt aus:

Ergebnis: 23 + 7.5 = 30.5

Ergebnis: 23+7.5=30.5

```
Hamburg
Berlin
Augsburg
Stadt=>Hamburg # Stadt=>Berlin # Stadt=>Augsburg #
```

Zur Erläuterung:

- ▶ Zunächst wird die normale Ausgabe einer Berechnung erzeugt, mit Leerzeichen zwischen den einzelnen Elementen.
- ▶ Es folgen zwei Anweisungen, jeweils mit der Funktion `print()`. Bisher führte dies dazu, dass zwei Ausgabezeilen erzeugt wurden.
  - Weisen Sie allerdings dem Parameter `end` eine leere Zeichenkette zu, so wird am Ende der Zeile nichts ausgegeben. Die nächste Ausgabe erfolgt dann in der gleichen Zeile.
  - Bei der nächsten Ausgabe wurde dem Parameter `sep` eine leere Zeichenkette zugewiesen. Dies führt dazu, dass die Ausgaben ohne Leerzeichen direkt hintereinander stehen.

- ▶ Es folgt die Ausgabe der Elemente einer Liste, zunächst in gewohnter Form. In der zweiten Version werden sowohl der Separator als auch das Zeilenende verändert.

### Unterschiede in Python 2

Die Anweisung `print` in Python 2 bietet nicht die komfortablen Möglichkeiten der Funktion `print()` aus Python 3. Im folgenden Programm wurde die gleiche Ausgabe mit anderen Mitteln erzeugt:

```
# Berechnung
a = 23
b = 7.5
c = a + b
# normale Ausgabe
print "Ergebnis:", a, "+", b, "=", c

# Ausgabe ohne Zeilenende und Leerzeichen
print "Ergebnis: " + str(a) + "+" + str(b) \
+ "=" + str(c)

# Neue Zeile
print

# Liste
stadt = ["Hamburg", "Berlin", "Augsburg"]
for x in stadt:
    print x
ausgabe = ""

for x in stadt:
    ausgabe += "Stadt" + ">" + x + " # "
print ausgabe
```

**Listing 5.7** Datei `funktion_print.py` für Python 2

Zur Erläuterung:

- ▶ Das Weglassen der Leerzeichen zwischen den einzelnen Ausgaben erreichen Sie durch den Einsatz des Operators `+`, der Ausgaben direkt aneinanderkettet. Allerdings müssen Sie dazu vorher die Zahlen mit der Funktion `str()` in Zeichenketten umwandeln.
- ▶ Das Trennen einzelner Ausgabeelemente durch bestimmte Trennzeichen wurde erreicht, indem die Ausgabe stückweise in einer Variablen zusammengesetzt und erst anschließend vollständig ausgegeben wird.

## 5.2.2 Formatierte Ausgabe mit »format()«

Schleifen unterstützen unter anderem die Durchführung von Berechnungen für eine Liste von Werten. Bei der Ausgabe steht man häufig vor dem Problem, diese Werte übersichtlich in Tabellenform anzugeben.

Die eingebaute Funktion `format()` bietet Möglichkeiten für eine einheitliche, formatierte Ausgabe von Zahlen und Zeichenketten. Sie können unter anderem:

- ▶ die Mindestausgabebreite der Zahlen bestimmen
- ▶ die Anzahl der Nachkommastellen bestimmen

`format()`Breite, Anzahl  
Stellen

Eine weitere, recht ähnliche Schreibweise lernen Sie in [Abschnitt 5.2.3, »Formatierte Ausgabe wie in C«](#), kennen. Zunächst sollen zwei Zahlen mit Nachkommastellen auf verschiedene Arten ausgegeben werden:

```
# 1: Zahl mit Nachkommastellen
x = 100/7
y = 2/7
print("Zahlen:")
print(x, y)
print()

# 2: Format f
print("Format f")
print("{0:f} {0:f} {1:f}".format(x,y))
print("{0:15.10f} {1:.25f}".format(x,y))
print()

# 3: Format e
print("Format e")
print("{0:e}".format(x))
print("{0:12.3e}".format(x))
print("{0:.3e}".format(x))
print()

# 4: Format %
print("Format %")
print("{0:}%".format(y))
print("{0:12.3 %}".format(y))
print("{0:.3 %}".format(y))
```

**Listing 5.8** Datei `ausgabe_formatieren.py`

Folgende Ausgabe wird erzeugt:

**Zahlen:**

**14.285714285714286    0.2857142857142857**

### Format f

```
14.285714 14.285714 0.285714
14.2857142857 0.2857142857142856984253854
```

### Format e

```
1.428571e+01
1.429e+01
1.429e+01
```

### Format %

```
28.571429 %
28.571 %
28.571 %
```

Zur Erläuterung:

- ▶ Zu Block 1: Die Variablen `x` und `y` erhalten den Wert von `100/7` bzw. von `2/7`. Sie werden zunächst unformatiert ausgegeben. Dies führt zu einer Ausgabe in einer Gesamtbreite von 13 Stellen. Die Anzahl der Nachkommastellen richtet sich nach der Größe der Zahl.
- ▶ Zu Block 2: Es wird eine Ausgabezeichenkette gebildet, in der die gewünschten Formate stehen. Diese Ausgabezeichenkette ist ein Objekt. Für dieses Objekt wird die Methode `format()` aufgerufen. Parameter der Methode `format()` sind die auszugebenden Variablen bzw. Werte.

### Laufende Nummer

- ▶ Die auszugebenden Variablen bzw. Werte werden intern nummeriert, beginnend bei 0. Diese Nummer wird in der Ausgabezeichenkette benötigt.

### Geschweifte Klammern

- ▶ Die Ausgabezeichenkette enthält normalen Text und Ausdrücke in geschweiften Klammern. Diese Ausdrücke in geschweiften Klammern bestehen wiederum aus
  - den oben genannten laufenden Nummern der Variablen bzw. des Werts,
  - einem Doppelpunkt und
  - einer Formatierungsangabe.

### Format f

- ▶ Die erste Formatierungsangabe in Block 2 ist ein einzelnes `f`. Es steht für: Ausgabe als Zahl mit sechs Nachkommastellen. Die Variable `x` (laufende Nummer 0) wird zweimal in dieser Form ausgegeben, die Variable `y` (laufende Nummer 1) einmal.

- ▶ Die zweite Formatierungsangabe in Block 2 enthält die folgenden Angaben:
  - Variable `x` (laufende Nummer 0) in `15.10f`: Breite der Ausgabe insgesamt 15 Stellen, Zahl rechtsbündig, zehn Stellen nach dem Komma
  - Variable `y` (laufende Nummer 1) in `.25f`: unbestimmte Breite der Ausgabe gesamt, 25 Stellen nach dem Komma

### Format e

- ▶ Das Format `e` in Block 3 steht für das Exponentialformat.
  - Ein einfaches `e` bedeutet: Ausgabe mit einer Stelle vor dem Komma, sechs Stellen nach dem Komma, mit Exponent.

- Die Angabe 12.3e bedeutet: Breite der Ausgabe insgesamt 12 Stellen inklusive Exponent, Zahl rechtsbündig, eine Stelle vor dem Komma, drei Stellen nach dem Komma, mit Exponent.
  - Die Angabe .3e bedeutet: Breite der Ausgabe gesamt ist unbestimmt, drei Stellen nach dem Komma, mit Exponent.
- Das Format % in Block 4 steht für das Prozentformat. Die Zahl wird für die Ausgabe mit 100 multipliziert, intern bleibt die Zahl unverändert. Format %
- Ein einfaches % bedeutet: Ausgabe mit sechs Stellen nach dem Komma, gefolgt von einem Prozentzeichen.
  - Die Angabe 12.3% bedeutet: Breite der Ausgabe insgesamt 12 Stellen inklusive Prozentzeichen, Zahl rechtsbündig, 3 Stellen nach dem Komma, Prozentzeichen.
  - Die Angabe .3% bedeutet: Breite der Ausgabe insgesamt ist unbestimmt, drei Stellen nach dem Komma, mit Prozentzeichen.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

Es folgt ein Beispiel für die Ausgabe von ganzen Zahlen und Zeichenketten:

```
# Formatierung von Zeichenketten
print("{0:>4}{1:>9}{2:>4}{3:>4}".format
      ("dez", "dual", "okt", "hex"))
# Formatierung ganzer Zahlen
for z in range(59,69):
    print("{0:4d}{0:9b}{0:4o}{0:4x}".format(z))
print()

# Tabelle mit verschiedenen Objekten
fm = "{0:04d}{1:>12}{2:4d}{3:8.2f} Euro{4:8.2f} Euro"
artname = {23:"Apfel", 8:"Banane", 42:"Pfirsich"}
anzahl = {23:1, 8:3, 42:5}
epreis = {23:2.95, 8:1.45, 42:3.05}

print("{0:>4}{1:>12}{2:>4}{3:>13}{4:>13}".format
      ("Nr", "Name", "Anz", "EP", "GP"))
for x in 23, 8, 42:
    print(fm.format(x, artname[x], anzahl[x],
                    epreis[x], anzahl[x] * epreis[x]))
```

**Listing 5.9** Datei `ausgabe_tabelle.py`

Die Ausgabe sieht wie folgt aus:

dez	dual	okt	hex
59	111011	73	3b
60	111100	74	3c
61	111101	75	3d
62	111110	76	3e
63	111111	77	3f
64	1000000	100	40
65	1000001	101	41
66	1000010	102	42
67	1000011	103	43
68	1000100	104	44

Nr	Name	Anz	EP	GP
0023	Apfel	1	2.95 Euro	2.95 Euro
0008	Banane	3	1.45 Euro	4.35 Euro
0042	Pfirsich	5	3.05 Euro	15.25 Euro

Zur Erläuterung:

- ▶ Zeichenketten werden standardmäßig linksbündig ausgegeben. Zahlen werden standardmäßig rechtsbündig ausgegeben.

### Rechtsbündig >

- ▶ Als Erstes werden im Programm vier Zeichenketten formatiert ausgegeben. Nach der laufenden Nummer (0 bis 3) folgt das Zeichen > für rechtsbündig und die Gesamtbreite der Ausgabe (4 bzw. 9). Das Zeichen < würde für linksbündig stehen.

### Formate d, b, o, x

- ▶ Es folgt eine einheitlich formatierte Zahlentabelle. Die Zahlen von 59 bis 68 werden nacheinander ausgegeben als:
  - Dezimalzahl (Zeichen d, hier in Gesamtbreite 4)
  - Binärzahl bzw. Dualzahl (Zeichen b, hier in Gesamtbreite 9)
  - Oktalzahl (Zeichen o, hier in Gesamtbreite 4)
  - Hexadezimalzahl (Zeichen x, hier in Gesamtbreite 4)

### Mehrmals verwenden

- ▶ Zum Abschluss folgt noch eine kleine Artikeltabelle. Die Formatierungsangabe wird einheitlich festgelegt und als Zeichenkette gespeichert. Dies hat den Vorteil, dass sie mehrmals im Programm verwendet werden kann. Die Zeichenkette enthält Formate für ganze Zahlen, Zahlen mit Nachkommastellen und Zeichenketten sowie Text.
- ▶ Es werden drei Dictionarys angelegt: für Artikelname, Anzahl und Einzelpreis. Alle Elemente der Dictionarys werden einheitlich ausgegeben, zusammen mit dem ermittelten Gesamtpreis.

### Führende Nullen

- ▶ Steht vor der Angabe der Gesamtbreite eine Null, so wird die Zahl mit führenden Nullen bis zur angegebenen Gesamtbreite aufgefüllt. Dies ist bei der ersten Spalte der Fall.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### Übung u\_tabelle

Schreiben Sie das Programm aus der Übung *u\_range\_inch* so um, dass die folgende Ausgabe (mit Überschrift) erzeugt wird. Die Beträge für `inch` und `cm` sollen jeweils mit zwei Nachkommastellen angezeigt und rechtsbündig ausgerichtet werden (Datei *u\_tabelle.py*).

inch	cm
15.0	38.1
20.0	50.8
25.0	63.5
30.0	76.2
35.0	88.9
40.0	101.6

### 5.2.3 Formatierte Ausgabe wie in C

Sie können die gleiche Ausgabe wie mit der Funktion `format()` auch mithilfe einer Formatierung erzielen, die Sie so ähnlich möglicherweise aus der Programmiersprache C kennen. Sie wird ebenfalls häufig eingesetzt.

Es gibt die Formatangaben `%f`, `%e`, `%%`, `%d`, `%o` und `%x`, außerdem das Format `%s` für Zeichenketten. Nur das Format `b` für Dualzahlen gibt es nicht. Als Beispiel dienen die Programme aus Abschnitt 5.2.3, angepasst auf die Formatierung wie in C. Die Ausgaben der Programme stimmen überein, außer für die Dualzahlen.

Format s

Zunächst einige Zeilen aus der ersten Datei:

```
...
print("%15.10f %.25f" % (x, y))
...
print("%12.3e" % (x))
...
print("%12.3f%%" % (y*100))
...

```

**Listing 5.10** Datei `ausgabe_formatieren_c.py`, Ausschnitte

Die Formatangabe innerhalb von Anführungszeichen wird jeweils mithilfe eines Prozentzeichens eingeleitet. Es folgen die Formate wie in [Abschnitt 5.2.2](#), dann ein einzelnes Prozentzeichen. Als Letztes werden die Variablen in der auszugebenden Reihenfolge aufgelistet, innerhalb von runden Klammern.

- Prozentzeichen** Das Prozentzeichen selbst wird durch zwei aufeinanderfolgende Prozentzeichen ausgegeben. Die Multiplikation des Werts mit der Zahl 100 muss von Hand vorgenommen werden.

Es folgen einige Zeilen aus der zweiten Datei:

```
...
for z in range(59,69):
    print("%4d%4o%4x" % (z, z, z))
...
for x in 23, 8, 42:
    print("%4d%12s%4d%8.2f Euro%8.2f Euro" % (x, artname[x],
        anzahl[x], epreis[x], anzahl[x] * epreis[x]))
```

**Listing 5.11** Datei `ausgabe_tabelle_c.py`, Ausschnitte

- linksbündig** In der ersten Ausgabe fehlt das Formatzeichen für Dualzahlen. Zeichenketten werden mithilfe der Formatangabe `%s` ausgegeben. Das Format `%-12s` würde zu einer linksbündigen Ausgabe führen.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen `\` für den Umbruch von langen Programmzeilen eingesetzt. Bei der Division muss den Zahlen ein `.0` angehängt werden, damit das Ergebnis richtig berechnet wird.

## 5.3 Conditional Expression

- Kürzere Schreibweise** Eine *Conditional Expression* (d. h. ein bedingter Ausdruck) kann als Schreibabkürzung für eine einfache Verzweigung dienen. Sie ist allerdings etwas schwerer lesbar. Ein Programm mit zwei Beispielen:

```
x = -12
y = 15

# Ausdruck zur Zuweisung
max = x if x>y else y
print(max)

# Ausdruck zur Ausgabe
print("positiv" if x>0 else "negativ oder 0")
```

**Listing 5.12** Datei `conditional_expression.py`

Das Programm erzeugt die Ausgabe:

```
15
negativ oder 0
```

Zur Erläuterung:

- ▶ Die erste Anweisung mit dem bedingten Ausdruck liest sich wie folgt: Die Variable `max` erhält den Wert von `x`, falls `x` größer als `y` ist; anderenfalls erhält die Variable `max` den Wert von `y`.
- ▶ Die zweite Anweisung mit dem bedingten Ausdruck liest sich wie folgt: Gib die Zeichenkette `positiv` aus, falls `x` größer als 0 ist, ansonsten gib `negativ` oder `0` aus.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 5.4 Iterierbare Objekte

Eine Abfolge von Objekten, die z. B. in einer `for`-Schleife durchlaufen werden kann, nennt man auch *iterierbares Objekt* oder kurz *Iterable*. Listen, Zeichenketten, Tupel, Dictionarys und andere Objekte sind iterierbar. Sie können iterierbare Objekte unter anderem in `for`-Schleifen benutzen.

**Iterable**

Es gibt eine Reihe von Funktionen, die mit iterierbaren Objekten arbeiten und die dem Entwickler viel Arbeit abnehmen können. Als Beispiele erläutere ich im Folgenden die Funktionen `zip()`, `map()` und `filter()`.

### 5.4.1 Funktion »zip()«

Die Funktion `zip()` verbindet Elemente aus verschiedenen iterierbaren Objekten. Sie liefert wiederum einen Iterator, der aus den verbundenen Objekten besteht. Ein Beispiel:

**Iterables verbinden**

```
# Mehrere iterierbare Objekte
plz = [49808, 78224, 55411]
stadt = ["Lingen", "Singen", "Bingen"]
bundesland = ["NS", "BW", "RP"]

# Verbinden
kombi = zip(plz, stadt, bundesland)

# Ausgabe
for element in kombi:
    print(element)
```

**Listing 5.13** Datei `iterable_zip.py`

Die Ausgabe lautet:

```
(49808, 'Lingen', 'NS')
(78224, 'Singen', 'BW')
(55411, 'Bingen', 'RP')
```

Zur Erläuterung:

- ▶ Zunächst werden verschiedene iterierbare Objekte erzeugt – in diesem Fall drei Listen, die die Postleitzahlen, Namen und die entsprechenden Bundesländer dreier Städte enthalten.
- ▶ Die Funktion `zip()` erhält drei Parameter: die drei iterierbaren Objekte. In der Funktion werden sie miteinander verbunden.
- ▶ Es wird das Objekt `kombi` zurückgeliefert. Die Elemente des Objekts `kombi` sind (thematisch zusammengehörige) Tupel.
- ▶ Diese Tupel werden mithilfe einer `for`-Schleife ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 5.4.2 Funktion »map()«

### Funktionsaufrufe

Mithilfe der Funktion `map()` können Sie Funktionen mit einer Reihe von Werten aufrufen. Die Funktion liefert einen Iterator, der aus den Funktionsergebnissen besteht. Ein Programm mit zwei Beispielen:

```
# Funktion mit einem Parameter
def quad(x):
    erg = x * x
    return erg

# Funktion mit mehr als einem Parameter
def summe(a,b,c):
    erg = a + b + c
    return erg

# Funktion mit einem Parameter mehrmals aufrufen
z = map(quad, [4, 2.5, -1.5])

# Jedes Ergebnis ausgeben
print("Quadrat:")
for element in z:
    print(element)
print()

# Funktion mit mehr als einem Parameter mehrmals aufrufen
z = map(summe, [3, 1.2, 2], [4.8, 2], [5, 0.1, 9])
```

```
# Jedes Ergebnis ausgeben
print("Summe:")
for element in z:
    print(element)
```

**Listing 5.14** Datei iterable\_map.py

Die Ausgabe lautet:

**Quadrat:**

16

6.25

2.25

**Summe:**

12.8

3.3000000000000003

Zur Erläuterung:

- ▶ Zunächst werden zwei Funktionen definiert:
  - Die Funktion `quad()` hat einen Parameter und liefert das Quadrat dieses Werts zurück.
  - Die Funktion `summe()` hat drei Parameter und liefert die Summe dieser drei Werte zurück.
- ▶ Im ersten Beispiel wird die Funktion `map()` mit zwei Parametern aufgerufen: **Zwei Parameter**
  - Der erste Parameter ist der Name der Funktion, die für die verschiedenen Werte aufgerufen werden soll.
  - Der zweite Parameter ist ein iterierbares Objekt, in dem die Werte stehen, für die die Funktion aufgerufen werden soll.
  - Es wird das Objekt `z` zurückgeliefert. Die Elemente des Objekts `z` sind die Funktionsergebnisse, also die Rückgabewerte der Funktion für die verschiedenen Aufrufe.
  - Diese Ergebnisse werden mithilfe einer `for`-Schleife ausgegeben.
- ▶ Im zweiten Beispiel wird die Funktion `map()` mit mehr als zwei Parametern aufgerufen: **Mehr als zwei Parameter**
  - Der erste Parameter ist nach wie vor der Name der Funktion, die für die verschiedenen Werte aufgerufen werden soll.
  - Der zweite und alle folgenden Parameter sind jeweils iterierbare Objekte, in denen die Werte stehen, für die die Funktion aufgerufen werden soll.
  - Für die Bildung der ersten Summe wird aus jedem iterierbaren Objekt das erste Element (hier: 3, 4, 8 und 5) verwendet. Für die Bildung der zweiten Summe wird aus jedem iterierbaren Objekt das zweite Element (hier: 1, 2, 2 und 0,1) verwendet usw.

- Das kürzeste iterierbare Objekt (hier: [4, 8, 2]) bestimmt (in Python 3) die Anzahl der Aufrufe. Die Funktion wird also niemals mit zu wenigen Parametern aufgerufen.
- Es wird das Objekt `z` zurückgeliefert. Die Elemente des Objekts `z` sind die Funktionsergebnisse, also die Rückgabewerte der Funktion für die verschiedenen Aufrufe.
- Diese Ergebnisse werden mithilfe einer `for`-Schleife ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die iterierbaren Objekte müssen alle gleich lang sein, sonst kommt es zu einem Programmabbruch. Daher wurde in der Version für Python 2 das zweite Objekt um ein Element ergänzt.

### 5.4.3 Funktion »filter()«

**Iterable filtern** Die Funktion `filter()` untersucht Elemente eines iterierbaren Objekts mithilfe einer Funktion. Sie liefert diejenigen Elemente, für die die Funktion den Wahrheitswert `True` zurückliefert. Ein Beispiel:

```
# Funktion, die True oder False liefert
def test(a):
    if a>3:
        return True
    else:
        return False

# Funktion mehrmals aufrufen
z = filter(test, [5, 6, -2, 0, 12, 3, -5])

# Ausgabe der Werte, die True ergeben
for element in z:
    print("True:", element)
```

**Listing 5.15** Datei `iterable_filter.py`

Es wird die Ausgabe erzeugt:

```
True: 5
True: 6
True: 12
```

Zur Erläuterung:

- ▶ Der erste Parameter der Funktion `filter()` ist der Name der Funktion, die für einen untersuchten Wert `True` oder `False` liefert.

- Der zweite Parameter ist das iterierbare Objekt, in diesem Fall eine Liste.
- Es wird das iterierbare Objekt `z` zurückgeliefert. In dieser Liste stehen nur noch die Elemente, für die die Funktion `True` ergibt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 5.5 List Comprehension

Mithilfe von *List Comprehensions* erzeugen Sie auf einfache Art und Weise eine Liste aus einer anderen Liste. Dabei können Sie die Elemente der ersten Liste filtern und verändern.

[Liste aus Liste](#)

In insgesamt drei Beispielen wird die herkömmliche Technik der Technik der List Comprehension gegenübergestellt:

```
# Zwei Beispieldlisten
xliste = [3, 6, 8, 9, 15]
print(xliste)
yliste = [2, 13, 4, 8, 4]
print(yliste)
print()

# Beispiel 1: Version ohne List Comprehension
aliste = []
for item in xliste:
    aliste.append(item+1)
print(aliste)

# Beispiel 1: Version mit List Comprehension
aliste = [item + 1 for item in xliste]
print(aliste)
print()

# Beispiel 2: Version ohne List Comprehension
bliste = []
for item in xliste:
    if(item > 7):
        bliste.append(item + 1)
print(bliste)

# Beispiel 2: Version mit List Comprehension
bliste = [item + 1 for item in xliste if item > 7]
print(bliste)
print()
```

```
# Beispiel 3: Version ohne List Comprehension
cliste = []
for i in range(len(xliste)):
    if xliste[i] < 10 and yliste[i] < 10:
        cliste.append(xliste[i]*10 + yliste[i])
print(cliste)

# Beispiel 3: Version mit List Comprehension
cliste = [xliste[i]*10 + yliste[i]
          for i in range(len(xliste))
          if xliste[i] < 10 and yliste[i] < 10]
print(cliste)
```

**Listing 5.16** Datei `list_comprehension.py`

Die Ausgabe lautet:

[3, 6, 8, 9, 15]

[2, 13, 4, 8, 4]

[4, 7, 9, 10, 16]

[4, 7, 9, 10, 16]

[9, 10, 16]

[9, 10, 16]

[32, 84, 98]

[32, 84, 98]

Zur Erläuterung:

- ▶ Zunächst werden zwei Beispieldaten gebildet. Für die Nutzung innerhalb des dritten Beispiels ist es wichtig, dass sie gleich groß sind.
- ▶ In Beispiel 1 wird zunächst ohne List Comprehension gearbeitet. Es wird eine leere Liste erstellt. Innerhalb einer `for`-Schleife, die über jedes Element iteriert, wird die Ergebnisliste mithilfe der Funktion `append()` gefüllt.

### Alle Elemente

- ▶ Das Gleiche erreichen Sie auch in einem einzigen Schritt. Der Ausdruck `aliste = [item + 1 for item in xliste]` bedeutet: Liefere den Wert von `item+1` für jedes einzelne Element in `xliste`, dabei ist »`item`« der Name eines einzelnen Elements.

### Elemente filtern

- ▶ In Beispiel 2 sehen Sie, dass Sie eine Liste auch filtern können. Es werden nur die Elemente übernommen, deren Wert größer als 7 ist. Der Ausdruck `bliste = [item + 1 for item in xliste if item > 7]` bedeutet: Liefere den Wert von `item+1` für jedes einzelne Element in `xliste`, aber nur, wenn der Wert des einzelnen Elements größer als 7 ist.

### Mit Index

- ▶ Beispiel 3 zeigt, dass Sie natürlich auch eine `for`-Schleife mit `range` verwenden können. Die einzelnen Listenelemente werden über einen Index ange-

sprochen. Im Beispiel wird eine Liste aus zwei anderen, gleich langen Listen gebildet. Dabei wird eine Filterung vorgenommen.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 5.6 Fehler und Ausnahmen

Dieser Abschnitt bietet weitergehende Erläuterungen und Programmiertechniken im Zusammenhang mit Fehlern und Ausnahmen.

### 5.6.1 Allgemeines

Während Sie ein Programm entwickeln und testen, treten oft noch Fehler auf. Das ist völlig normal, und häufig können Sie aus diesen Fehlern hinzulernen. Fehler lassen sich in drei Gruppen untergliedern: Syntaxfehler, Laufzeitfehler und logische Fehler.

Fehler sind normal

Syntaxfehler bemerken Sie spätestens dann, wenn Sie ein Programm starten. Laufzeitfehler, also Fehler zur Laufzeit des Programms, die einen Programmabsturz zur Folge haben, können Sie mit einem `try-except`-Block behandeln.

Abbruch vermeiden

Logische Fehler treten auf, wenn das Programm richtig arbeitet, aber nicht die erwarteten Ergebnisse liefert. Hier hat der Entwickler den Ablauf nicht richtig durchdacht. Diese Fehler sind erfahrungsgemäß am schwersten zu finden. Dabei bietet das Debugging eine gute Hilfestellung.

Logische Fehler

### 5.6.2 Syntaxfehler

Syntaxfehler treten zur Entwicklungszeit des Programms auf und haben ihre Ursache in falsch oder unvollständig geschriebenem Programmcode. Spätestens beim Starten eines Programms macht Python auf Syntaxfehler aufmerksam. Der Programmierer erhält eine Meldung und einen Hinweis auf die Fehlerstelle. Das Programm wird nicht weiter ausgeführt. Ein Beispiel für einen fehlerhaften Code:

Falscher Code

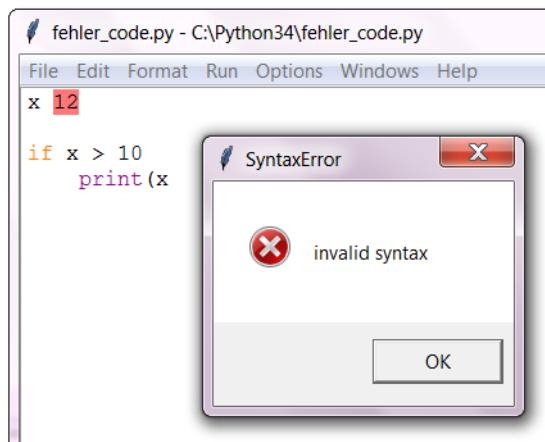
```
x 12
```

```
if x > 10
    print(x)
```

**Listing 5.17** Datei `fehler_code.py`

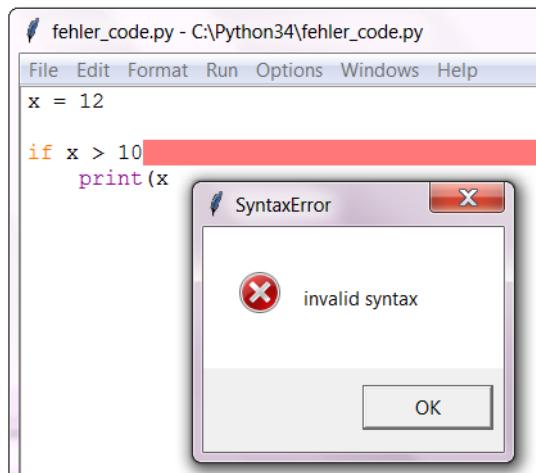
**Fehlerstelle  
markiert**

Nach dem Start des Programms erscheint die Fehlermeldung `invalid syntax`. Im Code wird die `12` markiert, da an dieser Stelle das Gleichheitszeichen erwartet wird, siehe [Abbildung 5.1](#). Das Programm läuft nicht weiter.



**Abbildung 5.1** Anzeige des ersten Fehlers

Nach der Verbesserung des Programms wird es erneut gestartet. Es erscheint die gleiche Fehlermeldung. Der Bereich nach `x > 10` wird markiert, da an dieser Stelle der Doppelpunkt erwartet wird, siehe [Abbildung 5.2](#).



**Abbildung 5.2** Anzeige des zweiten Fehlers

Nach erneuter Verbesserung des Programms wird es wieder gestartet. Es erscheint noch einmal die gleiche Fehlermeldung. Die Zeile mit `print(x)` wird markiert, da die schließende Klammer erwartet wird, siehe [Abbildung 5.3](#).

Erst nachdem auch der letzte Fehler beseitigt wurde, läuft das Programm fehlerfrei bis zum Ende.

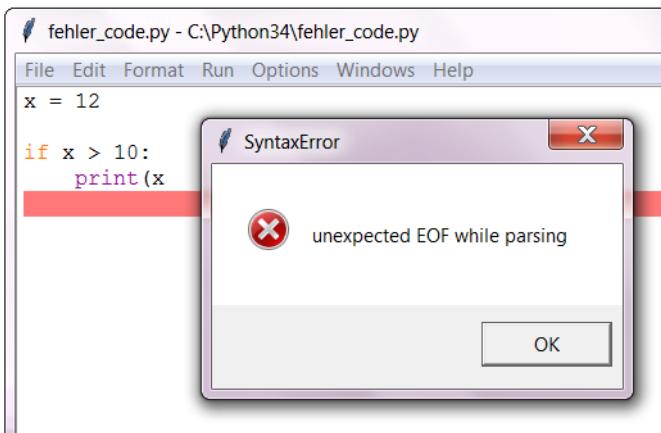


Abbildung 5.3 Anzeige des dritten Fehlers

### 5.6.3 Laufzeitfehler

Ein `try-except`-Block dient zum Abfangen von Laufzeitfehlern, wie bereits in Abschnitt 3.6, »Fehler und Ausnahmen«, angesprochen. Laufzeitfehler treten auf, wenn das Programm versucht, eine unzulässige Operation durchzuführen, beispielsweise eine Division durch 0 oder das Öffnen einer nicht vorhandenen Datei.

`try-except`

Natürlich wäre es besser, Laufzeitfehler von Anfang an zu unterbinden. Dies ist allerdings unmöglich, da es Vorgänge gibt, auf die der Entwickler keinen Einfluss hat, etwa die fehlerhafte Eingabe eines Benutzers oder ein beim Druckvorgang ausgeschalteter Drucker. Weitere Möglichkeiten zum Abfangen von Laufzeitfehlern werden in Abschnitt 5.6.6, »Unterscheidung von Ausnahmen«, erläutert.

Nicht vermeidbar

### 5.6.4 Logische Fehler und Debugging

Logische Fehler treten auf, wenn eine Anwendung zwar ohne Syntaxfehler übersetzt und ohne Laufzeitfehler ausgeführt wird, aber nicht das geplante Ergebnis liefert. Ursache hierfür ist ein fehlerhafter Aufbau der Programmlogik.

Die Ursache logischer Fehler zu finden, ist oft schwierig und erfordert intensives Testen und Analysieren der Abläufe und Ergebnisse. Die Entwicklungsumgebung IDLE stellt zu diesem Zweck einen einfachen Debugger zur Verfügung.

Debugging

### Einzelschrittverfahren

- Einzelne Schritte** Sie können ein Programm im Einzelschrittverfahren ablaufen lassen, um sich dann bei jedem einzelnen Schritt die aktuellen Inhalte von Variablen anzuschauen. Als Beispiel dient ein einfaches Programm mit einer Schleife und einer Funktion:

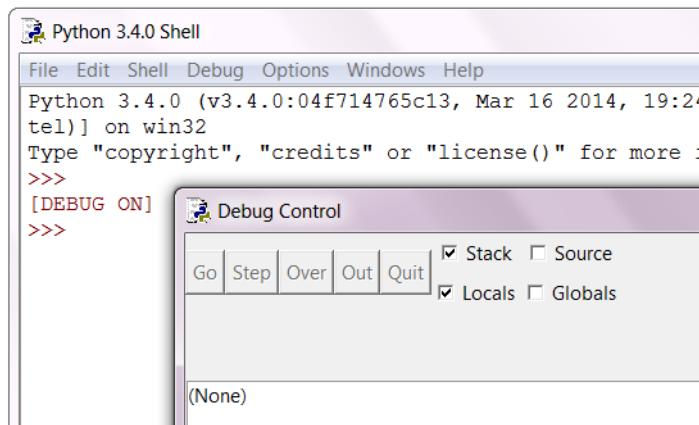
```
def summe(a,b):
    c = a + b
    return c

for i in range(5):
    erg = summe(10,i)
    print(erg)
```

**Listing 5.18** Datei fehler\_debuggen.py

Dieses Programm schreibt nacheinander die Zahlen von 10 bis 14 auf den Bildschirm.

- Debugger starten** Sie starten den Debugger, indem Sie in der PYTHON SHELL im Menü DEBUG den Menüpunkt DEBUGGER aufrufen. Es erscheint das Dialogfeld DEBUG CONTROL und in der PYTHON SHELL die Meldung [DEBUG ON], siehe Abbildung 5.4.



**Abbildung 5.4** Dialogfeld »Debug Control« und Meldung

Starten Sie nun das Programm wie gewohnt über den Menüpfad RUN • RUN MODULE oder über die Taste **F5**. Im Dialogfeld DEBUG CONTROL wird auf die erste Zeile des Programms hingewiesen, siehe Abbildung 5.5.

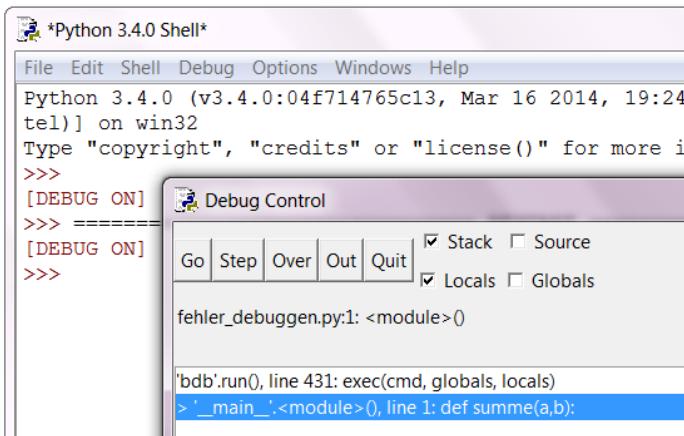


Abbildung 5.5 Nach dem Start des Programms

Jetzt können Sie auch die Buttons im Dialogfeld DEBUG CONTROL betätigen. Mit dem Button STEP gehen Sie schrittweise durch das Programm. Mit dem nächsten Schritt gelangen Sie direkt hinter die Funktionsdefinition zur ersten ausgeführten Programmzeile. Die Funktion wird erst beim Aufruf durchlaufen, siehe Abbildung 5.6.

Buttons bedienen

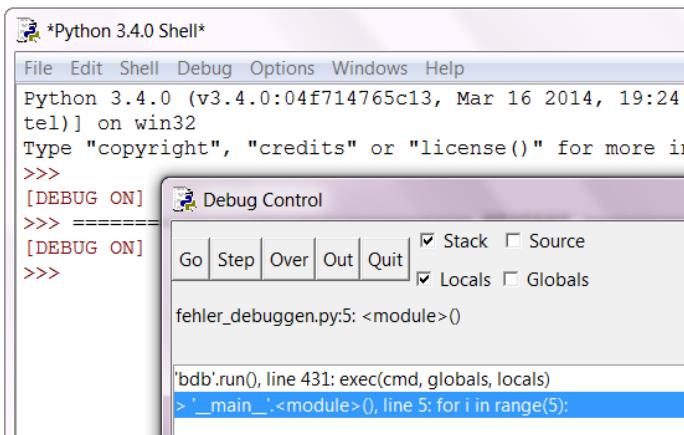


Abbildung 5.6 Erste ausgeführte Programmzeile

Durch wiederholtes Drücken des Buttons STEP können Sie nun die Schleife mehrmals durchlaufen. Bei jedem Durchlauf wird auch die Funktion `summe()` durchlaufen. Im unteren Bereich des Dialogfelds DEBUG CONTROL sehen Sie die jeweils gültigen Variablen und ihre sich ständig verändernden Werte. Befinden Sie sich im Hauptprogramm in der Schleife, so sehen Sie die Werte von `i` und `erg`, siehe Abbildung 5.7.

Aktuelle Werte

Locals	
<code>_builtins_</code>	<module 'builtins' (built-in)>
<code>_doc_</code>	None
<code>_file_</code>	'C:\\\\Python34\\\\fehler_debuggen.py'
<code>_loader_</code>	<class '_frozen_importlib.BuiltinImporter'>
<code>_name_</code>	'__main__'
<code>_package_</code>	None
<code>_spec_</code>	None
<code>erg</code>	10
<code>i</code>	0
<code>summe</code>	<function summe at 0x023E55D0>

Abbildung 5.7 Hauptprogramm: aktuelle Werte von i und erg

Wenn Sie sich in der Funktion `summe()` befinden, sehen Sie die Werte von a, b, und c (siehe Abbildung 5.8).

a	10
b	2
c	12

Abbildung 5.8 Funktion: aktuelle Werte von a, b und c

### Ergebnisse parallel

In der Python Shell werden parallel dazu die ersten Ergebnisse ausgegeben, siehe Abbildung 5.9.

>>>
[DEBUG ON]
>>> =====
[DEBUG ON]
>>>
10
11

Abbildung 5.9 Python Shell: Ausgabe der ersten Ergebnisse

### Debug On

Nach Durchlauf der letzten Programmzeile wird noch der Hinweis [Debug On] in der Python Shell ausgegeben. IDLE befindet sich nach wie vor im Debug-Modus, aber die Buttons können Sie erst nach einem erneuten Programmstart wieder betätigen.

## Weitere Möglichkeiten

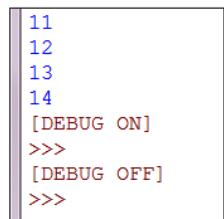
Um das Programm in etwas größeren Schritten zu durchlaufen, betätigen Sie den Button OVER. Die Funktionen werden dann nicht in Einzelschritten, sondern als Ganzes durchlaufen. Der Debugger springt also über die Funktionen hinweg.

Sie können auch zwischen den beiden Möglichkeiten (Button STEP und Button OVER) flexibel hin und her wechseln – je nachdem, welchen Programmteil Sie sich ganz genau ansehen möchten.

Wenn Sie sich gerade in Einzelschritten durch eine Funktion bewegen, führt die Betätigung des Buttons OUT dazu, dass der Rest der Funktion übersprungen und mit dem ersten Schritt nach dem Funktionsaufruf fortgefahren wird.

Der Button Go lässt das Programm, das Sie gerade debuggen, in einem Schritt bis zum Ende laufen. Der Button QUIT bricht den Lauf des Programms sofort ab, ohne es zu Ende laufen zu lassen. In beiden Fällen ist der Debug-Modus noch eingeschaltet.

Der Debug-Modus lässt sich an derselben Stelle ausschalten, an der Sie ihn eingeschaltet haben: in der Python Shell im Menü DEBUG • DEBUGGER. In der Shell erscheint dann die Meldung [DEBUG OFF], siehe Abbildung 5.10.



```

11
12
13
14
[DEBUG ON]
>>>
[DEBUG OFF]
>>>

```

Abbildung 5.10 Nach dem Beenden des Debuggers

Auf weniger elegante Weise können Sie den Debugger beenden, indem Sie das Dialogfeld DEBUG CONTROL einfach schließen.

Andere Entwicklungsumgebungen für Python bieten weitere Möglichkeiten. Das Setzen von *Breakpoints* (deutsch: Haltepunkte) ist sehr nützlich. Diese Haltepunkte werden auf bestimmte Programmzeilen gesetzt. Das Programm lassen Sie dann bis zu einer solchen Programmzeile in einem Schritt durchlaufen und überprüfen die aktuellen Werte. Anschließend durchlaufen Sie entweder im Einzelschrittverfahren einen Programmabschnitt, in dem Sie Fehler vermuten, oder gehen direkt zum nächsten, vorher gesetzten Haltepunkt usw.

Funktion  
überspringen

Funktion verlassen

Programm beenden

Debug Off

Haltepunkte

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 5.6.5 Fehler erzeugen

»Wieso sollte man Fehler erzeugen?«, werden Sie sich angesichts dieser Überschrift fragen. Hierfür gibt es, besonders im Zusammenhang mit der Eingabe von Daten durch einen Anwender, durchaus sinnvolle Gründe.

- raise** Im folgenden Beispiel wird der Anwender dazu aufgefordert, eine Zahl einzugeben, deren Kehrwert anschließend berechnet wird. Diese Zahl soll allerdings positiv sein. Diese Einschränkung kann mithilfe der Anweisung `raise` bearbeitet werden, wie im folgenden Programm dargestellt:

```
# Wiederholte Eingabe
fehler = True
while fehler:
    try:
        zahl = float(input("Eine positive Zahl: "))
        if zahl < 0:
            raise
        kw = 1.0 / zahl
        fehler = False
    except:
        print("Fehler")

# Ausgabe
print("Der Kehrwert von", zahl, "ist", kw)
```

**Listing 5.19** Datei `fehler_erzeugen.py`

- Ausnahme erzeugen** Ist die eingegebene Zahl kleiner als 0, so wird die Anweisung `raise` ausgeführt. Dadurch wird eine Ausnahme erzeugt, so als ob der Anwender einen der anderen möglichen Fehler gemacht hätte. Das Programm verzweigt unmittelbar zur Anweisung `except` und führt die dort angegebenen Anweisungen aus.

In diesem Fall handelt es sich zwar nur um einen logischen Eingabefehler, aber er wird genauso behandelt wie ein Fehler im Programm. Der Anwender wird somit veranlasst, nur positive Zahlen einzugeben. Das folgende Listing zeigt eine mögliche Eingabe, zunächst dreimal mit Fehler, anschließend richtig:

```
Eine positive Zahl: 0
Fehler
Eine positive Zahl: abc
Fehler
Eine positive Zahl: -6
Fehler
Eine positive Zahl: 6
Der Kehrwert von 6.0 ist 0.1666666666666666
```

Der Benutzer macht verschiedene Fehler:

- ▶ Er gibt die Zahl 0 ein. Dies führt bei der Berechnung des Kehrwerts zu einem Laufzeitfehler, einem `ZeroDivisionError`.
- ▶ Er gibt einen Text ein. Dies führt beim Aufruf der Funktion `float()` zu einem Laufzeitfehler, einem `ValueError`.
- ▶ Er gibt eine negative Zahl ein. Dies führt wegen der vorgenommenen Einschränkung zu einem Laufzeitfehler.

Mithilfe der Anweisungen `try`, `raise` und `except` lassen sich also auch nicht sinnvolle Eingaben des Anwenders abfangen und behandeln. Die vorgeführte Methode hat den Nachteil, dass alle Fehler gleich behandelt werden und die Informationen für den Anwender im Fehlerfall noch nicht sehr genau sind. Dies soll im nächsten Abschnitt verbessert werden.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

## 5.6.6 Unterscheidung von Ausnahmen

Im folgenden Programm sollen verschiedene Arten von Fehlern spezifisch abgefangen werden. Damit informieren Sie beispielsweise einen Benutzer genauer über seinen Irrtum und ermöglichen eine komfortablere Programmbedienung. Es soll wiederum der Kehrwert einer eingegebenen Zahl ermittelt werden:

```
# Wiederholte Eingabe
fehler = True
while fehler:
    try:
        zahl = float(input("Eine positive Zahl: "))
        if zahl == 0:
            raise RuntimeError("Zahl gleich 0")
        if zahl < 0:
            raise RuntimeError("Zahl zu klein")
        kw = 1.0 / zahl
        fehler = False
    except ValueError:
        print("Fehler: keine Zahl")
    except ZeroDivisionError:
        print("Fehler: Zahl 0 eingegeben")
    except RuntimeError as e:
        print("Fehler:", e)

# Ausgabe
print("Der Kehrwert von", zahl, "ist", kw)
```

**Listing 5.20** Datei `fehler_unterscheiden.py`

Verschiedene Ausnahmen

Das Programm enthält zu einem Versuch (Anweisung `try`) mehrere spezifische Auffangmöglichkeiten (Anweisung `except`). Nachfolgend wird eine mögliche Eingabe gezeigt – zunächst dreimal mit Fehler, anschließend richtig:

```
Eine positive Zahl: 0
Fehler: Zahl gleich 0
Eine positive Zahl: abc
Fehler: keine Zahl
Eine positive Zahl: -6
Fehler: Zahl zu klein
Eine positive Zahl: 6
Der Kehrwert von 6.0 ist 0.166666666667
```

Der Benutzer macht verschiedene Fehler:

- |                          |   |
|--------------------------|---|
| <b>Eigener Fehler</b>    | ► Er gibt die Zahl 0 ein. Dies führt wegen der vorgenommenen Einschränkung zu einem Laufzeitfehler. Dieser wird als allgemeiner <code>RuntimeError</code> abgefangen mit der Meldung <code>Fehler: Zahl gleich 0</code> .   |
| <b>ZeroDivisionError</b> | ► Würde die Eingabe von 0 nicht auf diese Weise abgefangen, käme es später bei der Berechnung des Kehrwerts zu einem Laufzeitfehler, einem <code>ZeroDivisionError</code> . Dieser würde abgefangen mit der Meldung <code>Fehler: Zahl 0 eingegeben</code> . Zu Demonstrationszwecken wurde der Fehler zweimal abgefangen, dies ist eigentlich nicht notwendig. |
| <b>ValueError</b>        | ► Der Benutzer gibt einen Text ein. Dies führt beim Aufruf der Funktion <code>float()</code> zu einem Laufzeitfehler, einem <code>ValueError</code> . Dieser wird abgefangen mit der Meldung <code>Fehler: keine Zahl</code> .  |
| <b>Eigener Fehler</b>    | ► Er gibt eine negative Zahl ein. Dies führt wegen der zweiten Einschränkung wiederum zu einem Laufzeitfehler. Dieser wird auch als allgemeiner <code>RuntimeError</code> abgefangen mit der Meldung <code>Fehler: Zahl zu klein</code> .   |
| <b>as</b>                | ► Beim Erzeugen des Fehlers mit der Anweisung <code>raise</code> werden ein Fehler ( <code>RuntimeError</code> ) und eine Meldung übergeben.  |

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

## 5.7 Funktionen

Python bietet zum Thema »Funktionen« noch einige sehr nützliche Erweiterungen, die in diesem Abschnitt erläutert werden.

### 5.7.1 Variable Anzahl von Parametern

In den bisherigen Beispielen wurde darauf geachtet, dass Reihenfolge und Anzahl der Funktionsparameter bei Definition und Aufruf miteinander übereinstimmen. Sie können aber auch Funktionen mit einer variablen Anzahl von Parametern definieren.

Bei der Definition einer solchen Funktion müssen Sie vor dem letzten (gegebenenfalls einzigen) Parameter einen \* (Stern) notieren. Dieser Parameter enthält ein Tupel mit den bis dahin nicht zugeordneten Werten der Parameterkette.

Parameter mit \*

Im folgenden Beispiel wird eine Funktion definiert, die in der Lage ist, die Summe aller Parameter zu berechnen und auszugeben. Dies gilt unabhängig von der Anzahl der zu summierenden Werte.

```
# Funktion
def summe(*summanden):
    print(len(summanden), "Zahlen")
    print(summanden)
    erg = 0
    for s in summanden:
        erg += s
    print("Summe:", erg)

# Aufrufe
summe(3, 4)
summe(3, 8, 12, -5)
```

**Listing 5.21** Datei parameter\_variabel.py

Folgende Ausgabe wird erzeugt:

```
2 Zahlen
(3, 4)
Summe: 7
4 Zahlen
(3, 8, 12, -5)
Summe: 18
```

Zur Erläuterung:

- ▶ Die (nunmehr flexiblere) Funktion wird mit zwei oder mit vier Werten aufgerufen, die zu summieren sind.
- ▶ Zu Demonstrationszwecken werden Größe und Inhalt des Tupels ausgegeben.
- ▶ Die for-Schleife dient zur Summierung der Werte des Tupels. Die Summe wird anschließend ausgegeben.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 5.7.2 Benannte Parameter

Die definierte Reihenfolge der Parameter muss beim Aufruf nicht eingehalten werden, falls Parameter mit ihrem Namen übergeben werden (benannte Parameter).

#### Parameter mit Namen

Im folgenden Beispiel sind einige Varianten zum Aufruf der Funktion `volumen()` dargestellt. Diese Funktion berechnet das Volumen eines Quaders und gibt dieses aus. Außerdem wird die übergebene Farbe ausgegeben.

```
# Funktion
def volumen(breite, laenge, tiefe, farbe):
    print("Werte:", breite, laenge, tiefe, farbe)
    erg = breite * laenge * tiefe
    print("Volumen:", erg, "Farbe:", farbe)

# Aufrufe
volumen(4, 6, 2, "rot")
volumen(laenge=2, farbe="gelb", tiefe=7, breite=3)
volumen(5, tiefe=2, laenge=8, farbe="blau")

# Fehler
# volumen(3, tiefe=4, laenge=5, "schwarz")
```

**Listing 5.22** Datei `parameter_benannt.py`

Die Ausgabe lautet:

```
Werte: 4 6 2 rot
Volumen: 48 Farbe: rot
Werte: 3 2 7 gelb
Volumen: 42 Farbe: gelb
Werte: 5 8 2 blau
Volumen: 80 Farbe: blau
```

Zur Erläuterung:

- ▶ Der erste Aufruf findet in der bekannten Form, ohne Benennung von Parametern, statt. Die Werte werden den Parametern in der übergebenen Reihenfolge zugeordnet.
- ▶ Beim zweiten Aufruf werden alle vier Parameter mit Namen übergeben. Die Reihenfolge beim Aufruf ist nicht wichtig, da die Parameter eindeutig zugeordnet werden.

#### Reihenfolge unwichtig

- Beim dritten Aufruf wird der erste Parameter über seine Stellung in der Parameterreihe zugeordnet, die anderen Parameter werden über ihre Namen zugeordnet. Es sind also auch Mischformen möglich.
- Sobald allerdings der erste benannte Parameter beim Aufruf erscheint, müssen alle folgenden Parameter auch benannt sein. Daher würde beim vierten Aufruf ein Fehler bereits in der Syntax erkannt, obwohl die Zuordnung eigentlich eindeutig wäre.

Einmal benannt,  
immer benannt

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 5.7.3 Voreinstellung von Parametern

Durch die Voreinstellung von Parametern bei der Definition wird eine variable Parameterzahl ermöglicht. Auch hier können Sie benannte Parameter einsetzen. Dabei ist es wichtig, dass nicht benannte Parameter vor benannten Parametern stehen.

Die Funktion zur Volumenberechnung des Quaders wurde in dieser Hinsicht geändert. Es müssen nur noch zwei Parameter angegeben werden. Die anderen beiden Parameter sind optional, es werden gegebenenfalls Voreinstellungen verwendet.

Parameter mit  
Standardwert

```
# Funktion
def volumen(breite, laenge, tiefe=1, farbe="schwarz"):
    print("Werte:", breite, laenge, tiefe, farbe)
    erg = breite * laenge * tiefe
    print("Volumen:", erg, "Farbe:", farbe)

# Aufrufe
volumen(4, 6, 2, "rot")
volumen(2, 12, 7)
volumen(5, 8)
volumen(4, 7, farbe="rot")
```

**Listing 5.23** Datei `parameter_voreinstellung.py`

Das Programm erzeugt die Ausgabe:

```
Werte: 4 6 2 rot
Volumen: 48 Farbe: rot
Werte: 2 12 7 schwarz
Volumen: 168 Farbe: schwarz
Werte: 5 8 1 schwarz
Volumen: 40 Farbe: schwarz
```

Werte: 4 7 1 rot

Volumen: 28 Farbe: rot

Zur Erläuterung:

- ▶ Bei der Funktionsdefinition wurden die beiden Parameter `tiefe` und `farbe` voreingestellt. Sie sind optional und müssen am Ende der Parameterreihe stehen.
- ▶ Beim ersten Aufruf werden alle vier Parameter gesendet.
- ▶ Beim zweiten Aufruf wird nur einer der beiden optionalen Parameter (`tiefe`) gesendet. Der zweite optionale Parameter erhält deshalb den voreingestellten Wert (`farbe = "schwarz"`).
- ▶ Beim dritten Aufruf wird keiner der optionalen Parameter gesendet. Daher erhalten beide den voreingestellten Wert.
- ▶ Beim vierten Aufruf wird nur der letzte optionale Parameter gesendet. Da dieser Parameter nicht mehr über die Reihenfolge zugeordnet werden kann, muss er benannt werden.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 5.7.4 Mehrere Rückgabewerte

Im Unterschied zu vielen anderen Programmiersprachen können Funktionen in Python mehr als einen Rückgabewert liefern. Es kann z. B. ein Tupel oder eine Liste zurückgeliefert werden.

##### Ergebnistupel

Im folgenden Beispiel werden in der Funktion `geom()` die Fläche und der Umfang eines Rechtecks berechnet und als Tupel zurückgegeben.

```
import math

# Funktion, die zwei Werte berechnet
def kreis(radius):
    flaeche = math.pi * radius * radius
    umfang = 2 * math.pi * radius
    return flaeche, umfang

# 1. Aufruf
f, u = kreis(3)
print("Flaeche:", f)
print("Umfang:", u)

# 2. Aufruf
x = kreis(3)
```

```
print("Flaeche:", x[0])
print("Umfang:", x[1])
```

```
# Fehler
# a, b, c = kreis(3)
```

**Listing 5.24** Datei rueckgabe\_tupel.py

Die Ausgabe lautet:

```
Flaeche: 28.274333882308138
Umfang: 18.84955592153876
Flaeche: 28.274333882308138
Umfang: 18.84955592153876
```

Zur Erläuterung:

- ▶ Die Anweisung `return` liefert ein Tupel mit den beiden Ergebnissen der Funktion. An der Aufrufstelle muss ein Tupel passender Größe zum Empfang bereitstehen.
- ▶ Im ersten Fall wird das Tupel zwei einzelnen Variablen zugeordnet.
- ▶ Im zweiten Fall wird das Tupel einer Variablen zugeordnet, die damit zum Tupel wird.
- ▶ Der letzte Aufruf würde zu einem Laufzeitfehler führen, da die Größe des zurückgelieferten Tupels nicht passt.

`return <Tupel>`

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 5.7.5 Übergabe von Kopien und Referenzen

Werden Parameter, die an eine Funktion übergeben werden, innerhalb der Funktion verändert, so wirkt sich dies im aufrufenden Programmteil unterschiedlich aus:

- ▶ Bei der Übergabe eines einfachen Objekts (Zahl oder Zeichenkette) wird eine Kopie des Objekts angelegt. Eine Veränderung der Kopie hat keine Auswirkungen auf das Original. Kopie übergeben
- ▶ Bei der Übergabe eines Objekts vom Typ Liste, Dictionary oder Set wird mit einer Referenz auf das Originalobjekt gearbeitet. Eine Veränderung über die Referenz verändert auch das Original. Referenz übergeben

Zur Verdeutlichung dieses Zusammenhangs werden im folgenden Beispiel insgesamt fünf Parameter an eine Funktion übergeben: eine Zahl, eine Zeichenkette, eine Liste, ein Dictionary und ein Set. Die Objekte werden jeweils dreimal ausgegeben:

**Vorher, nachher**

- ▶ vor dem Aufruf der Funktion
- ▶ nach einer Veränderung innerhalb der Funktion
- ▶ nach der Rückkehr aus der Funktion

```
# Funktion
def chg(v, zk, li, di, st):
    v = 8
    zk = "ciao"
    li[0] = 7
    di["x"] = 7
    st.discard(3)

    # lokale Ausgabe
    print("In Funktion:")
    print(v, zk)
    print(li, di, st)

# Startwerte
hv = 3
hli = [3, "abc"]
hzk = "hallo"
hdi = {"x":3, "y":"abc"}
hst = set([3, "abc"])

# Ausgabe vorher
print("vorher:")
print(hv, hzk)
print(hli, hdi, hst)

# Aufruf der Funktion
chg(hv, hzk, hli, hdi, hst)

# Ausgabe nachher
print("nachher:")
print(hv, hzk)
print(hli, hdi, hst)
```

**Listing 5.25** Datei parameter\_uebergabe.py

Die Ausgabe lautet:

```
vorher:
3 hallo
[3, 'abc'] {'y': 'abc', 'x': 3} {3, 'abc'}
In Funktion:
8 ciao
[7, 'abc'] {'y': 'abc', 'x': 7} {'abc'}
nachher:
3 hallo
[7, 'abc'] {'y': 'abc', 'x': 7} {'abc'}
```

Zur Erläuterung:

- ▶ Es zeigt sich, dass nur bei Liste, Dictionary und Set eine dauerhafte Veränderung durch die Funktion erfolgte. Dies ist je nach Problemstellung ein erwünschter oder ein unerwünschter Effekt.

(Un)erwünschter  
Effekt

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Ausgabe des Sets lautet `set([...])` statt `{...}`.

Ist die dauerhafte Veränderung von einfachen Objekten durch eine Funktion erwünscht, so können Sie die Tatsache ausnutzen, dass Python-Funktionen mehr als einen Rückgabewert haben können. Im folgenden Beispiel dient eine Funktion zum Sortieren von zwei Variablen. Die beiden Variablen werden als Tupel zurückgeliefert:

```
# Sortierfunktion
def sortieren(eins, zwei):
    if eins < zwei:
        return zwei, eins
    else:
        return eins, zwei

# Beispiel 1
x = 24
y = 29
x, y = sortieren(x, y)
print("x =", x, "y =", y)

# Beispiel 2
x = 124
y = 29
x, y = sortieren(x, y)
print("x =", x, "y =", y)
```

**Listing 5.26** Datei `werte_veraendern.py`

Die Ausgabe lautet:

```
x = 29 y = 24
x = 124 y = 29
```

Zur Erläuterung:

- ▶ Es werden zwei Variablen an die Funktion übergeben. Innerhalb der Funktion wird überprüft, ob die zweite Variable größer als die erste Variable ist.

- ▶ Trifft dies zu, so werden beide Variablen in umgekehrter Reihenfolge an die aufrufende Stelle zurückgeliefert.
- ▶ Falls nicht, stehen die beiden Variablen bereits in der gewünschten Reihenfolge und werden unverändert an die aufrufende Stelle zurückgeliefert.
- ▶ An der Ausgabe erkennen Sie, dass `x` nachher in jedem Fall die größere Zahl enthält, also gegebenenfalls verändert wurde.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 5.7.6 Lokal, global

**Lokal** Die Definition einer Funktion in Python erzeugt einen lokalen Namensraum. In diesem lokalen Namensraum stehen alle Namen der Variablen, denen innerhalb der Funktion ein Wert zugewiesen wird, und die Namen der Variablen aus der Parameterliste.

**Global** Wird bei der Ausführung der Funktion auf eine Variable zugegriffen, so wird diese Variable zunächst im lokalen Namensraum gesucht. Wird der Name der Variablen im lokalen Namensraum nicht gefunden, so wird im bisher bekannten globalen Namensraum gesucht, das heißt in den bisher bearbeiteten Programmzeilen außerhalb der Funktion. Sollte die Variable im globalen Namensraum nicht gefunden werden, so tritt ein Fehler auf. Ein Beispiel:

```
# Testfunktion
def func():
    try:
        print(x)
    except:
        print("Fehler")

# Hauptprogramm
func()
x = 42
func()
```

**Listing 5.27** Datei namensraum.py

Die Ausgabe lautet:

**Fehler**  
**42**

Zur Erläuterung:

- ▶ Der erste Aufruf von `func()` führt zu einem Fehler, da in der Funktion der Wert von `x` ausgegeben werden soll. `x` ist nicht im lokalen Namensraum vorhanden, aber auch nicht in den bisher bearbeiteten Programmzeilen außerhalb der Funktion.
  - ▶ Der zweite Aufruf von `func()` führt zu keinem Fehler, denn `x` hat vorher außerhalb der Funktion einen Wert erhalten und ist somit im globalen Namensraum bekannt.
- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>▶ Der erste Aufruf von <code>func()</code> führt zu einem Fehler, da in der Funktion der Wert von <code>x</code> ausgegeben werden soll. <code>x</code> ist nicht im lokalen Namensraum vorhanden, aber auch nicht in den bisher bearbeiteten Programmzeilen außerhalb der Funktion.</li> <li>▶ Der zweite Aufruf von <code>func()</code> führt zu keinem Fehler, denn <code>x</code> hat vorher außerhalb der Funktion einen Wert erhalten und ist somit im globalen Namensraum bekannt.</li> </ul> | <b>Variable noch unbekannt</b><br><br><b>Variable global bekannt</b> |
|---|--|

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 5.7.7 Lambda-Funktion

Die Lambda-Funktion bietet die Möglichkeit, eine Funktionsdefinition zu verkürzen. Einer solchen Funktion können Sie Parameter übergeben. Sie liefert ihr Ergebnis als einen Ausdruck zurück, der in der gleichen Zeile stehen muss. In diesem Ausdruck dürfen keine Mehrfachanweisungen, Ausgaben oder Schleifen vorkommen. Ein Beispielprogramm sieht wie folgt aus:

```
# Funktionsdefinitionen
mal = lambda x,y: x*y
plus = lambda x,y: x+y

# Funktionsaufrufe
print(mal(5,3))
print(plus(4,7))
```

**Listing 5.28** Datei `funktion_lambda.py`

Es wird die Ausgabe erzeugt:

```
15
11
```

Zur Erläuterung:

- ▶ Es wird die Funktion `mal` definiert. Diese hat zwei Parameter (`x` und `y`). Das Ergebnis der Funktion ist die Multiplikation dieser beiden Parameter.
- ▶ Die Lambda-Funktion `plus` ist analog nach folgendem Muster aufgebaut:  
`Ergebnis = lambda Parameterliste: Ausdruck.`

Eine Lambda-Funktion ermöglicht es Ihnen auch, eine Funktion mit Parametern an einer Stelle zu übergeben, an der eigentlich nur der Name einer Funktion übergeben werden darf. Dazu mehr in [Abschnitt 11.3.2](#), »Ein einfacher Taschenrechner«.

Kürzere  
Schreibweise

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 5.7.8 Funktionsname als Parameter

Sie können einer Funktion nicht nur Werte übergeben, sondern auch den Namen einer anderen Funktion. Dies macht die erstgenannte Funktion flexibler. Ein Beispiel:

```
# Funktion zur Berechnung einzelner Werte
def quadrat(x):
    return x * x

# Funktion zur Berechnung einzelner Werte
def hochdrei(x):
    return x * x * x;

# Funktion zur Ausgabe von Funktionswerten
def ausgabe(unten, oben, schritt, f):
    for x in range(unten, oben, schritt):
        print(x, f(x))
    print()

# Aufruf, Funktionsname ist Parameter
ausgabe(2, 11, 2, quadrat)
ausgabe(1, 6, 1, hochdrei)
```

**Listing 5.29** Datei `parameter_funktion.py`

Es wird die Ausgabe erzeugt:

```
2 4
4 16
6 36
8 64
10 100
```

```
11
2 8
3 27
4 64
5 125
```

Zur Erläuterung:

- ▶ Zunächst werden die beiden herkömmlichen Funktionen `quadrat()` und `hochdrei()` definiert. Diese liefern jeweils als Ergebnis einen einzelnen Funktionswert zurück.

- Die Funktion `ausgabe()` dient dazu, eine zweispaltige Tabelle auszugeben, die aus mehreren Werten und den zugehörigen Funktionswerten besteht. Sie erwartet insgesamt vier Parameter. Die ersten drei Parameter dienen zur Steuerung der `for`-Schleife.
- Der vierte Parameter gibt den Namen der Funktion an, die zur Berechnung des Werts verwendet werden soll, der in der zweiten Spalte der Tabelle ausgegeben wird. Der Name wird ohne Klammern angegeben. Im ersten Fall ist das die Funktion `quadrat()`, im zweiten Fall die Funktion `hochdrei()`.

Ohne Klammern

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 5.8 Eingebaute Funktionen

Als Entwickler können Sie eine Reihe von eingebauten Funktionen ohne Einbindung eines Moduls verwenden.

Tabelle 5.1 gibt eine Übersicht über die eingebauten Funktionen, die in diesem Buch behandelt werden.

Name	Beschreibung	Beispiel in Abschnitt
<code>abs()</code>	Liefert den Betrag einer Zahl.	4.1.6
<code>bin()</code>	Liefert eine binäre bzw. duale Zahl.	4.1.1
<code>bytes()</code>	Liefert ein Objekt des Datentyps <code>bytes</code> .	4.2.7
<code>chr()</code>	Liefert ein Zeichen zu einer Unicode-Zahl.	5.8.3
<code>eval()</code>	Liefert einen ausgeführten Python-Ausdruck.	5.1.5
<code>exec()</code>	Führt eine Anweisung aus.	5.1.5
<code>filter()</code>	Liefert die Elemente eines iterierbaren Objekts, für die eine Funktion <code>True</code> ergibt.	5.4.3
<code>float()</code>	Liefert eine Zahl mit Nachkommastellen.	3.2.3
<code>format()</code>	Formatiert Zahlen und Zeichenketten.	5.2.2
<code>frozenset()</code>	Liefert ein unveränderliches Set.	4.6.4

**Tabelle 5.1** Einige eingebaute Funktionen

Name	Beschreibung	Beispiel in Abschnitt
hex()	Liefert eine hexadezimale Zahl.	4.1.1
input()	Wartet auf eine Eingabe des Benutzers.	3.2.2
int()	Liefert eine ganze Zahl.	3.2.3
len()	Liefert die Anzahl der Elemente.	4.2.3
map()	Liefert Funktionsergebnisse zu einer Reihe von Aufrufen.	5.4.2
max()	Liefert das größte Element.	5.8.2
min()	Liefert das kleinste Element.	5.8.2
oct()	Liefert eine oktale Zahl.	4.1.1
open()	Öffnet eine Datei zum Lesen oder Schreiben.	8.2
ord()	Liefert die Unicode-Zahl zu einem Zeichen.	5.8.3
print()	Erzeugt eine Ausgabe.	5.2.1
range()	Liefert ein iterierbares Objekt über einen Bereich.	3.4.5
reversed()	Liefert ein iterierbares Objekt in umgekehrter Reihenfolge.	5.8.4
repr()	Liefert Informationen über ein Objekt.	6.4
round()	Liefert eine gerundete Zahl.	4.1.4
set()	Liefert ein Set.	4.6
sorted()	Liefert eine sortierte Liste.	5.8.4
str()	Liefert eine Zeichenkette.	4.2.6
sum()	Liefert die Summe der Elemente.	5.8.2
type()	Liefert den Typ eines Objekts.	4.1.5
zip()	Verbindet Elemente aus iterierbaren Objekten.	5.4.1

Tabelle 5.1 Einige eingebaute Funktionen (Forts.)

### 5.8.1 Funktionen »max()«, »min()« und »sum()«

Die Funktionen `max()` und `min()` liefern den größten und den kleinsten Wert, falls nur ein iterierbares Objekt angegeben ist. Falls dagegen mehrere iterierbare Objekte angegeben sind, liefern sie das Objekt mit der größten oder der kleinsten Summe. Die Funktion `sum()` liefert die Summe der Elemente des iterierbaren Objekts. Ein Beispiel:

```
t1 = (3, 12, 9)
print("t1:", t1)
print("Max. Wert:", max(t1))
print("Min. Wert:", min(t1))
print("Summe:", sum(t1))
print()

t2 = (1, 0, 13, 1, 2)
print("t2:", t2)
print("Max. Summe:", max(t1,t2))
print("Min. Summe:", min(t1,t2))
```

**Listing 5.30** Datei `max_min_sum.py`

Die Ausgabe lautet:

```
t1: (3, 12, 9)
Max. Wert: 12
Min. Wert: 3
Summe: 24
```

```
t2: (1, 0, 13, 1, 2)
Max. Summe: (3, 12, 9)
Min. Summe: (1, 0, 13, 1, 2)
```

Zur Erläuterung:

- ▶ Der größte und der kleinste Wert des Tupels `t1` werden ausgegeben. **Ein Parameter**  
Anschließend wird die Summe der Werte ausgegeben.
- ▶ Beim zweiten Aufruf der Funktionen `max()` und `min()` ist mehr als ein Parameter angegeben. Dies bedeutet, dass das Tupel mit der größten oder der kleinsten Summe der Werte ermittelt und ausgegeben wird. **Mehr als ein Parameter**

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 5.8.2 Funktionen »chr()« und »ord()«

**Unicode-Zahl** Die Funktion `chr()` liefert das zugehörige Zeichen zu einer Unicode-Zahl. Umgekehrt erhalten Sie mithilfe der Funktion `ord()` die Unicode-Zahl zu einem Zeichen. Ein Beispiel:

```
# Ziffern
for i in range(48,58):
    print(chr(i), end="")
print()

# grosse Buchstaben
for i in range(65,91):
    print(chr(i), end="")
print()

# kleine Buchstaben
for i in range(97,123):
    print(chr(i), end="")
print()

# Codenummern
for z in "Robinson":
    print(ord(z), end=" ")
print()

# Verschoben
for z in "Robinson":
    print(chr(ord(z)+1), end="")
```

**Listing 5.31** Datei `chr_ord.py`

Es wird die Ausgabe erzeugt:

```
0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
82 111 98 105 110 115 111 110
Spcjotpo
```

Zur Erläuterung:

**Ziffern, Buchstaben**

- ▶ Die Unicode-Zahlen von 48 bis 57 verweisen auf die Ziffern 0 bis 9. Mithilfe der Unicode-Zahlen von 65 bis 90 bzw. von 97 bis 122 erhalten Sie die großen und die kleinen Buchstaben.
- ▶ Eine Zeichenkette ist ein iterierbares Objekt, daher können die einzelnen Elemente (sprich: Zeichen) in einer `for`-Schleife durchlaufen werden. Für jedes Zeichen der Zeichenkette wird die zugehörige Unicode-Zahl ausgegeben.

- Im letzten Teil des Programms wird jedes Zeichen einer Zeichenkette in das codemäßig folgende Zeichen umgewandelt. Die Zeichenkette wird *verschlüsselt*. Dazu werden beide Funktionen eingesetzt.

Verschlüsselung

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Das Weglassen des Zeilenendes wird (wie in [Abschnitt 5.2.1](#), »Funktion `>print()<`«) erreicht, indem die Ausgabe stückweise in einer Variablen zusammengesetzt und erst anschließend vollständig ausgegeben wird.

### 5.8.3 Funktionen »reversed()« und »sorted()«

Die Funktion `reversed()` liefert ein iterierbares Objekt in umgekehrter Reihenfolge. Mithilfe der Funktion `sorted()` wird eine sortierte Liste erstellt. Ein Beispiel:

```
# Originale
z = "Robinson"
print(z)
t = [4, 12, 6, -2]
print(t)

# Umgedreht
r = reversed(z)
for x in r:
    print(x, end="")
print()

# Sortierte Listen
s1 = sorted(z)
print(s1)
s2 = sorted(t)
print(s2)
```

**Listing 5.32** Datei `reversed_sorted.py`

Die Ausgabe lautet:

```
Robinson
[4, 12, 6, -2]
nosniboR
['R', 'b', 'i', 'n', 'n', 'o', 'o', 's']
[-2, 4, 6, 12]
```

Zur Erläuterung:

Umdrehen,  
sortieren

- Umdrehen** ► Die Funktion `reversed()` liefert einen Iterator, der die Elemente in umgekehrter Reihenfolge enthält. Die Elemente können z. B. mithilfe einer `for`-Schleife ausgegeben werden.
- Sortieren** ► Mithilfe der Funktion `sorted()` wird eine Liste erstellt, die die Elemente einer gegebenen Sequenz in sortierter Reihenfolge enthält. Bei Zahlen ist die Sortierung aufsteigend nach Wert, bei Zeichen aufsteigend nach Codezahl.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Das Weglassen des Zeilenendes wird (wie in [Abschnitt 5.2.1](#), »Funktion `>print()`«) erreicht, indem die Ausgabe stückweise in einer Variablen zusammengesetzt und erst anschließend vollständig ausgegeben wird.

## 5.9 Statistikfunktionen

Die Statistik dient dazu, große Mengen von Zahlenwerten zu analysieren und bestimmte, repräsentative Informationen über diese Zahlenwerte zu ermitteln. Diese Mengen können z. B. aus Umfragen oder aus anderen Stichproben stammen. Python bietet dazu seit der Version 3.4 einige hilfreiche Funktionen innerhalb des Moduls `statistics`.

Ein Beispiel mit einigen dieser Funktionen:

```
import statistics

# Arithmetischer Mittelwert, hier einer Liste
probe1 = [5,2,4,17]
print("Mittelwert:", statistics.mean(probe1))
print()

# Median
print("Median:", statistics.median(probe1))
probe2 = [5,2,4,17,3]
print("Median:", statistics.median(probe2))
print()

# Unterer Median
print("Unterer Median:", statistics.median_low(probe1))
print("Unterer Median:", statistics.median_low(probe2))
print()

# Oberer Median
print("Oberer Median:", statistics.median_high(probe1))
print("Oberer Median:", statistics.median_high(probe2))
print()
```

```
# Tupel, Werte eines Dictionary
probe3 = (5,2,4,17)
print("aus Tupel:", statistics.mean(probe3))
probe4 = {'D':5, 'NL':2, 'CH':4, 'F':17}
print("aus Dictionary:", statistics.mean(probe4.values()))
```

**Listing 5.33** Datei statistik.py

Es wird die folgende Ausgabe erzeugt:

**Mittelwert: 7.0**

**Median: 4.5**

**Median: 4**

**Unterer Median: 4**

**Unterer Median: 4**

**Oberer Median: 5**

**Oberer Median: 4**

**aus Tupel: 7.0**

**aus Dictionary: 7.0**

Zur Erläuterung:

- ▶ Die Werte der Zahlenmenge können ganzzahlig sein oder auch Nachkommastellen haben. Sie können aus einer Liste (wie hier), aber auch aus einem Tupel oder einem Dictionary stammen, siehe unten. Sie müssen nicht in sortierter Reihenfolge vorliegen.
- ▶ Die Funktion `mean()` liefert den arithmetischen Mittelwert der Zahlenmenge, also die Summe der Werte, geteilt durch die Anzahl der Werte.
- ▶ Die Funktion `median()` liefert den Median. Dieser wird auch Zentralwert genannt, da er im Zentrum der Zahlenmenge steht: Es gibt genau so viele Werte, die größer sind als der Median, wie Werte, die kleiner sind als der Median. Bei einer ungeraden Menge von Werten handelt es sich um das Element in der Mitte der Zahlenmenge. Bei einer geraden Menge von Werten handelt es sich um den arithmetischen Mittelwert der beiden Elementen in der Mitte der Zahlenmenge.
- ▶ Die Funktionen `median_low()` und `median_high()` liefern den unteren bzw. oberen Median. Dabei handelt es sich in jedem Fall um Elemente aus der Zahlenmenge. Bei einer ungeraden Menge von Werten handelt es sich um das Element in der Mitte der Zahlenmenge. Bei einer geraden Menge von Werten handelt es sich um die beiden Elemente in der Mitte der Zahlenmenge.
- ▶ Die untersuchte Zahlenmenge kann auch aus einem Tupel oder, mithilfe der Methode `values()`, aus dem Werte-View eines Dictionary stammen.

`mean()`

`median()`

## Unterschiede in Python 2

In Python 2 gibt es das Modul `statistics` nicht.

## 5.10 Eigene Module

In den bisherigen Beispielen wurden die Funktion und das eigentliche Hauptprogramm in der gleichen Datei definiert. Bei spezifischen Funktionen, die auf ein bestimmtes Programm zugeschnitten wurden, ist dies auch sinnvoll.

<b>Gemeinsame Nutzung</b>	Allerdings werden Sie bald feststellen, dass einige nützliche Funktionen immer wieder und von verschiedenen Programmen aus benötigt werden. Diese Funktionen sollten in eigenen Modulen gespeichert werden. Die Erstellung und Nutzung von Modulen ist in Python sehr einfach und wird in diesem Abschnitt erläutert.
---------------------------	---

### 5.10.1 Eigene Module erzeugen

<b>Externe Funktion</b>	Zur Erzeugung eines Moduls speichern Sie die gewünschte Funktion einfach in einer eigenen Datei. Der Name der Datei ist gleichzeitig der Name des Moduls. Die Funktion erstellen Sie wie gewohnt:
-------------------------	---

```
def quadrat(x):
    erg = x * x
    return erg
```

**Listing 5.34** Datei `modul_neu.py`

Anschließend können Sie die Funktion in jedem Programm nutzen, vorausgesetzt, sie wird aus dem betreffenden Modul importiert. Die Importmöglichkeiten erläutere ich im nächsten Abschnitt.

### 5.10.2 Eigene Module verwenden

Eine Funktion in einem eigenen Modul können Sie auf mehrere Arten nutzen. Das Modul importieren Sie zunächst wie gewohnt:

```
import modul_neu
z = modul_neu.quadrat(3)
print(z)
```

**Listing 5.35** Datei `modul_verwenden.py`

Zur Erläuterung:

<b>import</b>	► Alle Funktionen des Moduls <code>modul_neu</code> , also der Datei <code>modul_neu.py</code> , werden mithilfe der Anweisung <code>import</code> zugänglich gemacht.
---------------	--

- Die Funktion `quadrat()` wird in der Schreibweise `Modulname.Funktionsname` aufgerufen.

Falls es sich um ein Modul mit einem langen, unhandlichen Namen handelt, `import ... as` können Sie es mithilfe von `as` in Ihrem Programm umbenennen:

```
import modul_neu as mn
z = mn.quadrat(3)
print(z)
```

**Listing 5.36** Datei `modul_as.py`

Zur Erläuterung:

- Auf das Modul `modul_neu` kann innerhalb dieses Programms mit `mn` zugegriffen werden.

Es folgt der Import mithilfe der Anweisung `from`:

```
from modul_neu import quadrat
z = quadrat(3)
print(z)
```

**Listing 5.37** Datei `modul_from.py`

Zur Erläuterung:

- Mit `from modul_neu import quadrat` wird die Funktion `quadrat()` aus dem Modul `modul_neu` importiert.
- Die Funktion `quadrat()` wird dann ohne den Modulnamen aufgerufen, so als ob sie in der gleichen Datei definiert worden wäre.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Hinweise

1. Falls noch weitere Funktionen im Modul `modul_neu` stehen, die alle importiert werden sollen, so können Sie auch die Anweisung `from modul_neu import *` nutzen.
2. Die Schreibweise `from Modulname import ...` hat allerdings Nachteile, falls Sie mehrere Funktionen mit gleichem Namen aus unterschiedlichen Modulen importieren möchten.
3. Auch aus diesem Grund ist die gewohnte Schreibweise mit `import Modulname` (wie in Datei `modul_verwenden.py`) sehr zu empfehlen.
4. Es wird in den Beispielen in den beiden Dateien `modul_verwenden.py` und `modul_from.py` davon ausgegangen, dass sich die Datei `modul_neu.py` im gleichen Verzeichnis befindet.

### Empfehlung

**Übung u\_modul**

Schreiben Sie das Programm aus Übung *u\_rueckgabewert* um: Die Funktion `steuer()` soll in die Datei *u\_modul\_finanz.py* ausgelagert werden.

Das Hauptprogramm in Datei *u\_modul.py* soll die Funktion aus dieser Datei importieren und nutzen.

## 5.11 Parameter der Kommandozeile

Ein Python-Programm kann bekanntlich von der Kommandozeile des Betriebssystems aus aufgerufen werden. In [Abschnitt 2.3.2](#), »Ausführen unter Windows«, und [Abschnitt 2.3.3](#), »Ausführen unter Ubuntu Linux und unter OS X«, wurde beschrieben, wie Sie zur Kommandozeile bzw. zum Terminal gelangen.

- sys.argv** Der Aufruf ähnelt dem Aufruf einer Funktion mit Parametern. Die einzelnen Parameter werden durch Leerzeichen voneinander getrennt. Sie stehen innerhalb des Programms in der Liste `argv` aus dem Modul `sys` zur Verfügung. Die Anzahl der Parameter könnten Sie mithilfe der eingebauten Funktion `len()` (Länge der Liste) ermitteln.

### 5.11.1 Übergabe von Zeichenketten

Dem folgenden Programm wird ein einzelnes Wort als Parameter übergeben. Anschließend werden Elemente der Liste `sys.argv` ausgegeben:

```
import sys
print("Programmname:", sys.argv[0])
print("Erster Parameter:", sys.argv[1])
```

**Listing 5.38** Datei kommando\_text.py

Wird das Programm von der Kommandozeile aus in folgender Form aufgerufen ...

```
python kommando_text.py hallo
```

... so lautet die Ausgabe:

```
Programmname: kommando_text.py
Erster Parameter: hallo
```

Zur Erläuterung:

- ▶ Das erste Element der Liste ist der Name des Programms.
- ▶ Die weiteren Elemente sind die einzelnen Parameter.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 5.11.2 Übergabe von Zahlen

Mit dem folgenden Programm sollen zwei Zahlen addiert werden, die dem Programm als Kommandozeilenparameter übergeben werden. Dazu müssen die beiden Parameter mithilfe der Funktion `float()` in Zahlen umgewandelt und eventuell auftretende Ausnahmen abgefangen werden.

Parameter verrechnen

```
import sys

try:
    x = float(sys.argv[1])
    y = float(sys.argv[2])
    z = x + y
    print("Ergebnis:", z)
except:
    print("Parameterfehler")
```

**Listing 5.39** Datei kommando\_zahl.py

Wird das Programm vom Betriebssystem aus wie folgt aufgerufen ...

`python kommando_zahl.py 3 6.2`

... so lautet die Ausgabe:

**Ergebnis: 9.2**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 5.11.3 Beliebige Anzahl von Parametern

Das folgende Programm soll dazu dienen, eine beliebige Menge von Zahlen zu addieren, die dem Programm als Kommandozeilenparameter übergeben werden.

Liste der Parameter

```
import sys
summe = 0

try:
    for i in sys.argv[1:]:
        summe += float(i)
```

```
    print("Ergebnis:", summe)
except:
    print("Parameterfehler")
```

### **Listing 5.40 Datei kommando\_variabel.py**

Wird das Programm vom Betriebssystem aus wie folgt aufgerufen ...

```
python kommando_variabel.py 3 6.2 5
```

... so lautet die Ausgabe:

**Ergebnis: 14.2**

Zur Erläuterung:

- Slice**
- ▶ Bei `sys.argv` handelt es sich um eine Liste. Das erste Element (der Name des Programms) muss dabei ausgespart werden, daher die Slice-Operation.
  - ▶ Innerhalb der `for`-Schleife werden die einzelnen Parameter umgewandelt und zur Gesamtsumme addiert.

### **Unterschiede in Python 2**

Die Klammern bei der Anweisung `print` entfallen.

# Kapitel 6

## Objektorientierte Programmierung

Dieses Kapitel bietet Ihnen eine Einführung in die objektorientierte Programmierung. Es erläutert die Möglichkeiten zur Erzeugung einer Klassenhierarchie, mit der Sie große Softwareprojekte anhand von Klassen, Objekten, Eigenschaften, Methoden und dem Prinzip der Vererbung bearbeiten können.

### 6.1 Was ist OOP?

Die objektorientierte Programmierung (OOP) bietet zusätzliche Möglichkeiten zum verbesserten Aufbau und zur vereinfachten Wartung und Erweiterung von Programmen. Diese Vorteile erschließen sich besonders bei großen Programmierprojekten.

Bei der objektorientierten Programmierung werden Klassen erzeugt, in denen die Eigenschaften von Objekten sowie die Funktionen festgelegt werden, die auf diese Objekte angewendet werden können (sogenannte *Methoden*). Sie können viele verschiedene Objekte dieser Klassen erzeugen, den Eigenschaften unterschiedliche Werte zuweisen und die Methoden anwenden. Die Definitionen aus der Klasse und die zugewiesenen Werte begleiten diese Objekte über ihren gesamten *Lebensweg* während der Dauer des Programms.

Klasse

Ein Beispiel: Es wird die Klasse `Fahrzeug` erschaffen, in der Eigenschaften und Methoden von verschiedenen Fahrzeugen bestimmt werden. Ein Fahrzeug hat unter anderem die Eigenschaften Bezeichnung, Geschwindigkeit und Fahrtrichtung. Außerdem kann man ein Fahrzeug beschleunigen und lenken. Innerhalb des Programms können viele unterschiedliche Fahrzeuge erschaffen und eingesetzt werden.

Klassen können ihre Eigenschaften und Methoden außerdem vererben. Sie fungieren dann als Basisklasse, ihre Erben nennt man *abgeleitete Klassen*. Dadurch kann die Definition von ähnlichen Objekten, die über eine Reihe von gemeinsamen Eigenschaften und Methoden verfügen, vereinfacht werden.

Vererbung

Ein Beispiel: Es werden die Klassen `PKW` und `LKW` geschaffen. Beide Klassen sind von der Basisklasse `Fahrzeug` abgeleitet und erben alle Eigenschaften und Methoden dieser Klasse. Zusätzlich verfügen sie über eigene Eigenschaften und Methoden, die bei der jeweiligen Klasse besonders wichtig sind. Ein `PKW` hat etwa eine bestimmte Anzahl von Insassen, und man kann einsteigen und aussteigen. Ein `LKW` hat eine Ladung, man kann ihn beladen und entladen.

## 6.2 Klassen, Objekte und eigene Methoden

**Eigenschaft, Methode** Als Beispiel wird die Klasse Fahrzeug definiert. Zunächst verfügt ein Objekt dieser Klasse nur über die Eigenschaft Geschwindigkeit und die Methoden beschleunigen() und ausgabe(). Die Methode ausgabe() soll dazu dienen, den Anwender über den aktuellen Zustand des jeweiligen Fahrzeugs zu informieren.

**Klassendefinition** Die Definition der Klasse sieht wie folgt aus:

```
# Definition der Klasse Fahrzeug
class Fahrzeug:
    geschwindigkeit = 0           # Eigenschaft
    def beschleunigen(self, wert): # Methode
        self.geschwindigkeit += wert
    def ausgabe(self):            # Methode
        print("Geschwindigkeit:", self.geschwindigkeit)
```

**Listing 6.1** Datei oop\_klasse.py, Klassendefinition

Zur Erläuterung:

- class** ▶ Die Definition der Klasse wird eingeleitet vom Schlüsselwort `class`, gefolgt vom Namen der Klasse und einem Doppelpunkt. Eingerückt folgt die eigentliche Definition.
- def** ▶ Die Eigenschaft `geschwindigkeit` wird definiert und auf den Wert 0 gesetzt.
- self** ▶ Die Methoden sind Funktionen der Klasse, sie werden also mithilfe des Schlüsselworts `def` definiert. Methoden haben immer mindestens einen Parameter, dies ist das Objekt selbst. Häufig wird dieser Parameter `self` genannt – Sie können aber auch einen anderen Namen wählen.
- ▶ Die Methode `beschleunigen()` hat insgesamt zwei Parameter: Der erste Parameter ist das Objekt selbst; es wird hier mit `self` bezeichnet. Der zweite Parameter ist der Wert für die Änderung der Geschwindigkeit. Innerhalb der Methode wird dieser Wert genutzt, um die Eigenschaft des Objekts zu ändern.
- ▶ Die Methode `ausgabe()` hat nur einen Parameter: das Objekt selbst. Sie dient zur Ausgabe der Geschwindigkeit des Objektes.

**Hauptprogramm** Bisher enthielt das Programm nur eine Klassendefinition, es führte noch nichts aus. Das Hauptprogramm hat folgendes Aussehen:

```
# Objekte der Klasse Fahrzeug erzeugen
opel = Fahrzeug()
volvo = Fahrzeug()

# Objektmethoden
volvo.ausgabe()
volvo.beschleunigen(20)
```

```
volvo.ausgabe()
```

```
# Objekt betrachten  
opel.ausgabe()
```

**Listing 6.2** Datei oop\_klasse.py, Hauptprogramm

Das Programm erzeugt die Ausgabe:

```
Geschwindigkeit: 0  
Geschwindigkeit: 20  
Geschwindigkeit: 0
```

Zur Erläuterung:

- ▶ Nach der bereits beschriebenen Klassendefinition werden zunächst zwei Objekte der Klasse `Fahrzeug` erzeugt, hier mit den Namen `opel` und `volvo` (`opel=Fahrzeug()`, `volvo=Fahrzeug()`). Diesen Vorgang nennt man auch: Instanzen einer Klasse erzeugen. Objekt erzeugen
- ▶ Beim anschließenden Aufruf der Methoden (hier `beschleunigen()` und `ausgabe()`) ist zu beachten, dass das Objekt selbst nicht als Parameter angegeben wird. Eine Methode bekommt also beim Aufruf immer einen Parameter weniger übermittelt, als in der Definition angegeben ist. Dies liegt daran, dass die Methode *für* ein bestimmtes Objekt aufgerufen wird. Innerhalb der Methode ist daher bekannt, um welches Objekt es sich handelt. Methode für Objekt ausführen
- ▶ Die Geschwindigkeit des Objekts `volvo` wird ausgegeben, einmal vor und einmal nach der Beschleunigung. Die Geschwindigkeit des Objekts `opel` wird nur einmal ausgegeben. Zu Beginn, also nach ihrer Erzeugung, haben die Objekte die Geschwindigkeit 0, wie in der Definition angegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Hinweis

Die in diesem Kapitel dargestellten Programme sind ein Kompromiss, denn die Vorteile der objektorientierten Programmierung (OOP) kommen erst bei größeren Programmierprojekten zum Tragen. Bei einem kleineren Problem fragen Sie sich vielleicht, warum Sie für dieses einfache Ergebnis ein verhältnismäßig komplexes Programm schreiben sollten. Anhand der hier vorgestellten Programme können Sie sich aber die Prinzipien der OOP erschließen, ohne den Überblick zu verlieren.

## 6.3 Konstruktor und Destruktor

Es gibt zwei besondere Methoden, die im Zusammenhang mit einer Klasse definiert werden können:

- |                    |  |
|--------------------|--|
| <b>Konstruktor</b> | ► Die Konstruktormethode wird genutzt, um einem Objekt zu Beginn seiner <i>Lebensdauer</i> Anfangswerte zuzuweisen.                              |
| <b>Destruktor</b>  | ► Die Destruktormethode wird genutzt, um am Ende der <i>Lebensdauer</i> eines Objekts Aktionen auszulösen, z. B. eine offene Datei zu schließen. |

Die Klasse `Fahrzeug` wird wie folgt verändert:

- Ein Fahrzeug erhält neben der Eigenschaft `geschwindigkeit` die Eigenschaft `bezeichnung`.
- Die Klasse erhält eine Konstruktormethode zur Festlegung von Anfangswerten für `bezeichnung` und `geschwindigkeit`.

Das Programm sieht in seiner veränderten Form wie folgt aus:

```
# Definition der Klasse Fahrzeug
class Fahrzeug:
    def __init__(self, bez, ge):          # Konstruktormethode
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def __del__(self):                  # Destruktormethode
        print("Objekt", self.bezeichnung, "entfernt")
    def beschleunigen(self, wert):
        self.geschwindigkeit += wert
        self.ausgabe()

    def ausgabe(self):
        print(self.bezeichnung, self.geschwindigkeit, "km/h")

# Objekte der Klasse Fahrzeug erzeugen
opel = Fahrzeug("Opel Admiral", 40)
volvo = Fahrzeug("Volvo Amazon", 45)

# Objekte betrachten
opel.ausgabe()
volvo.ausgabe()

# Objektmethode
volvo.beschleunigen(20)

# Destruktor aufrufen
del opel
del volvo
```

```
# Aufruf nicht mehr gestattet
# opel.ausgabe()
```

### **Listing 6.3 Datei oop\_konstruktor.py**

Die Ausgabe des Programms lautet:

```
Opel Admiral 40 km/h
Volvo Amazon 45 km/h
Volvo Amazon 65 km/h
Objekt Opel Admiral entfernt
Objekt Volvo Amazon entfernt
```

Zur Erläuterung:

- ▶ Der festgelegte Name `__init__()` (mit doppeltem Unterstrich davor und danach) bezeichnet die Konstruktormethode. Es werden zwei Parameter übergeben. Diese beiden Parameter werden genutzt, um die beiden Eigenschaften mit Anfangswerten zu versorgen.
- ▶ Der festgelegte Name `__del__()` bezeichnet die Destruktormethode. Sie wird mithilfe der Anweisung `del` aufgerufen. Anschließend existiert das Objekt nicht mehr.
- ▶ Die Methode `ausgabe()` dient zur Ausgabe beider Eigenschaften. Sie wird in diesem Programm sowohl aus dem Hauptprogramm als auch aus einer Objektmethode heraus aufgerufen. Im zweiten Fall muss, wie bei der Zuweisung eines Eigenschaftswerts, das Objekt mit `self` angegeben werden.
- ▶ Es werden zwei Objekte erzeugt. Dabei werden die Anfangswerte an den Konstruktor übergeben, hier die Werte `Opel Admiral` und `40` für das Objekt `opel`, für das Objekt `volvo` die Werte `Volvo Amazon` und `45`.
- ▶ Anschließend werden die Eigenschaften der Objekte verändert und ausgegeben.

### **Unterschiede in Python 2**

Die Klammern bei der Anweisung `print` entfallen.

### **Hinweise**

1. Konstruktoren werden häufig eingesetzt. Sie ermöglichen eine gezieltere Erzeugung von Objekten.
2. Destruktoren werden selten genutzt.

## 6.4 Besondere Methoden

Die Methoden `__init__()` und `__del__()` gehören zu einer Reihe von besonderen Methoden, die für die eingebauten Objekttypen bereits vordefiniert sind. Sie können für eigene Objekttypen, also Klassen, selbst definiert werden. Im folgenden Beispiel werden zwei dieser Methoden genutzt:

```
# Definition der Klasse Fahrzeug
class Fahrzeug:
    def __init__(self, bez, ge):      # Konstruktormethode
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def __str__(self):                # Ausgabemethode
        return self.bezeichnung + " " \
               + str(self.geschwindigkeit) + " km/h"
    def __repr__(self):               # Info-Methode
        return "Objekt " + self.bezeichnung \
               + " der Klasse Fahrzeug"

# Objekte der Klasse Fahrzeug erzeugen
opel = Fahrzeug("Opel Admiral", 40)
volvo = Fahrzeug("Volvo Amazon", 45)

# Objekte ausgeben
print(opel)
print(volvo)

# Informationen zu Objekt
print(repr(opel))
print(repr(volvo))
```

**Listing 6.4** Datei oop\_besondere.py

Die Ausgabe des Programms lautet:

```
Opel Admiral 40 km/h
Volvo Amazon 45 km/h
Objekt Opel Admiral der Klasse Fahrzeug
Objekt Volvo Amazon der Klasse Fahrzeug
```

Zur Erläuterung:

- `__str__()` ▶ Bei der Ausgabe eines Objekts mithilfe der Funktion `print()` wird die besondere Methode `__str__()` aufgerufen, falls sie definiert ist. Diese liefert eine Zeichenkette zur Ausgabe der Eigenschaften. Ist die Methode für die betreffende Klasse nicht definiert, so ergibt sich keine sinnvoll nutzbare Ausgabe.
- `__repr__()` ▶ Sie können mithilfe der eingebauten Funktion `repr()` Informationen zu einem Objekt abrufen. Das Aussehen dieser Informationen können Sie selber bestimmen, falls Sie innerhalb der Klassendefinition die besondere Methode `__repr__()` definieren.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 6.5 Operatormethoden

Ähnlich wie die besonderen Methoden sind die Operatormethoden für die eingebauten Objekttypen bereits vordefiniert. Auch die Operatormethoden können Sie für eigene Objekttypen, also Klassen, selbst definieren. Im folgenden Beispiel werden insgesamt drei dieser Methoden zum Vergleich zweier Objekte und zur Subtraktion zweier Objekte genutzt:

```
# Definition der Klasse Fahrzeug
class Fahrzeug:
    def __init__(self, bez, ge):      # Konstruktormethode
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def __gt__(self, other):         # 1. Vergleichsmethode
        return self.geschwindigkeit > other.geschwindigkeit
    def __eq__(self, other):          # 2. Vergleichsmethode
        return self.geschwindigkeit == other.geschwindigkeit
    def __sub__(self, other):         # Rechenmethode
        return self.geschwindigkeit - other.geschwindigkeit

# Objekte der Klasse Fahrzeug erzeugen
opel = Fahrzeug("Opel Admiral", 60)
volvo = Fahrzeug("Volvo Amazon", 45)

# Objekte vergleichen
if opel > volvo:
    print("Opel ist schneller")
elif opel == volvo:
    print("Beide sind gleich schnell")
else:
    print("Volvo ist schneller")

# Objekte subtrahieren
differenz = opel - volvo
print("Geschwindigkeitsdifferenz:", differenz, "km/h")
```

**Listing 6.5** Datei `oop_operator.py`

Die Ausgabe des Programms:

**Opel ist schneller**  
**Geschwindigkeitsdifferenz: 15 km/h**

Zur Erläuterung:

### Objekte vergleichen

- ▶ Die Methoden `__gt__()` und `__eq__()` werden beim Objektvergleich durch die Benutzung der Vergleichsoperatoren `>` und `==` aufgerufen.
  - Die Methode `__gt__()` liefert `True`, falls das Objekt `self` größer als das Objekt `other` ist. Für diese Klasse wird definiert: Größer bedeutet höhere Geschwindigkeit.
  - Die Methode `__eq__()` liefert `True`, falls die beiden Objekte gleich groß, also gleich schnell sind.
- ▶ `__sub__()` Die Methode `__sub__()` wird bei der Verwendung des Operators `-` (Minus) aufgerufen. Zur Ausgabe eines sinnvollen Ergebnisses sollte sie die Differenz der Geschwindigkeiten der beiden Objekte zurückliefern. Sie wird im vorliegenden Programm bei der Berechnung `differenz = opel - volvo` genutzt.

### Vergleichsoperatoren

Neben den Operatoren `>` (*greater than*) und `==` (*equal*) können weitere Vergleichsoperatoren zu Funktionsaufrufen führen:

- ▶ `>=` führt zu `__ge__()`, *greater equal*
- ▶ `<` führt zu `__lt__()`, *lower than*
- ▶ `<=` führt zu `__le__()`, *lower equal*
- ▶ `!=` führt zu `__ne__()`, *not equal*

### Rechenoperatoren

Neben dem Operator `-` (Minus) können viele weitere Rechenoperatoren zu Funktionsaufrufen führen, unter anderem:

- ▶ `+` führt zu `__add__()`
- ▶ `*` führt zu `__mul__()`
- ▶ `/` führt zu `__truediv__()`
- ▶ `//` führt zu `__floordiv__()`
- ▶ `%` führt zu `__mod__()`
- ▶ `**` führt zu `__pow__()`

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 6.6 Referenz, Identität und Kopie

### Zweite Referenz

Wie Sie bereits in [Abschnitt 4.8](#) erfahren haben, ist der Name eines Objekts lediglich eine Referenz auf das Objekt. Bei Objekten von benutzerdefinierten Klassen erzeugt die Zuweisung dieser Referenz an einen anderen Namen lediglich eine zweite Referenz auf das gleiche Objekt.

Echte Kopien von Objekten benutzerdefinierter Klassen können Sie durch Erzeugung eines neuen Objekts erstellen, dem Sie z. B. über den Konstruktor die Werte eines anderen Objekts der gleichen Klasse zuweisen.

Kopie erstellen

Für umfangreiche Objekte können Sie sich auch der Funktion `deepcopy()` aus dem Modul `copy` bedienen, siehe auch [Abschnitt 4.8.3](#), »Objekte kopieren«. Beide Vorgehensweisen werden im folgenden Programm gezeigt. Dort werden erzeugt:

`copy.deepcopy()`

- ▶ ein Objekt
- ▶ ein zweites Objekt als Kopie des ersten Objekts mithilfe von Wertzuweisungen
- ▶ ein drittes Objekt als Kopie des ersten Objekts, mit `deepcopy()`
- ▶ eine zusätzliche Referenz auf das erste Objekt

```
# Modul copy
import copy

# Definition der Klasse Fahrzeug
class Fahrzeug:
    def __init__(self, bez, ge):          # Konstruktormethode
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def beschleunigen(self, wert):
        self.geschwindigkeit += wert
    def __str__(self):                  # Ausgabemethode
        return self.bezeichnung + " " \
            + str(self.geschwindigkeit) + " km/h"

# Objekt der Klasse Fahrzeug erzeugen
opel = Fahrzeug("Opel Admiral", 40)
# Kopie eines Objektes erzeugen
zweit_opel = Fahrzeug(opel.bezeichnung, opel.geschwindigkeit)
zweit_opel.beschleunigen(30)
# Tiefe Kopie eines Objektes erzeugen
dritt_opel = copy.deepcopy(opel)
dritt_opel.beschleunigen(35)

# Zweite Referenz auf Objekt erzeugen
viert_opel = opel
viert_opel.beschleunigen(20)

# Kontrollausgaben
print("Original:", opel)
print("Kopie:", zweit_opel)
```

```
print("Kopie:", dritt_opel)
print("zweite Referenz auf Original:", viert_opel)

# Identisch
print("2:", opel is zweit_opel)
print("3:", opel is dritt_opel)
print("4:", opel is viert_opel)
```

**Listing 6.6** Datei oop\_kopieren.py

Die Ausgabe des Programms:

```
Original: Opel Admiral 60 km/h
Kopie: Opel Admiral 70 km/h
Kopie: Opel Admiral 75 km/h
zweite Referenz auf Original: Opel Admiral 60 km/h
2: False
3: False
4: True
```

Zur Erläuterung:

- ▶ Das Objekt `opel` ist das Original.
  - ▶ Das Objekt `zweit_opel` ist ein eigenes Objekt, das mithilfe des Konstruktors und der Daten des Originalobjekts erzeugt wird.
  - ▶ Das Objekt `dritt_opel` ist ein eigenes Objekt, das mithilfe der Funktion `copy.deepcopy()` erzeugt wird.
  - ▶ `viert_opel` ist eine zusätzliche Referenz auf das Originalobjekt `opel`.
  - ▶ Die Änderung einer Eigenschaft über diese Referenz ändert das Originalobjekt, wie die Ausgabe zeigt.
- Operator `is`**
- ▶ Die Vergleiche auf Identität mithilfe des Operators `is` zeigen, dass nur die Objekte `opel` und `viert_opel` identisch sind.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 6.7 Vererbung

<b>Klassenhierarchie</b>	Eine Klasse kann ihre Eigenschaften und Methoden an eine andere Klasse vererben. Dieser Mechanismus wird häufig angewendet. Sie erzeugen dadurch eine Hierarchie von Klassen, die die Darstellung von Objekten ermöglicht, die teilweise übereinstimmende, teilweise unterschiedliche Merkmale aufweisen.
<b>Erben</b>	Im folgenden Beispiel wird eine Klasse <code>PKW</code> definiert, mit deren Hilfe die Eigenschaften und Methoden von Pkws dargestellt werden sollen. Bei der Erzeugung

bedient man sich der bereits existierenden Klasse `Fahrzeug`, in der ein Teil der gewünschten Eigenschaften und Methoden bereits verwirklicht wurde. Bei der Klasse `PKW` kommen noch einige Merkmale hinzu. Diese Klasse ist spezialisiert, im Gegensatz zu der allgemeinen Klasse `Fahrzeug`.

Von der Klasse `PKW` aus gesehen ist die Klasse `Fahrzeug` eine Basisklasse. Von der Klasse `Fahrzeug` aus gesehen ist die Klasse `PKW` eine abgeleitete Klasse. Zunächst die Definition der Basisklasse `Fahrzeug`. Sie enthält drei Methoden und zwei Eigenschaften:

- ▶ die Konstruktormethode `__init__()`
- ▶ die eigene Methode `beschleunigen()`
- ▶ die Ausgabemethode `__str__()`
- ▶ die Eigenschaft `bezeichnung`
- ▶ die Eigenschaft `geschwindigkeit`

```
# Definition der Klasse Fahrzeug
class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def beschleunigen(self, wert):
        self.geschwindigkeit += wert
    def __str__(self):
        return self.bezeichnung + " " \
            + str(self.geschwindigkeit) + " km/h"
```

#### **Listing 6.7 Datei oop\_vererbung.py (Basisklasse Fahrzeug)**

Es folgt die Definition der abgeleiteten Klasse `PKW`. Diese erbt von der Klasse `Fahrzeug` und enthält fünf Methoden und drei Eigenschaften:

- ▶ die eigene Konstruktormethode `__init__()` und die eigene Ausgabemethode `__str__()`, die jeweils die gleichnamige Methode der Basisklasse überschreiben
- ▶ die eigenen Methoden `einstiegen()` und `aussteigen()`
- ▶ die von der Klasse `Fahrzeug` geerbte Methode `beschleunigen()`
- ▶ die eigene Eigenschaft `insassen`
- ▶ die von der Klasse `Fahrzeug` geerbten Eigenschaften `bezeichnung` und `geschwindigkeit`

**Basisklasse und  
abgeleitete Klasse**

```
# Definition der Klasse PKW
class PKW(Fahrzeug):
    def __init__(self, bez, ge, ins):
        Fahrzeug.__init__(self, bez, ge)
        self.insassen = ins
```

**Methode  
überschreiben**

```

def __str__(self):
    return Fahrzeug.__str__(self) + " " \
        + str(self.insassen) + " Insassen"
def einsteigen(self, anzahl):
    self.insassen += anzahl
def aussteigen(self, anzahl):
    self.insassen -= anzahl

```

**Listing 6.8** Datei oop\_vererbung.py, abgeleitete Klasse »PKW«

Zur Erläuterung:

- ▶ **class ... (Basisklasse)** Wenn eine Klasse von einer anderen Klasse abgeleitet wird, so wird der Name der Basisklasse in Klammern hinter dem Namen der abgeleiteten Klasse angegeben.
- ▶ Eine Methode einer Basisklasse kann in einer abgeleiteten Klasse überschrieben werden.
- ▶ Um eine Methode der Basisklasse aufzurufen, müssen Sie den Namen dieser Klasse angeben.

**Basisklassen-methode**

Im Hauptprogramm wird ein Objekt der Klasse PKW erzeugt, verändert und ausgegeben:

```

# Objekt der abgeleiteten Klasse PKW erzeugen
fiat = PKW("Fiat Marea", 50, 0)

# eigene Methode anwenden
fiat.einstiegen(3)
fiat.aussteigen(1)

# geerbte Methode anwenden
fiat.beschleunigen(10)

# ueberschriebene Methode anwenden
print(fiat)

```

**Listing 6.9** Datei oop\_vererbung.py, Hauptprogramm

Die Ausgabe des Programms:

**Fiat Marea 60 km/h 2 Insassen**

Zur Erläuterung:

**Konstruktor ruft Konstruktor**

- ▶ Es wird das Objekt fiat erzeugt, dabei wird der Konstruktor aufgerufen. Er wird unmittelbar in der zugehörigen Klasse PKW gefunden. Dieser Konstruktor ruft wiederum den Konstruktor der Basisklasse auf, der bereits in der Lage ist, die Erzeugung von Objekten der Basisklasse durchzuführen. Diese abge-

stufe Vorgehensweise ist nicht zwingend vorgeschrieben, aber empfehlenswert. Anschließend wird die verbliebene Eigenschaft der abgeleiteten Klasse initialisiert.

- ▶ Es werden die Methoden `einsteigen()` und `aussteigen()` aufgerufen. Diese werden unmittelbar in der Klasse `PKW` gefunden und verändern die Eigenschaft `insassen`.
- ▶ Es wird die Methode `beschleunigen()` aufgerufen. Diese wird nicht unmittelbar in der Klasse `PKW` gefunden, daher wird in der Basisklasse weitergesucht. Dort wird sie gefunden und dient zur Veränderung der Eigenschaft `geschwindigkeit`.
- ▶ Es wird die Ausgabemethode `__str__()` aufgerufen. Diese wird unmittelbar in der Klasse `PKW` gefunden. Sie ruft wiederum die gleichnamige Methode der Basisklasse auf. Das Ergebnis dieses Aufrufs wird mit den restlichen Daten aus der Klasse `PKW` zusammengesetzt und zurückgeliefert. Dies führt zur Ausgabe aller Daten des Objekts `fiat`.

**Methode suchen**

**Methode ruft  
Methode**

**Suchreihenfolge**

Eigenschaften und Methoden werden zunächst in der Klasse des Objekts gesucht. Sollten sie dort nicht vorhanden sein, so wird die Suche in der zugehörigen Basisklasse fortgesetzt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 6.8 Mehrfachvererbung

Eine abgeleitete Klasse kann wiederum Basisklasse für eine weitere Klasse sein. Dadurch ergibt sich eine Vererbung über mehrere Ebenen. Eine Klasse kann außerdem von zwei Basisklassen gleichzeitig abgeleitet sein (Mehrfachvererbung), sie erbt dann alle Eigenschaften und Methoden beider Basisklassen.

**Mehrere Basis-  
klassen**

Im folgenden Beispiel werden die beiden Klassen `PKW` und `LKW` von der Klasse `Fahrzeug` abgeleitet. Außerdem wird die Klasse `Lieferwagen` erzeugt, die sowohl von der Klasse `PKW` als auch von der Klasse `LKW` erbt. Die Definitionen der Klassen `Fahrzeug` und `PKW` sind bereits aus [Abschnitt 6.7, »Vererbung«](#), bekannt.

Die abgeleitete Klasse `LKW` ähnelt im Aufbau der Klasse `PKW`, erbt von der Klasse `Fahrzeug` und enthält fünf Methoden und drei Eigenschaften:

**Klasse LKW**

- ▶ die eigene Konstruktormethode `__init__()` und die eigene Ausgabemethode `__str__()`, die jeweils die gleichnamige Methode der Basisklasse überschreiben
- ▶ die eigenen Methoden `beladen()` und `entladen()`

- ▶ die von der Klasse `Fahrzeug` geerbte Methode `beschleunigen()`
- ▶ die eigene Eigenschaft `ladung`
- ▶ die von der Klasse `Fahrzeug` geerbten Eigenschaften `bezeichnung` und `geschwindigkeit`

```
# Definition der Klasse LKW
class LKW(Fahrzeug):
    def __init__(self, bez, ge, la):
        Fahrzeug.__init__(self, bez, ge)
        self.ladung = la
    def __str__(self):
        return Fahrzeug.__str__(self) + " " \
            + str(self.ladung) + " Tonnen Ladung"
    def beladen(self, wert):
        self.ladung += wert
    def entladen(self, wert):
        self.ladung -= wert
```

**Listing 6.10** Datei `oop_mehrach.py`, abgeleitete Klasse »LKW«

**Klasse Lieferwagen** Die Klasse `Lieferwagen` erbt von zwei Klassen, die eine gemeinsame Basisklasse haben. Sie enthält sieben Methoden und vier Eigenschaften:

- ▶ die eigene Konstruktormethode `__init__()` und die eigene Ausgabemethode `__str__()`, die jeweils die geerbten Methoden überschreiben
- ▶ die von der Klasse `PKW` geerbten Methoden `einstigen()` und `aussteigen()`
- ▶ die von der Klasse `LKW` geerbten Methoden `beladen()` und `entladen()`
- ▶ die von der Klasse `Fahrzeug` geerbte Methode `beschleunigen()`
- ▶ die von der Klasse `PKW` geerbte Eigenschaft `insassen`
- ▶ die von der Klasse `LKW` geerbte Eigenschaft `ladung`
- ▶ die von der Klasse `Fahrzeug` geerbten Eigenschaften `bezeichnung` und `geschwindigkeit`

```
# Definition der Klasse Lieferwagen
class Lieferwagen(PKW, LKW):
    def __init__(self, bez, ge, ins, la):
        PKW.__init__(self, bez, ge, ins)
        LKW.__init__(self, bez, ge, la)
    def __str__(self):
        return PKW.__str__(self) + "\n" \
            + LKW.__str__(self)
```

**Listing 6.11** Datei `oop_mehrach.py`, abgeleitete Klasse »Lieferwagen«

Im Hauptprogramm wird ein Objekt der Klasse Lieferwagen erzeugt, verändert und ausgegeben:

```
# Objekt der abgeleiteten Klasse Lieferwagen erzeugen
toyota = Lieferwagen("Toyota Allround", 0, 0, 0)
toyota.einsteigen(2)
toyota.beladen(3.5)
toyota.beschleunigen(30)
print(toyota)
```

**Listing 6.12** Datei oop\_mehrfach.py, Hauptprogramm

Die Ausgabe des Programms lautet:

**Toyota Allround 30 km/h 2 Insassen**  
**Toyota Allround 30 km/h 3.5 Tonnen Ladung**

Zur Erläuterung:

- ▶ Es wird das Objekt toyota erzeugt, dabei wird der Konstruktor aufgerufen. Er wird unmittelbar in der zugehörigen Klasse Lieferwagen gefunden. Dieser Konstruktor ruft wiederum die beiden Konstruktoren der Basisklassen PKW und LKW auf. Diese rufen jeweils zum einen den Konstruktor der Basisklasse Fahrzeug auf und initialisieren zum anderen eine spezifische Eigenschaft der Klasse (insassen oder ladung).
- ▶ Es werden die Methoden einsteigen(), beladen() und beschleunigen() aufgerufen. Diese werden in verschiedenen Basisklassen gefunden und verändern die Eigenschaften insassen, ladung und geschwindigkeit.
- ▶ Es wird die eigene Ausgabemethode `_str_()` aufgerufen. Ähnlich wie beim Konstruktor ruft diese die gleichnamigen Methoden der Basisklassen auf.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 6.9 Enumerationen

Enumerationen sind Aufzählungen von Konstanten. Der Programmcode wird durch die Nutzung der Elemente einer Enumeration besser lesbar. Python bietet seit der Version 3.4 innerhalb des Moduls `enum` unter anderem die Klasse `IntEnum` zur Erstellung einer Enumeration, deren Elemente als Konstanten für ganze Zahlen stehen.

`enum`

Ein Beispiel:

```
# Definition einer Enumeration mit ganzen Zahlen
import enum
class Farbe(enum.IntEnum):
    rot = 5
    gelb = 2
    blau = 4

# Vergleich
x = 2
if x == Farbe.gelb:
    print("Das ist gelb")
print()

# Alle Elemente
for f in Farbe:
    print(f, repr(f))
print()

# Verschiedene Ausgaben und Berechnungen
print(Farbe.gelb)
print(Farbe.gelb * 1)
print(Farbe.gelb * 10)
```

**Listing 6.13** Datei enumeration.py

Es wird die folgende Ausgabe erzeugt:

**Das ist gelb**

```
Farbe.rot <Farbe.rot: 5>
Farbe.gelb <Farbe.gelb: 2>
Farbe.blau <Farbe.blau: 4>
```

```
Farbe.gelb
2
20
```

Zur Erläuterung:

**IntEnum**

- ▶ Nach Import des Moduls `enum` wird die Klasse `Farbe` erzeugt, die von der Klasse `IntEnum` erbt. Die Elemente einer Enumeration müssen unmittelbar mit Werten versehen werden. Diese müssen nicht in aufsteigender Reihenfolge sein. Ein bestimmter Wert könnte durch mehrere Konstanten repräsentiert werden, dies wäre allerdings nicht sinnvoll.
- ▶ Der Vergleich eines Werts mit einem bestimmten anderen Wert wird durch die Elemente der Enumeration erleichtert. Dies verbessert die Lesbarkeit des Programmcodes.

- Die Enumeration ist iterierbar. Mithilfe einer Schleife können Sie alle Elemente in der definierten Reihenfolge auflisten. Die eingebaute Funktion `repr()` liefert weitere Informationen zu den Elementen.
- Sie können den repräsentierten, ganzzahligen Wert auch direkt in Berechnungen einbauen.

### Unterschiede in Python 2

In Python 2 gibt es das Modul `enum` nicht.

## 6.10 Spiel, objektorientierte Version

In diesem Abschnitt stelle ich Ihnen eine objektorientierte Version des Spiels vor. In der Anwendung gibt es ein Objekt der Klasse `Spiel`. Während der Lebensdauer dieses Objekts werden mehrere Objekte der Klasse `Aufgabe` erzeugt und genutzt. Zunächst die Import-Anweisung und das kurze Hauptprogramm:

Mehrere Klassen

```
import random
```

```
# Hauptprogramm
s = Spiel()
s.spielen()
print(s)
```

**Listing 6.14** Datei `spiel_oop.py`, Hauptprogramm

Zur Erläuterung:

- Das Modul `random` wird für den Zufallsgenerator benötigt.
- Im Hauptprogramm wird das Objekt `s` der Klasse `Spiel` erzeugt. Dabei wird ein Spiel initialisiert.
- Für dieses Objekt wird die Methode `spielen()` ausgeführt. Dies ist der eigentliche Spielvorgang.
- Anschließend werden die Spielergebnisse ausgegeben.

Es folgt die Klasse `Spiel`:

Klasse »Spiel«

```
class Spiel:
    def __init__(self):
        # Start des Spiels
        random.seed()
        self.richtig = 0
```

```

# Anzahl bestimmen
self.anzahl = -1
while self.anzahl<0 or self.anzahl>10:
    try:
        print("Wie viele Aufgaben (1 bis 10):")
        self.anzahl = int(input())
    except:
        continue

def spielen(self):
    # Spielablauf
    for i in range(1,self.anzahl+1):
        a = Aufgabe(i, self.anzahl)
        print(a)
        self.richtig += a.beantworten()

def __str__(self):
    # Ergebnis
    return "Richtig: " + str(self.richtig) \
           + " von " + str(self.anzahl)

```

**Listing 6.15** Datei `spiel_oop.py`, Klasse »Spiel«

Zur Erläuterung:

- |                                   |   |
|-----------------------------------|---|
| <b>Konstruktor, Eigenschaften</b> | <ul style="list-style-type: none"> <li>► Im Konstruktor der Klasse wird der Zufallsgenerator initialisiert. Der Zähler für die richtig gelösten Aufgaben wird zunächst auf 0 gesetzt. Es wird die Anzahl der zu lösenden Aufgaben ermittelt. Dabei werden zwei Eigenschaften der Klasse <code>Spiel</code> gesetzt: <code>richtig</code> und <code>anzahl</code>.</li> </ul>  |
| <b>Methoden</b>                   | <ul style="list-style-type: none"> <li>► In der Methode <code>spielen()</code> wird die gewünschte Anzahl an Aufgaben erzeugt. Jede Aufgabe ist ein Objekt der Klasse <code>Aufgabe</code>. Die Aufgabe wird ausgegeben, also dem Benutzer gestellt. Anschließend wird die Methode <code>beantworten()</code> aufgerufen, also die Eingabe des Benutzers verarbeitet. Rückgabewert der Methode <code>beantworten()</code> ist 1 oder 0. Entsprechend wird der Zähler von <code>richtig</code> verändert.</li> <li>► In der Ausgabemethode wird das Ergebnis des Spiels veröffentlicht.</li> </ul> |
| <b>Klasse »Aufgabe«</b>           | <p>Zu guter Letzt die Klasse <code>Aufgabe</code>:</p> <pre> class Aufgabe:     # Aufgabe initialisieren     def __init__(self, i, anzahl):         self.nr = i         self.gesamt = anzahl      # Aufgabe stellen     def __str__(self):         a = random.randint(10,30)         b = random.randint(10,30) </pre>   |

```

self.ergebnis = a + b
return "Aufgabe " + str(self.nr) \
+ " von " + str(self.gesamt) + " : " \
+ str(a) + " + " + str(b)

# Aufgabe beantworten
def beantworten(self):
    try:
        if self.ergebnis == int(input()):
            print(self.nr, ": ***Richtig ***")
            return 1
        else:
            raise
    except:
        print(self.nr, ": *** Falsch ***")
    return 0

```

**Listing 6.16** Datei `spiel_oop.py`, Klasse »Aufgabe«

Zur Erläuterung:

- ▶ Jedes Objekt der Klasse `Aufgabe` bekommt bei seiner Erzeugung seine eigene Nummer. Außerdem wird die Gesamtanzahl der Aufgaben übermittelt. Dabei werden zwei Eigenschaften der Klasse `Aufgabe` gesetzt: `nr` und `gesamt`.
- ▶ In der Ausgabemethode wird die Aufgabe zusammengesetzt und veröffentlicht. Das richtige Ergebnis der Aufgabe ist eine Eigenschaft der Klasse `Aufgabe`.
- ▶ In der Methode `beantworten()` wird die Eingabe des Benutzers entgegengenommen und mit dem richtigen Ergebnis der Aufgabe verglichen. Eine falsche oder ungültige Eingabe erzeugt eine Ausnahme, dies führt zur Rückgabe einer 0. Die Eingabe des richtigen Ergebnisses führt zur Rückgabe einer 1.

**Konstruktor,  
Eigenschaften**

**Rückgabe**

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.



# Kapitel 7

## Verschiedene Module

In diesem Kapitel erläutere ich Programmietechniken zur Arbeit mit Datums- und Zeitangaben, Collections, die Technik des Multithreadings sowie reguläre Ausdrücke.

### 7.1 Datum und Zeit

Das Modul `time` enthält Funktionen zur Verarbeitung und Formatierung von Datums- und Zeitangaben.

**Modul time**

Auf vielen Betriebssystemen gilt der 1. Januar 1970 00:00 Uhr als Nullpunkt für die Verarbeitung von Datums- und Zeitangaben. Die Zeit wird in Sekunden ab diesem Zeitpunkt gerechnet.

**Zeitlicher Nullpunkt**

#### 7.1.1 Spielen mit Zeitangabe

Das Spiel aus dem Programmierkurs können Sie um Datums- und Zeitangaben erweitern. Beispielsweise könnten Sie eine Zeitmessung einbauen, die feststellt, wie lange ein Benutzer zur Lösung einer oder aller Aufgaben benötigt hat.

**Zeit messen**

Falls eine Bestleistung erzielt wurde – z. B. eine hohe Anzahl an gelösten Aufgaben im ersten Versuch oder die Lösung der Aufgaben innerhalb besonders kurzer Zeit –, so möchten Sie dies vielleicht mit Datum und Uhrzeit festhalten.

#### 7.1.2 Aktuelle Zeit anzeigen

Zwei Beispielprogramme sollen die Ermittlung und die formatierte Ausgabe der aktuellen Zeit verdeutlichen. Zunächst ein Programm, das die Funktionen `time()` und `localtime()` aus dem Modul `time` nutzt:

**time(), localtime()**

```
# Modul time
import time

# Zeit in Sekunden
print("Zeit in Sekunden:", time.time())
# Aktuelle, lokale Zeit als Tupel
lt = time.localtime()

# Entpacken des Tupels
# Datum
jahr, monat, tag = lt[0:3]
```

```
print("Es ist der {0:02d}.{1:02d}.{2:02d}.".format(tag, monat, jahr))

# Uhrzeit
stunde, minute, sekunde = lt[3:6]
print("genau {0:02d}:{1:02d}:{2:02d}.".format(stunde, minute, sekunde))

# Wochentag
wtage = ["Montag", "Dienstag", "Mittwoch", "Donnerstag",
         "Freitag", "Samstag", "Sonntag"]
wtagnr = lt[6]
print("Das ist ein", wtage[wtagnr])

# Tag des Jahres
tag_des_jahres = lt[7]
print("Der {0:d}. Tag des Jahres".format(tag_des_jahres))

# Sommerzeit
dst = lt[8]
if dst == 1:
    print("Die Sommerzeit ist aktiv")
elif dst == 0:
    print("Die Sommerzeit ist nicht aktiv")
else:
    print("Keine Sommerzeitinformation vorhanden")
```

### Listing 7.1 Datei zeit\_localtime.py

Es wird die Ausgabe erzeugt:

```
Zeit in Sekunden: 1385535272.724174
Es ist der 27.11.2013
genau 07:54:32
Das ist ein Mittwoch
Der 331. Tag des Jahres
Die Sommerzeit ist nicht aktiv
```

Zur Erläuterung:

- ▶ **time()** Die Funktion `time()` liefert die aktuelle Zeit in Sekunden seit dem 1. Januar 1970.
- ▶ **localtime()** Die Funktion `localtime()` ohne Parameter liefert die aktuelle Zeit als ein Tupel, das aus einer Reihe von Einzelinformationen besteht. Diese Informationen werden sodann ausgewertet:
  - Die ersten drei Elemente (0 bis 2) liefern Jahr, Monat und Tag. Monat und Tag sind im Beispiel jeweils mit zwei Ziffern und führender Null formatiert.
- ▶ **Jahr, Monat, Tag**

- Die nächsten drei Elemente (3 bis 5) liefern Stunde, Minute und Sekunde. Alle drei Angaben sind im Beispiel jeweils mit zwei Ziffern und führender Null formatiert.
- Das nächste Element (6) stellt den Wochentag von 0 bis 6 bereit. Montag entspricht dabei der 0, Sonntag der 6.
- Die laufende Nummer des Tages innerhalb eines Jahres wird von Element 7 geliefert.
- Informationen über den Status der Sommerzeit liefert das letzte Element (8).

Stunde, Minute,  
Sekunde

Wochentag

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### Funktion »`strftime()`«

Noch genauere Informationen liefert die Funktion `strftime()`. Sie benötigt zwei Parameter:

- ▶ einen Formatierungsstring für die gewünschte Ausgabe
- ▶ ein Zeittupel, wie es z. B. die Funktion `localtime()` liefert

Über die Möglichkeiten von `localtime()` hinaus werden folgende Informationen bereitgestellt:

- ▶ Zeit im 12-Stunden-Format mit der Angabe AM oder PM
- ▶ Jahresangabe nur mit zwei Ziffern
- ▶ Name des Wochentags, abgekürzt oder ausgeschrieben; der Sonntag entspricht dabei der 0 (!) im Unterschied zu `localtime()`
- ▶ Name des Monats, abgekürzt oder ausgeschrieben
- ▶ Kalenderwoche des Jahres, bezogen auf zwei verschiedene Systeme (Sonntag oder Montag als erster Tag der Woche)
- ▶ Angabe der Zeitzone

Wochentagsname

Monatsname

Ein Beispielprogramm:

```
# Modul time
import time

# Aktuelle Zeit
lt = time.localtime()

print(time.strftime("Tag.Monat.Jahr: %d.%m.%Y", lt))
print(time.strftime("Stunde:Minute:Sekunde: %H:%M:%S", lt))
```

```

print(time.strftime("im 12-Stunden-Format:"
                   "%I:%M:%S Uhr %p", lt))
print(time.strftime("Datum und Zeit: %c", lt))
print(time.strftime("nur Datum: %x, nur Zeit: %X", lt))
print(time.strftime("Jahr in zwei Ziffern: %y", lt))
print(time.strftime("Tag des Jahres: %j", lt))
print()

# Woche, Monat
print(time.strftime("Wochentag kurz:%a, ganz:%A"
                   ", Nr.(Sonntag=0):%w", lt))
print(time.strftime("Monat kurz:%b, ganz:%B", lt))
print()

# Kalenderwoche
print(time.strftime("Woche des Jahres, "
                   "Beginn Sonntag: %U", lt))
print(time.strftime("Woche des Jahres, "
                   "Beginn Montag: %W", lt))
print()

# Zeitzone
print(time.strftime("Zeitzone: %Z", lt))

```

### **Listing 7.2 Datei zeit.strftime.py**

Die Ausgabe lautet:

**Tag.Monat.Jahr:** 27.11.2013  
**Stunde:Minute:Sekunde:** 07:57:30  
**im 12-Stunden-Format:** 07:57:30 Uhr AM  
**Datum und Zeit:** 11/27/13 07:57:30  
**nur Datum:** 11/27/13, **nur Zeit:** 07:57:30  
**Jahr in zwei Ziffern:** 13  
**Tag des Jahres:** 331

**Wochentag kurz:**Wed, **ganz:**Wednesday, **Nr.(Sonntag=0):**3  
**Monat kurz:**Nov, **ganz:**November

**Woche des Jahres, Beginn Sonntag:** 47  
**Woche des Jahres, Beginn Montag:** 47

**Zeitzone:** Mitteleuropäische Zeit

### **Unterschiede in Python 2**

Die Klammern bei der Anweisung `print` entfallen.

### Unterschiede in Ubuntu Linux und OS X

Die einfache Formatierung mit `%c` erzeugt die folgende Ausgabe: `Wed Nov 27 07:57:30 2013`. Für die Zeitzone wird der englische Begriff `CET` ausgegeben.

### 7.1.3 Zeitangabe erzeugen

Zur Erzeugung einer beliebigen Zeitangabe wird die Funktion `mkttime()` genutzt. Sie benötigt als Parameter ein Tupel mit allen neun Angaben, wie sie auch von der Funktion `localtime()` geliefert werden.

Die Angaben für Wochentag, Tag des Jahres und Sommerzeit kennen Sie natürlich nur in den wenigsten Fällen. Sie können einfach mit `0` besetzt werden, sie werden dennoch automatisch korrekt ermittelt, wie das folgende Beispiel zeigt:

```
# Modul time
import time

# Zeitangabe erzeugen
dztupel = 1979, 2, 15, 13, 0, 0, 0, 0, 0
print(time.strftime("%d.%m.%Y %H:%M:%S", dztupel))
damals = time.mktime(dztupel)

# Ausgabe
lt = time.localtime(damals)

# Wochentag
wtag = ["Montag", "Dienstag", "Mittwoch", "Donnerstag",
        "Freitag", "Samstag", "Sonntag"]
wtagnr = lt[6]
print("Das ist ein", wtag[wtagnr])

# Tag des Jahres
tag_des_jahres = lt[7]
print("Der {0:d}. Tag des Jahres".format(tag_des_jahres))
```

**Listing 7.3** Datei `zeit_erzeugen.py`

Die Ausgabe lautet:

**15.02.1979 13:00:00**

**Das ist ein Donnerstag**

**Der 46. Tag des Jahres**

Zur Erläuterung:

- ▶ Das Datum »15. Februar 1979 13:00 Uhr« soll gespeichert werden. Dazu werden die ersten vier Elemente des Tupels mit den gegebenen Werten besetzt. Die Angaben für Minute und Sekunde sowie die drei restlichen Angaben werden mit 0 besetzt.
- ▶ Wird die Funktion `localtime()` mit einer Zeitangabe als Parameter aufgerufen, so liefert sie ein Tupel mit den einzelnen Informationen zu dieser Zeitangabe.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 7.1.4 Mit Zeitangaben rechnen

**Zeitdifferenz** Zur Berechnung eines Zeitraums, also der Differenz zwischen zwei Zeitangaben, müssen beide Zeitangaben einzeln erzeugt werden. Anschließend können Sie die Differenz aus den beiden Zeitangaben in Sekunden berechnen. Aus dieser Differenz lässt sich dann auch die Differenz in Minuten, Stunden und Tagen ermitteln.

Im folgenden Programm wird die Differenz zwischen dem 15. Februar 1979 23:55:00 Uhr und dem 16. Februar 1979 00:05:15 Uhr berechnet.

```
# Modul time
import time

# Zwei Zeitangaben erzeugen
dztupel1 = 1979, 2, 15, 23, 55, 0, 0, 0, 0
damals1 = time.mktime(dztupel1)
print("Zeit 1:", time.strftime("%d.%m.%Y %H:%M:%S", dztupel1))

dztupel2 = 1979, 2, 16, 0, 5, 15, 0, 0, 0
damals2 = time.mktime(dztupel2)
print("Zeit 2:", time.strftime("%d.%m.%Y %H:%M:%S", dztupel2))
print()

# Differenz berechnen
print("Differenz:")

diff_sek = damals2 - damals1
print(diff_sek, "Sekunden")

diff_min = diff_sek/60
print(diff_min, "Minuten")
```

```
diff_std = diff_min/60
print(diff_std, "Stunden")
```

```
diff_tag = diff_std/24
print(diff_tag, "Tage")
```

**Listing 7.4** Datei zeit\_rechnen.py

Folgende Ausgabe wird erzeugt:

Zeit 1: 15.02.1979 23:55:00

Zeit 2: 16.02.1979 00:05:15

Differenz:

615.0 Sekunden

10.25 Minuten

0.1708333333333334 Stunden

0.00711805555555555 Tage

Zur Erläuterung:

- ▶ In der Variablen `diff_sek` wird die Differenz zwischen den beiden Zeitangaben `damals1` und `damals2` in Sekunden berechnet.
- ▶ Zur Ermittlung der Minuten wird diese Zahl durch 60 geteilt.
- ▶ Zur Ermittlung der Stunden wird wiederum dieses Ergebnis durch 60 geteilt.
- ▶ Zur Ermittlung der Tage wird das letzte Ergebnis durch 24 geteilt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Alter berechnen

Zur Berechnung des Alters einer Person wird ein anderer Weg beschritten. Dies wird im folgenden Programm gezeigt.

**Alter berechnen**

```
# Modul time
import time

# Geburtstag
dztupel = 1979, 5, 7, 0, 0, 0, 0, 0, 0
print("Geburt:", time.strftime("%d.%m.%Y", dztupel))
geburt = time.mktime(dztupel)
ltgeburt = time.localtime(geburt)

# Aktuell
ltheute = time.localtime()
print("Heute:", time.strftime("%d.%m.%Y"))
```

```
# Alter berechnen
alter = ltheute[0] - ltgeburt[0]
if ltheute[1] < ltgeburt[1] or \
    ltheute[1] == ltgeburt[1] and \
    ltheute[2] < ltgeburt[2]:
    alter = alter - 1
print("Alter:", alter)
```

**Listing 7.5 Datei zeit.Alter.py**

Die Ausgabe lautet:

**Geburt: 07.05.1979**  
**Heute: 27.11.2013**  
**Alter: 34**

Zur Erläuterung:

- ▶ Die Zeitangabe für den Geburtstag, z. B. für den 7. Mai 1979, wird mithilfe der Funktion `mktime()` erzeugt.
- ▶ Die aktuelle Zeitangabe wird mit der Funktion `localtime()` ohne Parameter erzeugt.
- ▶ Das Alter wird zunächst aus der Differenz der Jahresangaben errechnet.
- ▶ Falls die Person in diesem Jahr noch nicht Geburtstag hatte, weil
  - entweder der Geburtsmonat noch nicht erreicht wurde
  - oder innerhalb des Geburtsmonats der Geburtstag noch nicht erreicht wurde,
 so wird das Alter um 1 reduziert.

### 7.1.5 Programm anhalten

**sleep()** Die Funktion `sleep()` aus dem Modul `time` ermöglicht das Anhalten eines Programms für einen bestimmten Zeitraum.

In dem folgenden Beispielprogramm wird innerhalb einer Schleife mehrmals die aktuelle Zeit ausgegeben. Nach jedem Ausgabevorgang wird das Programm für zwei Sekunden angehalten. Am Ende wird die Zeitdifferenz zwischen Start und Ende berechnet.

```
# Modul time
import time

# Start
startzeit = time.time()
print("Start:", startzeit)
```

```
# Zeitangaben, jeweils mit Pause
for i in range(5):
    time.sleep(2)
    print(time.time())

# Ende
endzeit = time.time()
print("Ende:", endzeit)

# Abstand
differenz = endzeit-startzeit
print("Differenz:", differenz)
```

#### **Listing 7.6 Datei zeit\_anhalten.py**

Es wird die folgende Ausgabe erzeugt:

```
Start: 1385536723.396322
1385536725.424326
1385536727.436729
1385536729.511533
1385536731.523937
1385536733.53634
Ende: 1385536733.53634
Differenz: 10.140017986297607
```

Wie die Ausgabe der Differenz zeigt, lässt sich die Funktion `sleep()` nur für eine zeitliche Steuerung von begrenzter Genauigkeit einsetzen, denn

- ▶ zum einen benötigen die einzelnen Programmschritte eine eigene, wenn auch kurze Laufzeit, und
- ▶ zum anderen unterscheiden sich die Zeitabstände geringfügig.

Dennoch können Sie die Funktion `sleep()` für viele Anwendungen sinnvoll einsetzen, unter anderem im Bereich von grafischen Animationen, Simulationen oder der Spieleprogrammierung.

**Einsatzmöglichkeiten**

#### **Unterschiede in Python 2**

Die Klammern bei der Anweisung `print` entfallen.

### **7.1.6 Spiel, Version mit Zeitmessung**

In dieser Spielversion werden fünf Additionsaufgaben mit Zahlen aus dem Bereich von 10 bis 30 gestellt. Der Spieler hat pro Aufgabe nur einen Versuch. Ungültige Eingaben oder falsche Ergebnisse werden einfach mit dem Text `Falsch` kommentiert und bewertet. Am Ende wird die Gesamtzeit angegeben, die der Spieler benötigt hat.

**Zeitmessung**

```

# Module
import random, time

# Initialisierung
random.seed()
richtig = 0
startzeit = time.time()

# 5 Aufgaben
for aufgabe in range(5):
    # Aufgabe mit Ergebnis
    a = random.randint(10,30)
    b = random.randint(10,30)
    c = a + b
    print("Aufgabe", aufgabe+1, "von 5:", a, "+", b)

    # Eingabe
    try:
        zahl = int(input("Bitte eine Zahl eingeben: "))
        if zahl == c:
            print("Richtig")
            richtig += 1
        else:
            raise
    except:
        print("Falsch")

# Auswertung
endzeit = time.time()
differenz = endzeit-startzeit
print("Richtig: {0:d} von 5 in {1:.2f} Sekunden".
      format(richtig, differenz))
print("Ergebnis erzielt:"+
      time.strftime("%d.%m.%Y %H:%M:%S"))

```

**Listing 7.7 Datei spiel\_zeit.py**

Die Ausgabe lautet:

```

Aufgabe 1 von 5: 14 + 30
Bitte eine Zahl eingeben: 44
Richtig
Aufgabe 2 von 5: 17 + 11
Bitte eine Zahl eingeben: 28
Richtig
Aufgabe 3 von 5: 25 + 19
Bitte eine Zahl eingeben: 44
Richtig
Aufgabe 4 von 5: 23 + 22

```

Bitte eine Zahl eingeben: 45

Richtig

Aufgabe 5 von 5: 19 + 28

Bitte eine Zahl eingeben: 47

Richtig

Richtig: 5 von 5 in 7.63 Sekunden

Ergebnis erzielt: 27.11.2013 10:57:23

Zur Erläuterung:

- ▶ Am Anfang und am Ende wird die Zeit genommen und anschließend die **Zeitdifferenz** berechnet.
- ▶ Die Anzahl der richtigen Ergebnisse wird mitgezählt.
- ▶ Im Fall einer ungültigen Eingabe oder eines falschen Ergebnisses wird eine Ausnahme erzeugt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### 7.1.7 Spiel, objektorientierte Version mit Zeitmessung

Die objektorientierte Version des Spiels mit Zeitmessung basiert auf der objektorientierten Version ohne Zeitmessung, siehe [Abschnitt 6.10](#), »Spiel, objektorientierte Version«. Nachfolgend stelle ich nur die Erweiterungen vor. Zunächst die Import-Anweisung und das kurze Hauptprogramm:

```
import random, time

# Hauptprogramm
s = Spiel()
s.messen(True)
s.spielen()
s.messen(False)
print s
```

**Listing 7.8** Datei `spiel_zeit_oop.py`, Hauptprogramm

Zur Erläuterung:

- ▶ Es wird zusätzlich das Modul `time` für die Zeitmessung benötigt.
- ▶ Vor und nach dem eigentlichen Spielvorgang wird die Methode `messen()` zur Ermittlung der Spieldauer aufgerufen. Der Parameter vom Typ `bool` gibt an, ob es sich um den Start des Spiels handelt.

**Klasse »Aufgabe«** Die Klasse Aufgabe wurde nicht verändert. In der Klasse Spiel wurde die Methode messen() hinzugefügt und die Ausgabemethode verändert:

```
class Spiel:
    ...
    def messen(self, start):
        # Zeitmessung
        if start:
            self.startzeit = time.time()
        else:
            endzeit = time.time()
            self.zeit = endzeit - self.startzeit
    def __str__(self):
        # Ergebnis
        ausgabe = "Richtig: {0:d} von {1:d} in " \
                  "{2:.2f} Sekunden".format(self.richtig,
                                             self.anzahl, self.zeit)
        ausgabe += "\nnam " + time.strftime("%d.%m.%Y") \
                  + " um " + time.strftime("%H:%M:%S")
        return ausgabe
```

**Listing 7.9** Datei spiel\_zeit\_oop.py, Klasse »Spiel«

Zur Erläuterung:

- Zeitmessung**
- ▶ In der Methode messen() wird die Startzeit ermittelt, falls der Parameterwert True übermittelt wurde. Andernfalls werden die Endzeit und die Spieldauer ermittelt. Die Spieldauer (zeit) ist eine Eigenschaft der Klasse Spiel.
  - ▶ In der Ausgabemethode wird das Ergebnis des Spiels zusammen mit Spieldauer, Datum und Uhrzeit veröffentlicht.

### Unterschiede in Python 2

Die Klammern bei der Anweisung print entfallen. Die Funktion zur Eingabe heißt raw\_input().

## 7.2 Modul »collections«

**Datentyp deque** Das Modul collections bietet einige spezialisierte Datentypen, die als Container dienen können, unter anderem den Datentyp deque. Die Datentypen ähneln teilweise den eingebauten Container-Datentypen (wie Dictionary, Liste, Set oder Tupel), haben allerdings zusätzliche Fähigkeiten und bieten einen sehr schnellen Zugriff.

**Double-ended Queue** Die Bezeichnung des Datentyps deque steht für *double-ended Queue*. An beiden Enden eines Objekts dieses Typs können Sie Elemente hinzufügen oder entfernen. Ein Beispiel:

```

# Modul
import collections

# Erzeugen
d = collections.deque("abc")
print("Neu:", d)

# Elemente
for x in d:
    print("Element:", x)

# Hinzu links, rechts
d.appendleft(5)
d.append(25)
print("Hinzu:", d)

# Erweitern links, rechts
d.extendleft([7,9])
d.extend([17,19])
print("Erweitert:", d)

# Entfernen links, rechts
li = d.popleft()
print("Links:", li)
re = d.pop()
print("Rechts:", re)
print("Entfernt:", d)

# Rotieren
d.rotate()
print("Rotiert +1:", d)
d.rotate(-2)
print("Rotiert -2:", d)

# Leeren
d.clear()
print("Geleert:", d)

```

**Listing 7.10 Datei modul\_collections.py**

Die Ausgabe lautet:

```

Neu: deque(['a', 'b', 'c'])
Element: a
Element: b
Element: c
Hinzu: deque([5, 'a', 'b', 'c', 25])
Erweitert: deque([9, 7, 5, 'a', 'b', 'c', 25, 17, 19])
Links: 9
Rechts: 19

```

```
Entfernt: deque([7, 5, 'a', 'b', 'c', 25, 17])
Rotiert +1: deque([17, 7, 5, 'a', 'b', 'c', 25])
Rotiert -2: deque([5, 'a', 'b', 'c', 25, 17, 7])
Geleert: deque([])
```

Zur Erläuterung:

- deque()**
  - Die Funktion `deque()` liefert ein neues Objekt des Datentyps `deque`. Als Parameter wird ein iterierbares Objekt benötigt, also eine Zeichenkette, eine Liste oder Ähnliches.
  - Die einzelnen Elemente erreichen Sie mit `for ... in`.
- append()**
  - Mit `appendleft()` und `append()` können Sie einzelne Elemente am linken oder rechten Ende der Double-ended Queue hinzufügen.
- extend()**
  - `extendleft()` und `extend()` dienen zum Erweitern der Double-ended Queue um mehrere Elemente am linken bzw. am rechten Ende.
    - Die Elemente werden nacheinander in der gegebenen Reihenfolge am jeweiligen Ende angefügt.
    - Dies führt dazu, dass die hinzugefügten Elemente bei der Methode `extendleft()` in einer anderen Reihenfolge stehen.
- pop()**
  - `popleft()` und `pop()` entfernen ein Element am linken Ende bzw. am rechten Ende. Dieses Element wird jeweils als Rückgabewert geliefert.
- rotate()**
  - Mithilfe der Methode `rotate()` lassen Sie die Elemente innerhalb der Double-ended Queue zirkular rotieren.
    - Geben Sie keinen Parameter an, so wird um ein Element nach rechts rotiert.
    - Sie können auch negative Wert angeben. In diesem Fall wird nach links rotiert.
    - Ein Element, das bei der Rotation *über den Rand geschoben wird*, wird am anderen Ende wieder angefügt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 7.3 Multithreading

### Parallele Verarbeitung

Der Begriff *Multithreading* bezeichnet die Eigenschaft eines Programms, mehrere Teile parallel bearbeiten zu lassen. Diese parallel laufenden Teile eines Programms werden *Threads* genannt. Moderne Betriebssysteme erlauben Multithreading, und die Programmiersprache Python bietet die entsprechenden Routinen.

### 7.3.1 Wozu dient Multithreading?

Es gibt eine Reihe von Problemstellungen, bei denen sich Multithreading als nützlich erweist, z.B. Simulationen von realen Prozessen, bei denen eine Aktion eine weitere Aktion anstößt, danach aber selbst auch weiterläuft. Die angestoßene Aktion kann wiederum eine dritte Aktion auslösen usw. Alle Aktionen greifen auf die gleichen Umgebungsdaten zu, verändern sie gegebenenfalls, enden zu verschiedenen Zeitpunkten und mit unterschiedlichen Ergebnissen.

Einsatzmöglichkeiten

Es kann sich auch um GUI-Anwendungen handeln, bei denen rechenintensive Programmteile im Hintergrund laufen (*ausgelagert werden*) oder grafisch permanent die Ergebnisse von Messwertauswertungen oder anderen Echtzeitprozessen dargestellt werden.

### 7.3.2 Erzeugung eines Threads

Das Modul `_thread` bietet eine einfache Möglichkeit zum Multithreading. Zur Erzeugung eines Threads in Python wird die Funktion `start_new_thread()` benötigt. Diese Funktion verlangt zwei Parameter:

`start_new_thread()`

- ▶ den Namen einer Funktion, die im Thread ausgeführt werden soll
- ▶ ein Tupel mit Parametern für diese Funktion; hat die Funktion keine Parameter, muss ein leeres Tupel angegeben werden

Im folgenden Programm wird aus dem Hauptprogramm ein neuer Thread gestartet. Anschließend wird das Hauptprogramm für 10 Sekunden angehalten und dann beendet. Beginn und Ende des Hauptprogramms werden angezeigt.

In dem Thread läuft die Funktion `show()`, in der insgesamt fünfmal die aktuelle Zeit angezeigt wird. Nach jeder Anzeige wird das Programm für 1,5 Sekunden angehalten. Beginn und Ende des Threads werden ebenfalls angezeigt.

```
# Module
import time, _thread
# Thread-Funktion
def show():
    print("Start Thread")
    for i in range(5):
        print(i, time.time())
        time.sleep(1.5)
    print("Ende Thread")
    return
# Hauptprogramm
print("Start Hauptprogramm:", time.time())
```

```
_thread.start_new_thread(show,())
time.sleep(10)
print("Ende Hauptprogramm:", time.time())
```

**Listing 7.11** Datei `thread_ergeben.py`

Es wird die Ausgabe erzeugt:

```
Start Hauptprogramm: 1385536993.970698
Start Thread
0 1385536993.986298
1 1385536995.5151
2 1385536997.028303
3 1385536998.528406
4 1385537000.057208
Ende Thread
Ende Hauptprogramm: 1385537003.988415
```

Zur Erläuterung:

- Hauptprogramm** ▶ Das Hauptprogramm ist ca. 10 Sekunden gelaufen.
- Thread** ▶ Parallel dazu lief ein Thread, der ungefähr alle 1,5 Sekunden eine Ausgabe macht.
- ▶ Bei den hier vorgenommenen Zeitangaben endet das Hauptprogramm erst, nachdem der parallel laufende Thread bereits beendet ist.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Das Modul heißt `thread` statt `_thread`, also ohne führenden Unterstrich.

### 7.3.3 Identifizierung eines Threads

**get\_ident()** Zur Unterscheidung der Auswirkungen unterschiedlicher Threads hat jeder Thread eine eindeutige Identifikationsnummer (ID), die Sie über die Funktion `get_ident()` ermitteln. Das Hauptprogramm ist ebenfalls ein Thread – auch dieser *Haupt-Thread* verfügt über eine ID.

Das vorherige Programm wurde erweitert. Aus dem Haupt-Thread werden, zeitlich versetzt, insgesamt zwei Threads mit der Funktion `show()` gestartet. Für jeden Thread, auch für den Haupt-Thread, wird die ID ermittelt und bei jeder Ausgabe zusätzlich angezeigt.

```
# Module
import time, _thread

# Thread-Funktion
def show():
```

```

id = _thread.get_ident()
print("Start Thread", id)
for i in range(5):
    print(i, "Thread", id)
    time.sleep(1.5)
print("Ende Thread", id)
return

# Hauptprogramm
id = _thread.get_ident()
print("Start Hauptprogramm", id)
_thread.start_new_thread(show,())
time.sleep(0.5)
_thread.start_new_thread(show,())
time.sleep(10)
print("Ende Hauptprogramm", id)

```

**Listing 7.12** Datei `thread_ident.py`

Das Programm erzeugt z. B. die Ausgabe:

```

Start Hauptprogramm 3752
Start Thread 3136
0 Thread 3136
Start Thread 1612
0 Thread 1612
1 Thread 3136
1 Thread 1612
2 Thread 3136
2 Thread 1612
3 Thread 3136
3 Thread 1612
4 Thread 3136
4 Thread 1612
Ende Thread 3136
Ende Thread 1612
Ende Hauptprogramm 3752

```

Zur Erläuterung:

- ▶ Der Haupt-Thread hat im vorliegenden Programmlauf die ID 3752, der erste Unter-Thread die ID 3136, der zweite Unter-Thread die ID 1612. Den parallelen Ablauf der Threads können Sie anhand der IDs beobachten.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Das Modul heißt `thread` statt `_thread`.

### 7.3.4 Gemeinsame Objekte

#### Globale Variablen

Alle Threads haben Zugriff auf alle globalen Daten und Objekte des übergeordneten Threads. Im folgenden Beispiel wird gezeigt, wie eine Variable vom Haupt-Thread und zwei weiteren Threads gemeinsam genutzt und verändert wird.

```
# Module
import time, _thread

# Thread-Funktion
def show():
    global counter
    id = _thread.get_ident()
    for i in range(5):
        counter += 1
        print(i, id, counter)
        time.sleep(1.5)
    return

# Hauptprogramm
id = _thread.get_ident()
counter = 0
print(id, counter)

_thread.start_new_thread(show,())
time.sleep(0.5)
_thread.start_new_thread(show,())
time.sleep(10)
counter += 1
print(id, counter)
```

**Listing 7.13** Datei `thread_gemeinsam.py`

Die Ausgabe lautet z. B.:

```
512 0
0 3876 1
0 2672 2
1 3876 3
1 2672 4
2 3876 5
2 2672 6
3 3876 7
3 2672 8
4 3876 9
4 2672 10
512 11
```

Zur Erläuterung:

- ▶ Die Variable `counter` ist global im Haupt-Thread (hier: ID 512) und wird dort mit 0 vorbesetzt.
- ▶ Sie wird mithilfe des Schlüsselworts `global` in der Thread-Funktion bekannt gemacht, um 1 erhöht und angezeigt.
- ▶ In beiden Unter-Threads (hier: ID 3876 und ID 2672) ist die Variable bekannt und kann verändert werden.
- ▶ Am Ende wird sie auch im Haupt-Thread um 1 erhöht und angezeigt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Das Modul heißt `thread` statt `_thread`.

## 7.3.5 Threads und Exceptions

Tritt eine unbehandelte Ausnahme auf, so betrifft dies nur den Thread, in dem sie auftritt. Weder der aufrufende Haupt-Thread noch die anderen Threads werden dadurch gestört.

Laufzeitfehler  
nur im Thread

Das vorherige Programm wurde erweitert. Falls die Variable `counter` den Wert 5 hat, so wird die unbehandelte Ausnahme `ZeroDivisionError` künstlich hervorgerufen. Der Thread, in dem dies geschieht, wird unmittelbar beendet. Die anderen Threads – auch der Haupt-Thread – laufen unbeeinflusst zu Ende. Das geänderte Programm sieht wie folgt aus:

```
# Module
import time, _thread

# Thread-Funktion
def show():
    global counter
    id = _thread.get_ident()
    for i in range(5):
        counter += 1
        print(i, id, counter)

        # Division durch 0
        if counter == 5:
            erg = 1/0
            time.sleep(1.5)
    return

# Hauptprogramm
id = _thread.get_ident()
counter = 0
```

```

print(id, counter)

_thread.start_new_thread(show,())
time.sleep(0.5)
_thread.start_new_thread(show,())
time.sleep(10)

counter += 1
print(id, counter)

```

**Listing 7.14** Datei `thread_ausnahme.py`

Es wird z. B. die folgende Ausgabe erzeugt:

```

1832 0
0 324 1
0 2372 2
1324 3
12372 4
2 324 5
Unhandled exception in thread started
  by <function show at 0x023D55D0>
Traceback (most recent call last):
  File "C:\Python34\thread_ausnahme.py", line 14, in show
    erg = 1/0
ZeroDivisionError: division by zero
2 2372 6
3 2372 7
4 2372 8
1832 9

```

Zur Erläuterung:

- ▶ Der Thread mit der ID 324 endet aufgrund der unbehandelten Ausnahme.
- Der Thread mit der ID 2372 läuft regulär zu Ende, ebenso der Haupt-Thread mit der ID 1832.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Das Modul heißt `thread` statt `_thread`. In der Ausgabe steht `C:\Python27`.

## 7.4 Reguläre Ausdrücke

### Suchen und Ersetzen

Reguläre Ausdrücke erleichtern das Suchen und Ersetzen von bestimmten Texten oder Teilstexten oder von einzelnen Zeichen. Sie kommen z. B. bei der Kontrolle und Auswertung von Benutzereingaben oder bei der Suche nach Dateien zum Einsatz.

Wenn festgestellt wird, dass ein Benutzer eine falsche oder unvollständige Eingabe gemacht hat, so kann dies zu einer Hilfestellung und einer erneuten Eingabeaufforderung führen. In Python wird das Modul `re` zum Arbeiten mit regulären Ausdrücken benötigt.

**Modul re**

Die beiden folgenden Programmbeispiele sollen einen Teil der umfangreichen Möglichkeiten von regulären Ausdrücken verdeutlichen.

### 7.4.1 Suchen von Teiltexten

Zum Suchen von Teiltexten wird hier die Methode `findall()` eingesetzt. Sie findet alle Teiltexte, die zum regulären Ausdruck passen, und liefert eine Liste dieser Teiltexte zur weiteren Auswertung zurück.

**findall()**

```
# Modul
import re

# 1: Exakter Text
tx = "Haus und Maus und Laus"
print(tx)
erg = re.findall("Maus",tx)
print("1: ", erg)
# 2: Wahl zwischen bestimmten Zeichen
erg = re.findall("[HM]aus",tx)
print("2: ", erg)

# 3: Alle Buchstaben aus Bereich
erg = re.findall("[L-M]aus",tx)
print("3: ", erg)

# 4: Alle Buchstaben nicht aus Bereich
erg = re.findall("[^L-M]aus",tx)
print("4: ", erg)

# 5: Beliebiges Zeichen
erg = re.findall(".aus",tx)
print("5: ", erg)

# 6: Suchbegriff nur am Anfang des Textes
erg = re.findall("^aus",tx)
print("6: ", erg)

# 7: Suchbegriff nur am Ende des Textes
erg = re.findall("aus$",tx)
print("7: ", erg)
print()
```

```

# 8: alle Ziffern aus Bereich
tx = "0172-445633"
print(tx)
erg = re.findall("[0-2]",tx)
print("8: ", erg)

# 9: alle Zeichen nicht aus Ziffernbereich
erg = re.findall("[^0-2]",tx)
print("9: ", erg)

# 10: alle Zeichen oder Ziffern, die angegeben sind
erg = re.findall("[047-]",tx)
print("10: ", erg)
print()

# 11: Wiederholung, beliebig oft
tx = "aa und aba und abba und abbba und aca"
print(tx)
erg = re.findall("ab*a",tx)
print("11: ", erg)
# 12: Wiederholung, 1 oder mehr
erg = re.findall("ab+a",tx)
print("12: ", erg)

# 13: Wiederholung, 0 oder 1
erg = re.findall("ab?a",tx)
print("13: ", erg)

# 14: Wiederholung, m bis n
erg = re.findall("ab{2,3}a",tx)
print("14: ", erg)

# 15: Wiederholung der max. Menge von Zeichen
tx = "aa und aba und abba und aca und addda"
erg = re.findall("a.*a",tx)
print("15: ", erg)

# 16: Wiederholung der min. Menge von Zeichen
erg = re.findall("a.*?a",tx)
print("16: ", erg)

```

**Listing 7.15 Datei regexp\_suchen.py**

Die Ausgabe dieses Programms:

**Haus und Maus und Laus**  
**1: ['Maus']**  
**2: ['Haus', 'Maus']**  
**3: ['Maus', 'Laus']**

- 4: ['Haus']  
 5: ['Haus', 'Maus', 'Laus']  
 6: ['Haus']  
 7: ['Laus']

- 0172–445633  
 8: ['0', '1', '2']  
 9: ['7', '–', '4', '4', '5', '6', '3', '3']  
 10: ['0', '7', '–', '4', '4']

**aa und aba und abba und abbb a und aca**

- 11: ['aa', 'aba', 'abba', 'abbb']  
 12: ['aba', 'abba', 'abbb']  
 13: ['aa', 'aba']  
 14: ['abba', 'abbb']  
 15: ['aa und aba und abba und aca und addda']  
 16: ['aa', 'aba', 'abba', 'aca', 'adda']

Zur Erläuterung:

1. Basis für die ersten sieben regulären Ausdrücke ist der Text Haus und Maus und Laus. Im ersten Beispiel wird der exakte Begriff Maus gesucht. Alle Vorkommen dieses Begriffs werden geliefert.
2. Es werden die Teiltexte gesucht, die mit einem H oder einem M beginnen und mit aus enden. In den eckigen Klammern steht: Suche wahlweise H oder M. Gefunden werden Haus und Maus, aber nicht Laus. Eckige Klammern
3. Es werden die Teiltexte gesucht, die mit einem der Zeichen von L bis M beginnen und mit aus enden. Die eckigen Klammern geben einen Bereich an. Gefunden werden Maus und Laus, aber nicht Haus.
4. Es werden die Teiltexte gesucht, die nicht mit einem der Zeichen von L bis M beginnen, aber mit aus enden. Das Zeichen ^ bedeutet im Zusammenhang mit einem Bereich eine logische Verneinung. Gefunden wird Haus, aber nicht Maus und Laus. Zeichen ^, nicht
5. Es werden die Teiltexte gesucht, die mit einem beliebigen Zeichen beginnen und mit aus enden. Das Zeichen . (Punkt) bedeutet: beliebiges Zeichen. Gefunden werden Haus, Maus und Laus. Zeichen .
6. Es werden die Teiltexte gesucht, die mit einem beliebigen Zeichen beginnen und mit aus enden. Dies gilt allerdings nur für Teiltexte, die zu Beginn des untersuchten Texts stehen. Das Zeichen ^ bedeutet in diesem Zusammenhang: zu Beginn. Gefunden wird nur Haus. Zeichen ^, Beginn
7. Es werden die Teiltexte gesucht, die mit einem beliebigen Zeichen beginnen und mit aus enden. Dies gilt allerdings nur für Teiltexte, die am Ende des untersuchten Texts stehen. Das Zeichen \$ bedeutet in diesem Zusammenhang: am Ende. Gefunden wird nur Laus. Zeichen \$

- Suche nach Ziffern**
8. Basis für die nächsten drei regulären Ausdrücke ist der Text 0172–445633. Es wird als Teilstext eine der Ziffern von 0 bis 2 gesucht. Zusammenhängende Bereiche können sich also auch auf Ziffern beziehen. Gefunden werden die Ziffern 0, 1 und 2.
  9. Es wird als Teilstext ein Zeichen gesucht, das *nicht* im Ziffernbereich von 0 bis 2 liegt. Gefunden werden alle Ziffern außerhalb von 0 bis 2 sowie alle Nichtziffern.
  10. Es wird als Teilstext eines der Zeichen aus der angegebenen Menge von Zeichen gesucht. Gefunden werden alle Zeichen sowie Ziffern aus dem Bereich 0, 4, 7 oder -.
- Zeichen \***
11. Basis für die letzten sechs regulären Ausdrücke ist der Text aa und aba und abba und abbba und aca. An diesem Text soll im Vergleich das Verhalten bei Wiederholungen von Zeichen verdeutlicht werden. Gefunden werden alle Teilstexte, die aus folgender Zeichenfolge bestehen: ein a, eine beliebige Anzahl (auch 0 möglich) des Zeichens b, wiederum ein a. Der \* (Stern) bedeutet: beliebige Wiederholungszahl.
- Zeichen +**
12. Gefunden werden alle Teilstexte, die aus folgender Zeichenfolge bestehen: ein a, mindestens ein b, wiederum ein a. Das + (Plus) bedeutet: beliebige Anzahl von Wiederholungen, mindestens eine.
- Zeichen ?**
13. Gefunden werden alle Teilstexte, die aus folgender Zeichenfolge bestehen: ein a, kein oder ein b, wiederum ein a. Das Fragezeichen ? bedeutet in diesem Zusammenhang: keine oder eine Wiederholung.
- Geschweifte Klammern**
14. Gefunden werden alle Teilstexte, die aus folgender Zeichenfolge bestehen: ein a, zweimal oder dreimal das Zeichen b, wiederum ein a. Die geschweiften Klammern bieten die Möglichkeit, die gewünschte Anzahl an Wiederholungen anzugeben.
  15. Gefunden werden alle Teilstexte, die aus folgender Zeichenfolge bestehen: ein a, anschließend so viele beliebige Zeichen wie möglich, wiederum ein a. Da der durchsuchte Text mit einem a beginnt und einem a endet, wird genau eine Zeichenfolge gefunden.
  16. Gefunden werden alle Teilstexte, die aus folgender Zeichenfolge bestehen: ein a, anschließend so wenig beliebige Zeichen wie möglich, wiederum ein a. Es wird eine ganze Reihe von Zeichenfolgen gefunden, die dazu passen.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 7.4.2 Ersetzen von Teiltexten

Die Methode `sub()` ersetzt alle Teiltexte, die zum regulären Ausdruck passen, durch einen anderen Text. Zur deutlicheren Darstellung werden in diesem Programm die gleichen regulären Ausdrücke eingesetzt wie im vorherigen Programm. Alle Teiltexte, die gemäß dem Ausdruck gefunden werden, werden durch `x` ersetzt.

```
# Modul
import re
# 1: Exakter Text
tx = "Haus und Maus und Laus"
print(tx)
txneu = re.sub("Maus", "x", tx)
print("1: ", txneu)
# 2: Wahl zwischen bestimmten Zeichen
txneu = re.sub("[H|M]aus", "x", tx)
print("2: ", txneu)
# 3: alle Buchstaben aus Bereich
txneu = re.sub("[L-M]aus", "x", tx)
print("3: ", txneu)
# 4: alle Buchstaben nicht aus Bereich
txneu = re.sub("[^L-M]aus", "x", tx)
print("4: ", txneu)
# 5: Beliebiges Zeichen
txneu = re.sub(".aus", "x", tx)
print("5: ", txneu)
# 6: Suchbegriff nur am Anfang des Textes
txneu = re.sub("^aus", "x", tx)
print("6: ", txneu)
# 7: Suchbegriff nur am Ende des Textes
txneu = re.sub("aus$", "x", tx)
print("7: ", txneu)
print()
# 8: alle Ziffern aus Bereich
tx = "0172-445633"
print(tx)
txneu = re.sub("[0-2]", "x", tx)
print("8: ", txneu)
# 9: alle Zeichen nicht aus Ziffernbereich
txneu = re.sub("[^0-2]", "x", tx)
print("9: ", txneu)
# 10: alle Zeichen oder Ziffern, die angegeben sind
txneu = re.sub("[047-]", "x", tx)
print("10: ", txneu)
print()
```

`sub()`

```
# 11: Wiederholung, beliebig oft
tx = "aa und aba und abba und abbba und aca"
print(tx)
txneu = re.sub("ab*a","x",tx)
print("11: ", txneu)
# 12: Wiederholung, 1 oder mehr
txneu = re.sub("ab+a","x",tx)
print("12: ", txneu)
# 13: Wiederholung, 0 oder 1
txneu = re.sub("ab?a","x",tx)
print("13: ", txneu)
# 14: Wiederholung, m bis n
txneu = re.sub("ab{2,3}a","x",tx)
print("14: ", txneu)
# 15: Wiederholung der max. Menge von Zeichen
tx = "aa und aba und abba und aca und addda"
txneu = re.sub("a.*a","x",tx)
print("15: ", txneu)
# 16: Wiederholung der min. Menge von Zeichen
txneu = re.sub("a.*?a","x",tx)
print("16: ", txneu)
```

**Listing 7.16** Datei `regexp_ersetzen.py`

Die Ausgabe dieses Programms sieht wie folgt aus:

```
Haus und Maus und Laus
1: Haus und x und Laus
2: x und x und Laus
3: Haus und x und x
4: x und Maus und Laus
5: x und x und x
6: x und Maus und Laus
7: Haus und Maus und x
0172-445633
8: xx7x-445633
9: 01x2xxxxxx
10: x1x2xxx5633
aa und aba und abba und abbba und aca
11: x und x und x und x und aca
12: aa und x und x und x und aca
13: x und x und abba und abbba und aca
14: aa und aba und x und x und aca
15: x
16: x und x und x und x
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 7.5 Audio-Ausgabe

Das Modul `winsound` bietet unter Windows die Möglichkeit zur Ausgabe von Systemtönen und WAV-Dateien. Ein kleines Beispiel:

```
import winsound, time

# Folge von Tönen mit unterschiedlicher Frequenz
for i in range (600, 1500, 200):
    print(i)
    winsound.Beep(i, 500)
    time.sleep(0.2)

# Beispiel für Systemton
print("SystemQuestion")
winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS)

# Beispiel für WAV-Datei
print("GAKkord.wav")
winsound.PlaySound("GAKkord.wav", winsound.SND_FILENAME)
```

**Listing 7.17** Datei `sound.py`

Zur Erläuterung:

- ▶ Die Funktion `Beep()` sendet ein Tonsignal. Der erste Parameter steht für die Frequenz, der zweite Parameter für die Dauer des Tonsignals.
- ▶ Mithilfe der Funktion `PlaySound()` können Windows-Systemtöne oder WAV-Dateien abgespielt werden. Dabei steht der erste Parameter für den Namen des Systemtons oder den Namen der WAV-Datei. Beim zweiten Parameter handelt es sich um eine Konstante. Im Falle eines Systemtons muss `SND_ALIAS` angegeben werden. Bei einer WAV-Datei ist es `SND_FILENAME`.
- ▶ In [Abschnitt 8.9](#), »Beispielprojekt Morsezeichen«, sehen Sie ein weiteres Beispiel für die Nutzung des Moduls `winsound`. Dabei geht es um die Umsetzung eines Textes in Morsecode, der als Tonsignal per Lautsprecher ausgegeben wird.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Unterschiede in Ubuntu Linux und OS X

Das Modul `winsound` steht nicht zur Verfügung. Unter Ubuntu Linux könnten Sie mit dem Paket `Beep` arbeiten, das vom Betriebssystem aus aufgerufen wird.



# Kapitel 8

## Dateien

Die dauerhafte Speicherung von Daten kann in einfachen Dateien oder in Datenbanken erfolgen. In diesem Abschnitt lernen Sie verschiedene Methoden zur Speicherung von Daten in Dateien kennen.

Abschließend stelle ich Ihnen das bereits bekannte Spiel in einer Version vor, die das Abspeichern von Spielergebnissen in einer Highscore-Datei ermöglicht.

Highscore speichern

### 8.1 Dateitypen

Bei der Ein- und Ausgabe von Daten in Dateien sollten Sie wissen, welcher Dateityp vorliegt und welche Zugriffsart Sie verwenden können. Wir unterscheiden zwischen folgenden Zugriffsarten:

- ▶ **Sequentieller Zugriff:** Diese Möglichkeit wird bei einer Datei bevorzugt, deren einzelne Zeilen unterschiedlich lang sind und jeweils mit einem Zeilenumbruch beendet werden. Der Inhalt der Datei kann mit einem einfachen Editor bearbeitet werden. Die Zeilen werden rein sequentiell gelesen und geschrieben. Der direkte Zugriff auf eine bestimmte Zeile ist nicht möglich, da die Länge der Vorgängerzeilen nicht bekannt ist.  
**Sequentiell**
- ▶ **Wahlfreier Zugriff:** Diese Möglichkeit haben Sie bei einer Datei, die Datensätze enthält. Zeilenumbrüche können, müssen aber nicht existieren. Die Länge und Struktur eines Datensatzes sollte bekannt sein oder innerhalb der Datei an einer vereinbarten Stelle stehen. Die Zeilen können direkt gelesen und verändert werden, da man den Ort jedes Datensatzes berechnen kann.  
**Wahlfrei**
- ▶ **Binärer Zugriff:** Diese Zugriffsmöglichkeit steht für alle Dateitypen zur Verfügung. Sie arbeiten mit den reinen Bytefolgen; diese können Sie mithilfe eines darauf angepassten Python-Programms lesen oder verändern. Allerdings kann dies zur Folge haben, dass die Dateien nicht mehr mit den zugehörigen Anwendungsprogrammen gelesen werden können. Überschreiben Sie beispielsweise in einer Oracle-Datenbank die Stelle, an der die Anzahl der Datensätze einer bestimmten Tabelle steht, so kann dies dazu führen, dass diese Tabelle oder mehrere Tabellen zerstört werden.  
**Binär**

Ohne Kenntnis der Struktur einer Datei ist es daher nicht möglich, sie korrekt zu bearbeiten. Außer den genannten Typen gibt es auch Mischformen.

Dateistruktur  
bekannt

## 8.2 Öffnen und Schließen einer Datei

- open()** Eine Datei, die bearbeitet werden soll, muss vorher geöffnet werden. Dies geschieht mithilfe der eingebauten Funktion `open()`. Dabei werden der Name der Datei (eventuell mit Pfadangabe) und der Öffnungsmodus angegeben.
- Pfad zur Datei** Es wird davon ausgegangen, dass sich die zu öffnende Datei im gleichen Verzeichnis wie das Python-Programm befindet. Andernfalls müssen Sie den absoluten oder relativen Pfad zur Datei angeben. Der relative Pfad kann wie folgt angegeben werden:

Beschreibung	Pfad
zur Datei <code>ein.txt</code> im Unterverzeichnis <code>unt</code>	<code>unt/ein.txt</code>
zur Datei <code>ein.txt</code> im übergeordneten Verzeichnis	<code>../ein.txt</code>
zur Datei <code>ein.txt</code> im parallelen Verzeichnis <code>neb</code>	<code>../neb/ein.txt</code>

Tabelle 8.1 Relativer Pfad

### Hinweis

Pfad mit / Bei Python unter Windows sind sowohl der Forward-Slash (/) als auch der Back-Slash (\) zum Verzeichniswechsel erlaubt. Häufig wird ein Dateizugriff im Zusammenhang mit Internet-Server-Programmierung genutzt. Im Internet herrschen Unix-Server vor, daher stelle ich hier die Unix-freundliche Variante mit / (Forward-Slash) vor.

Einige Öffnungsmodi:

Modus	Erläuterung
r	Öffnen zum Lesen (kann auch weggelassen werden, Standard)
w	Öffnen zum (Über-)Schreiben
a	Öffnen zum zusätzlichen Schreiben (Anhängen) am Ende der Datei
r+	Öffnen zum Lesen und Schreiben, aktuelle Position am Anfang
w+	Öffnen zum Lesen und Schreiben, Datei wird geleert

Tabelle 8.2 Öffnungsmodi

Modus	Erläuterung
a+	Öffnen zum Lesen und Schreiben, aktuelle Position am Ende
b	Eine zusätzliche Angabe, sie dient zum Öffnen einer Datei, die gelesen oder geschrieben werden soll; im Binärmodus; siehe auch <u>Abschnitt 8.5, »Serialisierung«.</u>

**Tabelle 8.2** Öffnungsmodi (Forts.)

Rückgabewert der Funktion `open()` ist ein Dateiobjekt, das für den weiteren Zugriff auf die Datei verwendet wird.

Dateiobjekt

In jedem Dateiobjekt ist die aktuelle Zugriffsposition gespeichert. Dies ist die Position, an der aktuell gelesen oder geschrieben wird. Sie verändert sich mit jedem Lese- oder Schreibvorgang. Außerdem kann sie mit der Funktion `seek()` verändert werden, ohne lesen und schreiben zu müssen.

`seek()`

Nach der Bearbeitung der Datei muss diese mit der Funktion `close()` geschlossen werden. Wird die Datei nicht geschlossen, ist sie eventuell für weitere Zugriffe gesperrt.

`close()`

## 8.3 Sequentielle Dateien

Dateien können sequentiell beschrieben oder gelesen werden.

### 8.3.1 Sequentielles Schreiben

Zum Schreiben in eine Datei muss die Datei zunächst mit der Funktion `open()` und dem Öffnungsmodus "w" geöffnet werden. Wenn die Datei noch nicht existiert, wird sie in diesem Moment erzeugt. Achtung: Wenn sie bereits existiert, wird sie ohne Vorwarnung überschrieben!

Modus »w«

Beim Öffnen einer Datei im Schreibmodus kann ein Laufzeitfehler auftreten (z. B. beim Schreiben auf ein schreibgeschütztes Medium oder ein nicht vorhandenes Laufwerk). Daher sollten Sie hier eine Ausnahmebehandlung mit einem vorzeitigen Programmabbruch durchführen. Dazu können Sie die Methode `exit()` aus dem Modul `sys` nutzen.

Datei schreiben

Programm  
abbrechen

Falls das Programm nicht vorzeitig beendet wurde, können Sie mit der Funktion `write()` einzelne Strings und mit der Funktion `writelines()` eine Liste von Strings in die Datei schreiben. Zahlen können Sie mithilfe der Funktion `str()` in einen String umwandeln. Das Zeichen `\n` (*new line*) für das Zeilenende müssen Sie jeweils hinzufügen.

`write(),  
writelines()`

```

import sys

# Zugriffsversuch
try:
    d = open("schreiben.txt","w")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Schreiben von einzelnen Strings, mit Zeilenende
d.write("Die erste Zeile\n")
for i in range(2,11,2):
    d.write(str(i) + " ")
d.write("\n")

# Schreiben einer Liste
x = ["abc\n", str(12/7.5)+"\n", "xyz\n"]
d.writelines(x)

# Schliessen der Datei
d.close()

```

**Listing 8.1** Datei schreiben.py

Die Datei *schreiben.txt* hat anschließend den folgenden Inhalt:

```

Die erste Zeile
2 4 6 8 10
abc
1.6
xyz

```

**Listing 8.2** Datei schreiben.txt

Zur Erläuterung:

- write()**
  - ▶ Zunächst wird eine einzelne Textzeile mit der Funktion `write()` in die Datei geschrieben.
  - ▶ Anschließend werden die Zahlen von 2 bis 10 in Schritten von 2 in Strings umgewandelt und in die Datei geschrieben.
  - ▶ Nach jeder Ausgabezeile wird ein Zeilenende zusätzlich ausgegeben.
- writelines()**
  - ▶ Die Liste `x` wird erzeugt. Die Elemente sind Strings, die mit dem Zeilenende-Zeichen enden. Diese Liste wird mit der Funktion `writelines()` in die Datei geschrieben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

**Hinweis**

Verwenden Sie beim Öffnen der Datei den Öffnungsmodus "a" statt "w" (`d = open("schreiben.txt", "a")`), so werden die Ausgaben an den bisherigen Dateiinhalt angehängt. Die Datei wird also immer größer.

**Modus »a«****8.3.2 Sequentielles Lesen**

Die Zeilen einer sequentiellen Datei können Sie mit den Funktionen `readlines()`, `readline()` oder `read()` lesen:

- ▶ Die Funktion `readlines()` liest alle Zeilen der Datei in eine Liste von Strings.
- ▶ Die Funktion `readline()` liest jeweils eine einzelne Zeile einer Datei in einen String.
- ▶ Die Funktion `read()` liest alle Zeilen der Datei in einen String.

**Datei lesen****`readlines()`****`readline()`****`read()`****Beispiel 1**

Zunächst werden alle Zeilen einer Datei mithilfe der Funktion `readlines()` in eine Liste gelesen:

```
import sys

# Zugriffsversuch
try:
    d = open("lesen.txt")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Lesen aller Zeilen in eine Liste
allezeilen = d.readlines()

# Schliessen der Datei
d.close()

# Ausgabe und Summierung der Listenelemente
summe = 0
for zeile in allezeilen:
    print(zeile, end="")
    summe += float(zeile)

# Ausgabe der Summe
print("Summe:", summe)
```

**Listing 8.3 Datei lesen\_all\_zeilen.py**

Die Datei *lesen.txt* hat folgenden Inhalt:

```
6
2.5
4
```

**Listing 8.4** Datei *lesen.txt*

Die Ausgabe des Programms lautet:

```
6
2.5
4
Summe: 12.5
```

Zur Erläuterung:

- ▶ `open()` ▶ Zunächst müssen Sie die Datei mithilfe der Funktion `open()` zum Lesen öffnen. Den Öffnungsmodus können Sie in diesem Fall weglassen.
- ▶ Gelingt das Öffnen der Datei nicht, so tritt eine Ausnahme auf. Dies geschieht beispielsweise, wenn
  - die Datei nicht existiert,
  - der Name der Datei falsch geschrieben wurde oder
  - sich die Datei in einem anderen Verzeichnis befindet.
- ▶ `readlines()` ▶ Die Funktion `readlines()` wird auf das Dateiobjekt angewendet und ergibt eine Liste von Strings. Jeder String enthält eine Zeile inklusive dem Zeichen für Zeilenende. Die Strings können unterschiedlich lang sein.
- ▶ `close()` ▶ Nach dem Lesen wird die Datei mit `close()` geschlossen.
- ▶ Alle Elemente der Liste können mit einer `for`-Schleife bearbeitet werden. Im vorliegenden Beispiel werden sie auf dem Bildschirm ausgegeben und in Zahlen umgewandelt. Da das Zeichen für das Zeilenende bereits vorhanden ist, sollte bei der Ausgabe kein zusätzliches Zeilenende-Zeichen ausgegeben werden, da sonst unnötige Leerzeilen entstehen.
- ▶ Die Elemente der Liste werden mithilfe der Funktion `float()` in Zahlen umgewandelt und anschließend summiert.
- ▶ Die Summe wird ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Da jede Zeile bereits das Zeilenende-Zeichen enthält, muss dieses abgeschnitten werden. Die entsprechende Programmzeile lautet daher: `print zeile[0:len(zeile)-1]` statt `print(zeile, end="")`.

## Beispiel 2

Im folgenden Programm werden einzelne Zeilen einer Datei mithilfe der Funktion `readline()` gelesen:

```
import sys

# Zugriffsversuch
try:
    d = open("lesen.txt")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Lesen und Ausgabe einzelner Zeilen
zeile1 = d.readline()
print(zeile1, end="")
zeile2 = d.readline()
print(zeile2, end="")

# Summierung und Ausgabe
summe = float(zeile1) + float(zeile2)
print("Summe:", summe)

# Schliessen der Datei
d.close()
```

**Listing 8.5** Datei `lesen_einzelzeile.py`

Die Ausgabe des Programms lautet:

```
6
2.5
Summe: 8.5
```

Zur Erläuterung:

- ▶ Nach dem erfolgreichen Öffnen der Datei ist die aktuelle Lese position der Anfang der ersten Zeile.
- ▶ Die Funktion `readline()` ergibt einen String, der eine Zeile inklusive dem Zeichen für Zeilenende enthält.
- ▶ Nach einem Aufruf von `readline()` ist die aktuelle Lese position der Anfang der nächsten Zeile. Daher wird beim zweiten Aufruf die zweite Zeile gelesen usw.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Da jede Zeile bereits das Zeilenende-Zeichen enthält, muss dieses abgeschnitten werden. Die entsprechende Programmzeile lautet daher: `print zeile1[0:len(zeile1)-1]` statt `print(zeile1, end="")`. Entsprechendes gilt für `zeile2`.

### Beispiel 3

Auch mit der Funktion `readline()` können Sie alle Zeilen einer Datei lesen. Dazu ist es notwendig, das Ende der Datei zu erkennen. Zu diesem Zweck können Sie den von `readline()` zurückgelieferten String untersuchen. Ein Beispielprogramm:

```
import sys

# Zugriffsversuch
try:
    d = open("lesen.txt")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Lesen, Ausgabe und Summierung aller Zeilen
summe = 0
zeile = d.readline()
while zeile:
    summe += float(zeile)
    print(zeile, end="")
    zeile = d.readline()

# Ausgabe der Summe
print("Summe:", summe)

# Schliessen der Datei
d.close()
```

**Listing 8.6** Datei lesen\_ende.py

Die Ausgabe des Programms:

```
6
2.5
4
Summe: 12.5
```

Zur Erläuterung:

- ▶ Über eine `while`-Schleife wird das Lesen der Zeilen gesteuert.
- ▶ Wenn das Dateiende noch nicht erreicht wurde, ist der String `z` nicht leer. Die Bedingung `while z` ist somit wahr, und die Schleife wird weiter ausgeführt.
- ▶ Wurde das Dateiende erreicht, so ist der String `z` leer. Die Bedingung `while z` ist somit falsch, und das Programm beendet die Schleife.

Dateiende erkennen

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Da jede Zeile bereits das Zeilenende-Zeichen enthält, muss dieses abgeschnitten werden. Die entsprechende Programmzeile lautet daher `print zeile[0:len(zeile)-1]` statt `print(zeile, end="")`.

### Beispiel 4

Die Funktion `read()` liest den gesamten Inhalt einer Datei in einen String. Dieser muss anhand des Zeilenende-Zeichens in einzelne Zeilen zerlegt werden. Ein Beispielprogramm:

```
import sys

# Zugriffsversuch
try:
    d = open("lesen.txt")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Lesen des gesamten Texts
gesamtertext = d.read()

# Schliessen der Datei
d.close()

# Umwandeln in eine Liste
zeilenliste = gesamtertext.split(chr(10))

# Summieren und Ausgeben
summe = 0
for zeile in zeilenliste:
    if zeile:
        summe += float(zeile)
    print(zeile)
```

```
# Summe ausgeben
print("Summe:", summe)
```

**Listing 8.7** Datei lesen\_alles.py

Die Ausgabe des Programms:

```
6
2.5
4
```

**Summe: 12.5**

Zur Erläuterung:

- read() ▶ Alle Zeilen werden mithilfe der Funktion `read()` in einen String gelesen.
- split() ▶ Der String wird anhand des Zeilenende-Zeichens (Unicode-Zahl 10) mithilfe der Funktion `split()` in einzelne Zeilen zerlegt.
- ▶ Über eine `while`-Schleife wird das Lesen der Zeilen gesteuert.
- ▶ Die Zeilen werden summiert und ausgegeben, anschließend wird die Summe ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 8.3.3 CSV-Datei schreiben

##### Comma-separated Values

CSV-Dateien (CSV = *Comma-separated Values*) sind Textdateien, in denen ein Datensatz pro Zeile steht. Die Daten des Datensatzes sind durch festgelegte Zeichen (wie Semikolon) voneinander getrennt. Solche Dateien haben meist die Dateiendung `.csv`. Sie können auch von anderen Programmen (z. B. Microsoft Excel oder LibreOffice Calc) erkannt und richtig genutzt werden.

##### Spiel mit CSV

Neben den Beispielen in diesem Abschnitt finden Sie in [Abschnitt 8.10, »Spiel, objektorientierte Version mit Highscore-Datei«](#), eine weitere Version des bereits bekannten Spiels. Darin werden die Daten des Spielers (Name und Zeit) in einer CSV-Datei dauerhaft gespeichert, sodass Sie eine Highscore-Liste führen können.

Ein Beispiel, in dem eine einfache Liste und eine zweidimensionale Liste als CSV-Datensätze in eine Datei geschrieben werden:

```
import sys

# Zugriffsversuch
try:
    d = open("daten.csv", "w")
```

```

except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Schreiben einer Liste als CSV-Datensatz
li = [42, "Maier", 3524.52]
d.write(str(li[0]) + ";" + li[1] + ";"
       + str(li[2]).replace(".",",") + "\n")

# Schreiben einer zweidim. Liste als Datensatztabelle
dli = [[55, "Warner", 3185.00], [57, "Schulz", 2855.20]]
for element in dli:
    d.write(str(element[0]) + ";"
           + element[1] + ";"
           + str(element[2]).replace(".",",") + "\n")

# Schliessen der Datei
d.close()

```

#### **Listing 8.8 Datei schreiben\_csv.py**

Die Ausgabedatei *daten.csv* hat anschließend den folgenden Inhalt:

```

42;Maier;3524,52
55;Warner;3185,0
57;Schulz;2855,2

```

#### **Listing 8.9 Datei daten.csv**

Zur Erläuterung:

- ▶ Nach dem erfolgreichen Öffnen werden die einzelnen Elemente der einfachen Liste mithilfe der Funktion `write()` in die Datei geschrieben. `write()`
- ▶ Zwischen den einzelnen Elementen wird jeweils ein Semikolon eingefügt. Semikolon
- ▶ Handelt es sich bei einem Element um eine Zahl, so wird diese mithilfe der Funktion `str()` in eine Zeichenkette umgewandelt.
- ▶ Bei Zahlen mit Nachkommastellen wird anschließend der Dezimalpunkt mithilfe der Funktion `replace()` in ein Komma verwandelt. Damit können diese Zahlen problemlos von einer deutschen Version von Microsoft Excel eingelesen werden. Dezimalpunkt
- ▶ Die Struktur der Daten muss also bekannt sein. Darüber hinaus müssen die Daten der verschiedenen Datentypen unterschiedlich verarbeitet werden, um einen korrekten Export zu gewährleisten.

Falls Excel unter Windows installiert ist, genügt ein Doppelklick auf die Ausgabedatei *daten.csv*, um den Inhalt als Excel-Tabelle darzustellen, siehe [Abbildung 8.1](#).

**Microsoft Excel**

The screenshot shows a Microsoft Excel spreadsheet window. The ribbon at the top has 'Datei' selected. The main area contains a table with four rows and three columns. The first column is labeled 'A', the second 'B', and the third 'C'. Row 1 contains '42 Maier' in column B and '3524,52' in column C. Row 2 contains '55 Warner' in column B and '3185' in column C. Row 3 contains '57 Schulz' in column B and '2855,2' in column C. Row 4 is empty.

A	B	C
1	42 Maier	3524,52
2	55 Warner	3185
3	57 Schulz	2855,2
4		

Abbildung 8.1 CSV-Datei in Excel

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Unterschiede unter Ubuntu Linux und OS X

Unter Ubuntu Linux können Sie die Datei mit *LibreOffice Calc* öffnen. Dies geschieht auch automatisch nach einem Doppelklick auf den Dateinamen in der Verzeichnisanzeige. Es wird das Dialogfeld `TEXTIMPORT` dargestellt. Die Standardeinstellungen können Sie mit einer Ausnahme übernehmen: Das Komma als Trennoption muss entfernt werden.

Ansonsten würden die Zahlen mit Nachkommastellen auf zwei Spalten aufgeteilt. Anschließend werden die Daten korrekt dargestellt, wie in Excel unter Windows.

Die Freeware *LibreOffice* gibt es auch in einer Version für OS X. Mit *LibreOffice Calc* lässt sich auch hier die CSV-Datei öffnen und bearbeiten.

#### 8.3.4 CSV-Datei lesen

Um den Inhalt einer CSV-Datei zu importieren, müssen auch Struktur und Datentypen der Daten bekannt sein. Die CSV-Datei aus dem vorherigen Beispiel wird in Excel ergänzt und abgespeichert, siehe Abbildung 8.2.

Die Datei wird mit dem folgenden Programm gelesen. Die Datensätze werden in einer zweidimensionalen Liste abgelegt:

```
import sys

# Zugriffsversuch
try:
    d = open("daten.csv")
```

```

except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Lesen des gesamten Texts
gesamtertext = d.read()

# Schliessen der Datei
d.close()

# Umwandeln in eine Liste von Zeilen
zeilenliste = gesamtertext.split(chr(10))

# Jede Zeile umwandeln in Liste von int, string, float
li = []
for zeile in zeilenliste:
    if zeile:
        zwliste = zeile.split(";")
        li.append([int(zwliste[0]),
                   zwliste[1],
                   float(zwliste[2].replace(",",".")))])
# Ausgabe
for p in li:
    print("{0:d} {1} {2:.2f}".format(p[0], p[1], p[2]))

```

**Listing 8.10** Datei lesen\_csv.py

	A	B	C
1	42	Maier	3524,52
2	55	Warner	3185
3	57	Schulz	2855,2
4	98	Esser	3120,66
5			

**Abbildung 8.2** CSV-Datei in Excel, ergänzt

Das Programm erzeugt die Ausgabe:

42 Maier 3524.52  
 55 Warner 3185.00  
 57 Schulz 2855.20  
 98 Esser 3120.66

Zur Erläuterung:

- read()**
  - ▶ Zunächst wird der gesamte Text der Datei mithilfe der Funktion `read()` in einen String eingelesen.
- split()**
  - ▶ Dieser String wird mithilfe der Funktion `split()` und dem Zeilenende-Zeichen (Unicode-Zahl 10) in eine Liste von Zeilen zerlegt. Diese Liste enthält das Zeilenende-Zeichen nicht mehr.
  - ▶ Jedes Element der Liste, also jede Zeile, wird mithilfe der Funktion `split()` und dem Semikolon-Zeichen in eine Zwischenliste (`zwliste`) mit den einzelnen Daten zerlegt. Diese Zerlegung wird nur durchgeführt, wenn die Zeile nicht leer ist.
  - ▶ Handelt es sich bei einem Element der Zwischenliste um eine Zahl, so wird diese mithilfe der Funktion `int()` oder mit der Funktion `float()` umgewandelt.
- Dezimalpunkt**
  - ▶ Bei Zahlen mit Nachkommastellen wurde vorher das Komma mithilfe der Funktion `replace()` in einen Dezimalpunkt verwandelt. Damit kann diese Zahl, die aus einer deutschen Version von Excel kommt, problemlos weiterverarbeitet werden.
  - ▶ Zuletzt wird die zweidimensionale Liste zeilenweise formatiert ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 8.4 Dateien mit festgelegter Struktur

- Lesen** Wenn eine Datei formatiert beschrieben wird, ist ihre Struktur festgelegt. Es ist genau bekannt, an welcher Stelle welche Information steht. Somit können die gewünschten Informationen direkt aus der Datei gelesen werden, ohne die gesamte Datei Zeile für Zeile einlesen zu müssen.
- Schreiben** Außerdem ist es möglich, bestimmte Informationen *punktuell* zu verändern, ohne den restlichen Inhalt der Datei zu beeinflussen. Beide Möglichkeiten werden in den folgenden Abschnitten vorgeführt.

### 8.4.1 Formatiertes Schreiben

- Formatieren** Das formatierte Schreiben in eine Datei geschieht auf die gleiche Art und Weise wie die formatierte Ausgabe. Es werden Datensätze gleicher Länge und gleichen Aufbaus gebildet.
- Editor** Optional fügen Sie nach jedem Datensatz ein Zeilenende ein, um die Datei in einem Editor leichter lesbar zu machen. Sie können die Datei auch mithilfe eines Editors verändern, solange Sie sich an die festgelegte Struktur halten.

Ein Beispiel:

```
import sys

# Zugriffsversuch
try:
    d = open("obst.txt", "w")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Tabelle mit verschiedenen Objekten
fm = "{0:04d}{1:>12}{2:>4}{3:8.2f} Euro{4:8.2f} Euro\n"
artname = {23:"Apfel", 8:"Banane", 42:"Pfirsich"}
anzahl = {23:1, 8:3, 42:5}
epreis = {23:2.95, 8:1.45, 42:3.05}

d.write("{0:>4}{1:>12}{2:>4}{3:>13}{4:>13}\n".format
        ("Nr", "Name", "Anz", "EP", "GP"))
for x in 23, 8, 42:
    d.write(fm.format(x, artname[x], anzahl[x],
                      epreis[x], anzahl[x] * epreis[x]))

# Schliessen der Datei
d.close()
```

**Listing 8.11** Datei schreiben\_formatiert.py

Anschließend sieht die Datei *obst.txt* wie folgt aus:

Nr	Name	Anz	EP	GP
0023	Apfel	1	2.95 Euro	2.95 Euro
0008	Banane	3	1.45 Euro	4.35 Euro
0042	Pfirsich	5	3.05 Euro	15.25 Euro

**Listing 8.12** Datei obst.txt

Zur Erläuterung:

- ▶ Die Datei wird zum Schreiben geöffnet.
- ▶ Die einzelnen Datensätze werden mithilfe der Funktion `format()` formatiert, wie in [Abschnitt 5.2.2](#), »Formatierte Ausgabe mit `format()`«, beschrieben. Zusätzlich wird jeweils ein `\n` als Zeilenende angefügt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 8.4.2 Lesen an beliebiger Stelle

**seek()** Die Funktion `seek()` ermöglicht die Veränderung der aktuellen Lese- oder Schreibposition innerhalb der Datei. Sie kann mit einem oder zwei Parametern aufgerufen werden. Wird sie mit einem Parameter aufgerufen, so ist dies der Abstand in Byte, gemessen vom Dateianfang. Der Dateianfang entspricht dem Wert 0, der Aufruf lautet dann: `d.seek(0)`.

Diese Position wird direkt erreicht, ohne Lesen der Informationen vor der Position. Ausgehend von der erreichten Position kann gelesen oder geschrieben werden.

**read(<Byteanzahl>)** Die Funktion `seek()` wird im vorliegenden Beispiel zusammen mit einer Variante der Funktion `read()` eingeführt. Diese Funktion gestattet neben dem Einlesen einer ganzen Datei (wie am Anfang dieses Kapitels gezeigt) auch das Einlesen einer bestimmten Anzahl von Byte, falls die Funktion mit der entsprechenden Zahl als Parameter angegeben wird. Ein Beispiel:

```
import sys, os

# Zugriffsversuch
try:
    d = open("obst.txt")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)
# Gezieltes Lesen
for i in range(1,4):
    # Nr Lesen
    d.seek(48*i)
    nr = int(d.read(4))

    # EP Lesen
    d.seek(20 + 48*i)
    ep = float(d.read(8))

    # Ausgabe
    print("Artikel Nr:", nr, ", EP:", ep)

# Schliessen der Datei
d.close()
```

**Listing 8.13** Datei lesen\_beliebig.py

Die Ausgabe lautet:

**Artikel Nr: 23 , EP: 2.95  
Artikel Nr: 8 , EP: 1.45  
Artikel Nr: 42 , EP: 3.05**

Die Datei ist wie folgt aufgebaut:

- ▶ Jeder Datensatz (also jede Zeile) hat eine Gesamtlänge von 48 Byte, inklusive der zwei Zeichen, die durch das `\n` erzeugt werden. Datensatz
  - ▶ Die erste Zeile beginnt an Position 0 und enthält die Überschrift, die zweite Zeile an Position 48 und enthält den ersten Datensatz, die dritte Zeile an Position 96 und enthält den zweiten Datensatz usw.
- Zur Erläuterung des Programms:
- ▶ Die Datei wird zum Lesen geöffnet.
  - ▶ In einer `for`-Schleife werden mithilfe der Funktion `seek()` nacheinander Positionen im ersten, zweiten und dritten Datensatz erreicht. Dies sind
    - die Positionen 48, 96 und 144 für den Anfang jedes Datensatzes zum Lesen der Artikelnummer und
    - die Positionen  $(48 + 20) = 68$ ,  $(96 + 20) = 116$  und  $(144 + 20) = 164$  zum Lesen des Einzelpreises.seek (<Position>)
  - ▶ An diesen Positionen werden die nächsten 4 Byte (die Artikelnummer) bzw. die nächsten 8 Byte (der Einzelpreis) gelesen. Es handelt sich dabei jeweils um Zeichenketten, die mithilfe von `int()` und `float()` in Zahlen umgewandelt werden.
  - ▶ Anschließend werden die Zahlen ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Unterschiede unter Ubuntu Linux und OS X

Das Zeichen `\n` für das Zeilenende erzeugt beim Schreiben der Datensätze nur ein Zeichen, nicht zwei Zeichen wie unter Windows. Daher ist ein Datensatz 47 statt 48 Zeichen lang. Die drei Datensätze beginnen an den Positionen 47, 94 und 141. Entsprechend müssen Sie im Programm bei der Methode `seek()` jeweils mit `47 * i` statt mit `48 * i` arbeiten.

### 8.4.3 Schreiben an beliebiger Stelle

Sie können eine Datei im Modus `r+` öffnen. Anschließend können Sie sie lesen und beschreiben. Im folgenden Programm werden der aktuelle Einzelpreis und der zugehörige Gesamtpreis eines bestimmten Artikels gelesen, verändert und wieder in die Datei geschrieben.

**Modus r+**

```

import sys, os

# Zugriffsversuch
try:
    d = open("obst.txt", "r+")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Lesen des Einzelpreises
d.seek(68)
ep_str = d.read(8)
ep = float(ep_str)

# Schreiben des Einzelpreises
d.seek(68)
ep = ep + 0.2
d.write("{0:8.2f}".format(ep))

# Lesen des Gesamtpreises
d.seek(81)
gp_str = d.read(8)
gp = float(gp_str)
# Schreiben des Gesamtpreises
d.seek(81)
gp = gp + 0.2
d.write("{0:8.2f}".format(gp))

# Schliessen der Datei
d.close()

```

**Listing 8.14** Datei schreiben\_beliebig.py

Anschließend sieht die Datei *obst.txt* wie folgt aus:

Nr	Name	Anz	EP	GP
0023	Apfel	1	3.15 Euro	3.15 Euro
0008	Banane	3	1.45 Euro	4.35 Euro
0042	Pfirsich	5	3.05 Euro	15.25 Euro

**Listing 8.15** Datei obst.txt

Der Einzelpreis des Artikels 23 im ersten Datensatz wurde von 2,95 € auf 3,15 € erhöht.

Zur Erläuterung:

- ▶ Die Datei wird im Modus r+ geöffnet.

- Lesen**
- ▶ Der Einzelpreis steht als Zeichenkette in den 8 Byte ab der Position 68. Die Zeichenkette wird gelesen und in eine Zahl umgewandelt.

- Diese Zahl wird verändert, als Zeichenkette mit einer Breite von 8 Byte formatiert und ab der Position 68 in die Datei geschrieben.
- Das Gleiche geschieht mit dem Gesamtpreis ab Position 81.

Schreiben

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Unterschiede unter Ubuntu Linux und OS X

Der Datensatz für den ersten Artikel beginnt an Position 47 statt 48. Daher steht der Einzelpreis ab Position 67 statt 68 und der Gesamtpreis ab Position 80 statt 81.

## 8.5 Serialisierung

Das Modul `pickle` dient zur Serialisierung und Deserialisierung von Objekten. Dies können Objekte der eingebauten Objekttypen oder Objekte eigener Klassen sein.

Mithilfe der (De-)Serialisierung können Objekte aus Programmen als Bytefolge auf einfache Art und Weise in Dateien gespeichert oder aus Dateien wieder in Programme geladen werden.

(De-)Serialisierung

Beim Speichern wird der Typ des Objekts mitgespeichert, sodass beim Laden der Typ bekannt ist.

Neben den Beispielen in diesem Abschnitt finden Sie in [Abschnitt 8.10, »Spiel, Version mit Highscore-Datei«](#), eine weitere Version des bereits bekannten Spiels. Darin werden die Daten des Spielers (Name und Zeit) mithilfe der Serialisierung dauerhaft gespeichert, sodass eine Highscore-Liste geführt werden kann.

Spiel mit  
Serialisierung

### 8.5.1 Objekte in Datei schreiben

Zum Schreiben (Serialisieren) von Objekten dient die Methode `dump()`. Im folgenden Beispiel werden das Objekt eines eingebauten Objekttyps und das Objekt einer eigenen Klasse nacheinander in einer Datei gespeichert.

`pickle.dump()`

```
import pickle

# Definition der Klasse Fahrzeug
class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
```

```

def __str__(self):
    return self.bezeichnung + " " \
           + str(self.geschwindigkeit) + " km/h"

# Zugriffsversuch
try:
    d = open("objekt.bin", "wb")

except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

# Eingebautes Objekt
x = ([4, "abc", 8], "xyz")
print(x)
pickle.dump(x, d)
# Objekt der eigenen Klasse
opel = Fahrzeug("Opel Admiral", 40)
print(opel)
pickle.dump(opel, d)

# Variable Anzahl an Objekten
pickle.dump(3, d)
pickle.dump("Berlin", d)
pickle.dump("Hamburg", d)
pickle.dump("Dortmund", d)

# Datei schliessen
d.close()

```

**Listing 8.16** Datei objekt\_schreiben.py

Die Ausgabe lautet:

```
([4, 'abc', 8], 'xyz')
Opel Admiral 40 km/h
```

Zur Erläuterung:

- ▶ Das Programm beginnt mit der Definition der eigenen Klasse `Fahrzeug`, die Sie bereits aus Kapitel 6 zur objektorientierten Programmierung kennen.
- ▶ Die Datei wird im Modus `wb` geöffnet, also zum Schreiben im Binärmodus. Als zusätzlicher Hinweis auf den Binärmodus wird die Dateiendung `.bin` gewählt.
- ▶ Es wird eine zweidimensionale Liste erzeugt. Diese wird zur Kontrolle auf dem Bildschirm ausgegeben.
- ▶ Mithilfe der Funktion `dump()` aus dem Modul `pickle` wird die Liste binär in der geöffneten Datei gespeichert. Der erste Parameter der Funktion ist der Name des Objekts, der zweite Parameter ist das Dateiobjekt.

- ▶ Es wird ein Objekt der Klasse `Fahrzeug` erzeugt. Dieses Objekt wird zur Kontrolle auf dem Bildschirm ausgegeben. Anschließend wird es binär in der geöffneten Datei gespeichert.
- ▶ Zur Speicherung einer variablen Anzahl von Objekten (hier: drei Zeichenketten) muss zunächst die Anzahl der Objekte gespeichert werden. Anschließend können die Objekte selbst gespeichert werden. Damit ist für ein Programm zum Laden der Objekte die Anzahl bekannt (siehe nächster Abschnitt 8.5.2).

Variable Anzahl von Objekten

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Hinweis

Ein potentieller Inhalt der Datei wird überschrieben. Es könnten auch Objekte an vorhandene Objekte angehängt werden. Zu diesem Zweck hätten Sie die Datei im Öffnungsmodus `ab` öffnen müssen.

Modus »ab«

## 8.5.2 Objekte aus Datei lesen

Objekte, die mit der Funktion `dump()` in einer Datei gespeichert wurden, können Sie mit `load()` wieder in der gleichen Reihenfolge aus der Datei in ein Programm übernehmen. Handelt es sich um ein Objekt einer eigenen Klasse, muss die Definition dieser Klasse bekannt sein.

`pickle.load()`

Es folgt ein Beispiel, in dem die beiden Objekte, die mit dem Programm aus dem vorherigen Abschnitt serialisiert wurden, wieder deserialisiert und in ein Programm übernommen werden.

```
import pickle

# Definition der Klasse Fahrzeug
class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def __str__(self):
        return self.bezeichnung + " " \
            + str(self.geschwindigkeit) + " km/h"

# Zugriffsversuch
try:
    d = open("objekt.bin", "rb")
except:
    print("Dateizugriff nicht erfolgreich")
```

```

        sys.exit(0)
# Eingebautes Objekt
x = pickle.load(d)
print(x)

# Objekt der eigenen Klasse
opel = pickle.load(d)
print(opel)
# Variable Anzahl an Objekten
anzahl = pickle.load(d)
for i in range(anzahl):
    print(pickle.load(d))

# Datei schliessen
d.close()

```

**Listing 8.17** Datei objekt\_leSEN.py

Es wird die Ausgabe erzeugt:

```

([4, 'abc', 8], 'xyz')
Opel Admiral 40 km/h
Berlin
Hamburg
Dortmund

```

Zur Erläuterung:

- ▶ Sie erkennen, dass die Objekte wieder *originalgetreu* geladen werden.
- ▶ Das Programm beginnt mit der Definition der eigenen Klasse `Fahrzeug`.
- ▶ Die Datei wird im Modus `rb` geöffnet, also zum Lesen im Binärmodus.
- pickle.load()** ▶ Mithilfe der Funktion `load()` aus dem Modul `pickle` werden nacheinander die Liste und das Objekt der Klasse `Fahrzeug` geladen.
- Variable Anzahl** ▶ Anschließend wird die Anzahl der folgenden Objekte als Zahl geladen (hier: 3). Damit ist für die folgende Schleife die Anzahl der zu ladenden Objekte bekannt.
- ▶ Alle Objekte werden zur Kontrolle auf dem Bildschirm ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 8.6 Bearbeitung mehrerer Dateien

Bisher wurde immer eine einzelne Datei bearbeitet, deren Name bekannt war. In der Praxis stellt sich aber häufig die Aufgabe, eine ganze Reihe von Dateien

aus einem Verzeichnis zu bearbeiten, deren genauer Name und deren Anzahl unbekannt sind.

Zu diesem Zweck können Sie sich der Funktion `glob()` aus dem Modul `glob` bedienen. Diese Funktion erzeugt eine Liste von Strings. Jeder dieser Strings enthält einen Dateinamen. `glob.glob()`

Im folgenden Beispiel werden zunächst mit der Funktion `glob()` bestimmte Dateien aus dem aktuellen Verzeichnis in einer Liste gespeichert. Anschließend werden diese Dateien der Reihe nach geöffnet und durchsucht. Wird ein bestimmter Suchtext gefunden, so wird der Name der betreffenden Datei ausgegeben. Dateiliste

```
import glob

# Liste der Dateien
dateiliste = glob.glob("schr*.py")

# Jedes Element der Liste durchsuchen
for datei in dateiliste:
    # Zugriffsversuch
    try:
        d = open(datei)
    except:
        print("Dateizugriff nicht erfolgreich")
        continue

    # Text einlesen
    gesamtertext = d.read()

    # Zugriff beenden
    d.close()

    # Suchtext suchen
    if gesamtertext.find("obst") != -1:
        print(datei)
```

**Listing 8.18** Datei `datei_liste.py`

Die Ausgabe lautet:

`schreiben_beliebig.py`  
`schreiben_formatiert.py`

Zur Erläuterung:

- ▶ Der Parameter der Funktion `glob()` ist eine Zeichenkette, die als Filter dient. Sie können unter anderem die Platzhalter `*` (für mehrere beliebige Zeichen) und `?` (für ein einzelnes beliebiges Zeichen) einsetzen. Platzhalter \* ?

- ▶ Es wird nur nach Dateien gesucht, deren Name mit `schr` beginnt und mit `.py` endet.
- ▶ Jedes Element der erzeugten Liste ist ein Dateiname. Jede dieser Dateien wird geöffnet. Falls das Öffnen nicht erfolgreich war, wird mit der nächsten Datei fortgefahrene.
- `read()`
- ▶ Der Inhalt der Datei wird mithilfe der Funktion `read()` vollständig in eine Zeichenkette eingelesen. Anschließend wird die Datei geschlossen.
- `find()`
- ▶ Diese Zeichenkette wird mithilfe der Funktion `find()` durchsucht. Wird der Suchtext (hier: »obst«) gefunden, so wird der Dateiname ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 8.7 Informationen über Dateien

`os.stat()` Die Funktion `stat()` aus dem Modul `os` liefert eine Reihe von Informationen über Dateien in Form eines Tupels.

Ein Beispiel:

```
import os, time

# Informationstupel
tu = os.stat("obst.txt")

# Elemente des Tupels
groesse = tu[6]
print("Byte:", groesse)

zugr = time.localtime(tu[7])
print("Letzter Zugriff:",
      time.strftime("%d.%m.%Y %H:%M:%S", zugr))

mod = time.localtime(tu[8])
print("Letzte Modifikation:",
      time.strftime("%d.%m.%Y %H:%M:%S", mod))
```

**Listing 8.19** Datei `datei_info.py`

Die Ausgabe lautet:

```
Byte: 192
Letzter Zugriff: 27.11.2013 07:56:04
Letzte Modifikation: 27.11.2013 08:03:42
```

Zur Erläuterung:

- ▶ Das Element 6 des Tupels gibt die Größe der Datei in Byte an. Dateigröße
- ▶ Die Elemente 7 und 8 liefern den Zeitpunkt des letzten Zugriffs auf die Datei und der letzten Änderung der Datei. Zugriffszeitpunkt
- ▶ Beide Zeitangaben werden mithilfe der Funktionen `localtime()` und `strftime()` formatiert ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### Unterschiede unter Ubuntu Linux und OS X

Die Datensätze haben nur eine Länge von 47 statt 48 Zeichen. Damit ergibt sich eine Dateigröße von 188 Byte.

## 8.8 Dateien und Verzeichnisse verwalten

Die Module `os` und `shutil` bieten weitere Funktionen zur Verwaltung von Dateien und Verzeichnissen. Im folgenden Beispiel werden Dateien kopiert und umbenannt (Ubuntu Linux und OS X) bzw. gelöscht (Windows).

**Module os, shutil**

```
import sys, shutil, os, glob

# Status 1
print(glob.glob("le*.txt"))

# Existenz feststellen
if not os.path.exists("lesen.txt"):
    print("Datei nicht vorhanden")
    sys.exit(0)

# Datei kopieren
shutil.copyfile("lesen.txt","lesen_kopie.txt")

# Status 2
print(glob.glob("le*.txt"))

# Datei umbenennen
try:
    os.rename("lesen_kopie.txt","lesen.txt")
except:
    print("Fehler beim Umbenennen")
```

```

# Status 3
print(glob.glob("le*.txt"))
# Datei entfernen
try:
    os.remove("lesen_kopie.txt")
except:
    print("Fehler beim Entfernen")

# Status 4
print(glob.glob("le*.txt"))

```

**Listing 8.20** Datei datei\_verwalten.py

Die Ausgabe dieses Programms lautet:

```

['lesen.txt']
['lesen.txt', 'lesen_kopie.txt']
Fehler beim Umbenennen
['lesen.txt', 'lesen_kopie.txt']
['lesen.txt']

```

Zur Erläuterung:

- ▶ Die erste Zeile der Ausgabe gibt den Status 1 wieder: die ursprüngliche Liste der Dateien, die mit le beginnen und mit .txt enden.
- exists()** ▶ Dann wird mithilfe der Funktion `exists()` aus dem Modul `os.path` geprüft, ob die Datei existiert, die kopiert werden soll. Ist das nicht der Fall, dann wird das Programm vorzeitig beendet.
- shutil.copyfile()** ▶ Anschließend wird mithilfe der Funktion `copyfile()` aus dem Modul `shutil` die Datei `lesen.txt` kopiert. Die Kopie heißt `lesen_kopie.txt`. Falls die Datei `lesen_kopie.txt` bereits existiert, so wird sie ohne Vorwarnung (!) überschrieben.
- ▶ Die zweite Zeile der Ausgabe gibt den Status 2 wieder: die geänderte Liste der Dateien nach dem Kopiervorgang.
- os.rename()** ▶ Anschließend wird mithilfe der Funktion `rename()` aus dem Modul `os` die Datei `lesen_kopie.txt` umbenannt in `lesen.txt`. Da die Datei `lesen.txt` bereits existiert, tritt (unter Windows) ein Laufzeitfehler auf, und die Umbenennung findet nicht statt.
- ▶ Die nächste Zeile der Ausgabe gibt den Status 3 wieder: die Liste der Dateien nach dem (hier fehlgeschlagenen) Umbenennen.
- os.remove()** ▶ Anschließend wird mithilfe der Funktion `remove()` aus dem Modul `os` die Datei `lesen_kopie.txt` gelöscht.
- ▶ Die nächste Zeile der Ausgabe gibt den Status 4 wieder: die Liste der Dateien nach dem Löschen.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Unterschiede unter Ubuntu Linux und OS X

Das Umbenennen ist möglich, obwohl die Zielfile bereits vorhanden ist; daher tritt hier kein Laufzeitfehler auf. Dagegen tritt später ein Laufzeitfehler auf, wenn versucht wird, eine Datei zu löschen, die nicht existiert.

## 8.9 Beispielprojekt Morsezeichen

In diesem Abschnitt sehen Sie ein Beispielprojekt, in dem auf Dateien, externe Module, Zeichenketten, Dictionarys, einige Funktionen und das Modul `winsound` zur Ausgabe von Tönen (siehe [Abschnitt 7.5, »Audio-Ausgabe«](#)) zugegriffen wird.

Ein Beispieltext soll in Morsecode umgesetzt werden. Morsecode besteht aus einzelnen Morsezeichen und Pausen. Ein Morsezeichen steht für ein einzelnes Zeichen eines Texts und setzt sich aus einer Folge von kurzen und langen Signalen zusammen. Damit ist es seit langer Zeit möglich, einen Text als eine Folge von Ton-, Funk- oder Lichtsignalen zu übermitteln.

Signale

In [Kapitel 13, »Raspberry Pi«](#), werden Sie sehen, wie ein Text mithilfe des Lichtsignals einer Leuchtdiode in einer elektronischen Schaltung gesendet werden kann.

### 8.9.1 Morsezeichen aus Datei lesen

Zunächst sollen einzelne Zeichen und das zugehörige Morsezeichen aus der Datei `morsen.txt` in ein Dictionary gelesen werden. Ein kurzes Signal wird in der Datei durch einen Punkt dargestellt, ein langes Signal durch einen Strich. Die einzelnen Zeichen und das zugehörige Morsezeichen sind jeweils durch ein Leerzeichen getrennt:

- A . -
- B - ...
- C - -. .
- D - ..
- E .

#### **Listing 8.21 Datei morsen.txt, Ausschnitt**

Zum Lesen dient eine Funktion im Modul `morsen`, die anderen Python-Programmen zur Verfügung gestellt werden kann.

Es folgt das Modul `morsen`:

```
import sys

# Erstellt Dictionary mit Morsezeichen aus Datei
def leseCode():
    # Lesen der Datei mit Zeichen und Morsezeichen
    try:
        d = open("morsen.txt")
    except:
        print("Dateifehler")
        sys.exit(0)
    allezeilen = d.readlines()
    d.close

    # Erste Zeichenkette in der Zeile ist das Zeichen
    # Es dient als Schluessel fuer das Dictionary
    code = {}
    for zeile in allezeilen:
        worte = zeile.split()
        code[worte[0]] = worte[1]

    # Morsezeichen gleich, ob kleine oder groÙe Buchstaben
    for i in range(97,123):
        code[chr(i)] = code[chr(i-32)]

    # Liste zurückliefern
    return code
```

**Listing 8.22** Datei `morsen.py`

Zur Erläuterung:

- ▶ In der Funktion `leseCode()` werden alle Zeilen der Datei `morsen.txt` gelesen. Es wird ein leeres Dictionary erzeugt.
- ▶ Jede Zeile wird anhand des Leerzeichens in zwei Zeichenketten zerlegt. Die erste Zeichenkette beinhaltet ein Unicode-Zeichen, die zweite Zeichenkette das zugehörige Morsezeichen. Das Unicode-Zeichen wird als Schlüssel für den Eintrag des Morsezeichens im Dictionary genutzt.
- ▶ Beim Morsen wird nicht zwischen Groß- und Kleinschreibung unterschieden. Daher wird den Dictionary-Elementen für die Kleinbuchstaben das gleiche Morsezeichen zugewiesen wie den entsprechenden Elementen für die Großbuchstaben.
- ▶ Das Dictionary wird als Ergebnis der Funktion `leseCode()` zurückgeliefert.

### 8.9.2 Ausgabe auf dem Bildschirm

Zunächst sollen die Morsezeichen eines Beispieltexsts auf dem Bildschirm ausgegeben werden. Die einzelnen Morsezeichen werden jeweils durch ein Leerzeichen getrennt. Es folgt das Programm:

```
import sys, morsen

# Beispieltext codieren
def schreibeCode(text, code):
    for zeichen in text:
        try:
            print(code[zeichen], end=" ")
        except KeyError:
            print(" ", end=" ")
    print()

# Lesefunktion aufrufen
code = morsen.leseCode()

# Schreibfunktion aufrufen
schreibeCode("Hallo Welt", code)
```

**Listing 8.23** Datei morsen\_bildschirm.py

Die Ausgabe des Beispieltexsts "Hallo Welt" in Morsezeichen:

.... - . -. -. - - - . - . - - -

Zur Erläuterung:

- ▶ In der Funktion `schreibeCode()` wird ein übergebener Beispieltexst in Morsezeichen codiert und ausgegeben. Falls es zu einem Zeichen keine Entsprechung im Dictionary gibt, dann tritt ein `KeyError` auf. In diesem Falle wird ein Leerzeichen ausgegeben.
- ▶ Im Hauptprogramm wird zunächst die Funktion `leseCode()` aus dem Modul `morsen` aufgerufen. Sie liefert ein Dictionary zurück. Anschließend wird die Funktion `schreibeCode()` mit einem Beispieltexst aufgerufen.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Das Weglassen des Zeilenendes wird (wie in [Abschnitt 5.2.1](#), »Funktion `>print()<`«) erreicht, indem die Ausgabe stückweise in einer Variablen zusammengesetzt und erst anschließend vollständig ausgegeben wird.

### 8.9.3 Ausgabe mit Tonsignalen

In diesem Abschnitt sollen die Morsezeichen eines Beispieltexts als Tonsignale ausgegeben werden. Bevor Sie das Programm starten, sollten Sie den Lautsprecher Ihres Windows-PCs einschalten.

Es folgt das Programm:

```
import sys, morsen, time, winsound

# Beispieltext codieren
def tonCode(text, code):
    # Zeitschema, Dauer eines Signals in msec.
    signalDauer = {".":200, "-":600}

    # Zeitschema, Dauer einer Pause in sec.
    signalPause = 0.2
    zeichenPause = 0.6
    wortPause = 1.4

    # Text in Worte zerlegen
    alleWorte = text.split()

    # Jedes Wort im Text
    for w in range(len(alleWorte)):
        # Übernahme eines Worts
        wort = alleWorte[w]
        # Jedes Zeichen im Wort
        for z in range(len(wort)):
            # Übernahme eines Zeichens
            zeichen = wort[z]
            # Kontrollausgabe des Zeichens
            print(zeichen, end="")
            # Versuch, ein Zeichen auszugeben
            try:
                # Übernahme des Morsezeichens für das Zeichen
                # Falls kein Eintrag im Dictionary: KeyError
                alleSignale = code[zeichen]
                # Jedes Signal des Morsezeichens
                for s in range(len(alleSignale)):
                    # Übernahme eines Symbols
                    signal = alleSignale[s]
                    # Ausgabe des Symbols, kurz oder lang
                    winsound.Beep(800, signalDauer[signal])
                    # Nach jedem Signal eine Signalpause,
                    # ausser nach dem letzten Signal
                    if s < len(alleSignale)-1:
                        time.sleep(signalPause)
            # Nach jedem Zeichen eine Zeichenpause,
            # ausser nach dem letzten Zeichen
            except KeyError:
                print("Kein Eintrag im Dictionary für Zeichen", zeichen)
```

```

        if z < len(wort)-1:
            time.sleep(zeichenPause)
        # Falls kein Eintrag im Dictionary: ignorieren
        except KeyError:
            pass
        # Nach jedem Wort eine Wortpause,
        # ausser nach dem letzten Wort
        if w < len(alleWorte)-1:
            print(" ", end="")
            time.sleep(wortPause)

# Lesefunktion aufrufen
code = morsen.leseCode()

# Schreibfunktion aufrufen
tonCode("Hallo Welt", code)

```

**Listing 8.24** Datei morsen\_ton.py

Zur Erläuterung:

- ▶ In der Funktion `tonCode()` wird ein übergebener Beispieltext in Morsezeichen codiert und als Folge von Tonsignalen ausgegeben. Zunächst wird ein weiteres Dictionary angelegt. Darin wird die Signaldauer für ein kurzes Signal mit 200 Millisekunden festgelegt. Ein langes Signal muss dreimal so lang sein. Daher bekommt es eine Länge von 600 Millisekunden. Signaldauer
- ▶ Nach jedem Signal folgt eine Signalpause in der Dauer eines kurzen Signals, nach jedem Zeichen in der Dauer eines langen Signals und nach jedem Wort in der Dauer von sieben kurzen Signalen. Signalpause
- ▶ Der gesamte Text wird anhand der Leerzeichen in einzelne Worte zerlegt. Jedes Wort wird in einzelne Zeichen zerlegt. Zu jedem Zeichen wird der zugehörige Eintrag im Dictionary der Morsezeichen gesucht. Jedes gefundene Morsezeichen wird in einzelne Signale zerlegt. Jedes Signal wird mithilfe der Funktion `Beep()` aus dem Modul `winsound` als Ton ausgegeben. Mehrfache Zerlegung
- ▶ Falls es zu einem Zeichen keine Entsprechung im Dictionary gibt, tritt ein `KeyError` auf. In diesem Falle wird das Zeichen ignoriert. KeyError
- ▶ Die Funktion `sleep()` aus dem Modul `time` sorgt für die Pausen zwischen den Signalen, Zeichen und Wörtern. Pause
- ▶ Im Hauptprogramm wird zunächst die Funktion `leseCode()` aus dem Modul `morsen` aufgerufen. Sie liefert ein Dictionary zurück. Anschließend wird die Funktion `tonCode()` mit einem Beispieltext aufgerufen.

### Unterschiede in Python 2

Die Kontrollausgabe der einzelnen Zeichen auf dem Bildschirm entfällt.

### Unterschiede in Ubuntu Linux und OS X

Das Modul `winsound` steht nicht zur Verfügung. Unter Ubuntu Linux könnten Sie mit dem Paket `Beep` arbeiten, das vom Betriebssystem aus aufgerufen wird.

## 8.10 Spiel, Version mit Highscore-Datei

- Serialisierung** Es folgt eine weitere Version des bereits bekannten Spiels. Darin werden die Daten des Spielers (Name und Zeit) mithilfe der Serialisierung (siehe Abschnitt 8.5, »Serialisierung«) dauerhaft gespeichert, sodass eine Highscore-Liste geführt werden kann.
- Zeit messen** Es wird zunächst nach dem Namen des Spielers gefragt. Anschließend bekommt der Spieler fünf Additionsaufgaben mit Zahlen aus dem Bereich von 10 bis 30 gestellt. Pro Aufgabe hat er nur einen Versuch. Ungültige Eingaben oder falsche Ergebnisse werden einfach mit dem Text `Falsch` kommentiert und bewertet. Zuletzt wird die Gesamtzeit angegeben, die der Spieler für die Lösungsversuche benötigt hat.
- Highscore in Datei** Wenn der Spieler alle fünf Aufgaben richtig löst, werden sein Name und die Zeit in eine Highscore-Liste aufgenommen, die in einer Datei abgespeichert wird.

### 8.10.1 Eingabebeispiel

Die folgende Ausgabe zeigt einen typischen Durchlauf durch das Programm:

```
Bitte eingeben (0: Ende, 1: Highscores, 2: Spielen): 1
P. Name      Zeit
1. Bernd    8.59 sec
2. Marc     8.92 sec
Bitte eingeben (0: Ende, 1: Highscores, 2: Spielen): 2
Bitte geben Sie Ihren Namen ein (max. 10 Zeichen): Paul
Aufgabe 1 von 5: 26 + 21 : 47
*** Richtig ***
Aufgabe 2 von 5: 26 + 24 : 50
*** Richtig ***
Aufgabe 3 von 5: 27 + 22 : 49
*** Richtig ***
Aufgabe 4 von 5: 10 + 29 : 39
*** Richtig ***
Aufgabe 5 von 5: 19 + 28 : 47
*** Richtig ***
Ergebnis: 5 von 5 in 10.02 Sekunden, Highscore
P. Name      Zeit
1. Bernd    8.59 sec
```

2. Marc 8.92 sec

3. Paul 10.02 sec

Bitte eingeben (0: Ende, 1: Highscores, 2: Spielen): 0

Zur Erläuterung:

- ▶ Der Benutzer (Paul) schaut sich zuerst die Highscores an. Dort gibt es bereits zwei Einträge (Bernd und Marc).
- ▶ Anschließend spielt er eine Runde, löst alle fünf Aufgaben richtig und erscheint nun auch selbst im Highscore.
- ▶ Er verlässt das Programm wieder.

### 8.10.2 Aufbau des Programms

Das Programm enthält ein Hauptprogramm und vier Funktionen. Im Hauptprogramm wird innerhalb einer Endlosschleife ein Hauptmenü aufgerufen. Der Spieler hat in diesem Hauptmenü die folgenden Möglichkeiten:

Endlosschleife

- ▶ Anzeige der Highscores
- ▶ spielen
- ▶ Programm beenden

Die vier Funktionen des Programms sind:

- ▶ Funktion `hs_lesen()`: Lesen der Highscores aus der Datei in eine Liste
- ▶ Funktion `hs_anzeigen()`: Anzeigen der Highscore-Liste auf dem Bildschirm
- ▶ Funktion `hs_schreiben()`: Schreiben der Highscore-Liste in die Datei
- ▶ Funktion `spiel()`: Stellen der Aufgaben, Lösen, ermittelte Zeit gegebenenfalls in Highscore-Liste einfügen

### 8.10.3 Code des Programms

Das folgende Listing zeigt zunächst den Beginn des Programms mit der Funktion zum Lesen der Highscore-Daten aus der Datei:

Daten lesen

```
# Module
import random, time, glob, pickle
# Funktion Highscore lesen
def hs_lesen():
    # Liste wird hier erzeugt
    global hs_liste

    # Kein Highscore vorhanden, leere Liste
    if not glob.glob("highscore.bin"):
        hs_liste = []
        return
```

```
# Highscore vorhanden, laden
d = open("highscore.bin", "rb")
hs_liste = pickle.load(d)
d.close()
```

**Listing 8.25** Datei spiel\_datei.py, Teil 1 von 5

Zur Erläuterung:

- ▶ Es werden die Module `random` (für den Zufallsgenerator), `time` (für die Zeitmessung) und `glob` (für die Prüfung der Datei) benötigt.
- Globale Liste** ▶ In der Liste `hs_liste` stehen nach dem Einlesen aus der Datei die Namen und Ergebniszeiten der Spieler. Da die Liste in dieser Funktion erstmalig benutzt wird, aber im gesamten Programm benötigt wird, muss sie hier mit `global` im globalen Namensraum bekannt gemacht werden.
- ▶ Falls die Datei mit den Highscores nicht vorhanden ist, wird eine leere Liste erzeugt.
- ▶ Ist die Datei vorhanden, so wird der gesamte Inhalt mithilfe der Funktion `load()` aus dem Modul `pickle` in die Liste `hs_liste` gelesen.

Es folgt die Funktion zum Anzeigen des Highscores:

```
# Funktion Highscore anzeigen
def hs_anzeigen():
    # Highscore nicht vorhanden
    if not hs_liste:
        print("Keine Highscores vorhanden")
        return

    # Ausgabe Highscore
    print(" P. Name           Zeit")
    for i in range(len(hs_liste)):
        print("{0:2d}. {1:10} {2:5.2f} sec".format(
            (i+1, hs_liste[i][0], hs_liste[i][1])))
    if i >= 9:
        break
```

**Listing 8.26** Datei spiel\_datei.py, Teil 2 von 5

Zur Erläuterung:

- ▶ Falls die Highscore-Liste leer ist, wird eine entsprechende Meldung angezeigt.
- ▶ Ist die Liste nicht leer, so werden nach einer Überschrift die folgenden Informationen formatiert ausgegeben: *Platzierung, Name, Ergebniszeit*.
- ▶ Es werden maximal 10 Highscores angezeigt.

Die Funktion zum Schreiben der Highscore-Liste in die Datei sieht wie folgt aus:

```
# Funktion Highscore schreiben
def hs_schreiben():
    # Zugriff
    d = open("highscore.bin", "wb")
    pickle.dump(hs_liste, d)
    d.close()
```

**Listing 8.27** Datei spiel\_datei.py, Teil 3 von 5

Zur Erläuterung:

- Die gesamte Liste wird mithilfe der Funktion `dump()` aus dem Modul `pickle` in die Datei geschrieben.

Es folgt die Funktion zum Spielen:

```
# Funktion Spiel
def spiel():
    # Eingabe des Namens
    name = input("Bitte geben Sie Ihren "
                "Namen ein (max. 10 Zeichen): ")
    name = name[0:10]

    # Initialisierung Counter und Zeit
    richtig = 0
    startzeit = time.time()

    # 5 Aufgaben
    for aufgabe in range(5):
        # Aufgabe mit Ergebnis
        a = random.randint(10,30)
        b = random.randint(10,30)
        c = a + b
        # Eingabe
        try:
            zahl = int(input("Aufgabe "
                            + str(aufgabe+1) + " von 5: "
                            + str(a) + " + " + str(b) + " : "))
            if zahl == c:
                print("*** Richtig ***")
                richtig += 1
            else:
                raise
        except:
            print("* Falsch *")
    # Auswertung
    endzeit = time.time()
    differenz = endzeit-startzeit
    print("Ergebnis: {0:d} von 5 in {1:.2f} Sekunden".
          format(richtig, differenz), end = "")
    if richtig == 5:
        print(", Highscore")
```

```

else:
    print(", leider kein Highscore")
    return
# Mitten in Liste schreiben
gefunden = False
for i in range(len(hs_liste)):
    # Einsetzen in Liste
    if differenz < hs_liste[i][1]:
        hs_liste.insert(i, [name, differenz])
        gefunden = True
        break
# Ans Ende der Liste schreiben
if not gefunden:
    hs_liste.append([name, differenz])
# Highscoreliste anzeigen
hs_anzeigen()

```

Listing 8.28 Datei spiel\_datei.py, Teil 4 von 5

Zur Erläuterung:

- ▶ Nach der Eingabe des Namens wird dieser Name zur einheitlichen Ausgabe auf maximal 10 Zeichen verkürzt.
- ▶ Es folgen einige Programmteile, die bereits aus früheren Versionen des Spiels bekannt sind: Stellen der fünf Aufgaben, Eingabe der Lösungen, Beurteilen der Lösungen, Messen der Zeit.
- ▶ Es ist zu beachten, dass die Zahlenvariablen `aufgabe`, `a` und `b` für die Funktion `input()` in Zeichenketten umgewandelt werden müssen.
- ▶ Wenn nicht alle fünf Aufgaben richtig gelöst wurden, kann kein Eintrag in die Highscore-Liste erfolgen.
- ▶ Die Highscore-Liste wird Element für Element durchsucht. Wird ein Element gefunden, in dem eine größere als die neue Ergebniszeit eingetragen ist, so wird die neue Ergebniszeit mithilfe der Funktion `insert()` an dieser Stelle in die Highscore-Liste eingefügt.
- ▶ Falls kein Eintrag durch Einfügen erfolgte, wird die neue Ergebniszeit mithilfe der Funktion `append()` ans Ende der Liste angefügt.
- ▶ Auf diese Weise ist dafür gesorgt, dass die Highscore-Liste immer aufsteigend nach Ergebniszeit sortiert ist.
- ▶ Die Highscore-Liste wird unmittelbar nach jeder Änderung angezeigt.

Das folgende Listing zeigt schließlich das Hauptprogramm mit dem Hauptmenü:

```

# Hauptprogramm

# Initialisierung Zufallsgenerator
random.seed()

```

```

# Highscore aus Datei in Liste lesen
hs_leSEN()

# Endlosschleife
while True:
    # Hauptmenue, Auswahl
    try:
        menu = int(input("Bitte eingeben "
                         "(0: Ende, 1: Highscores, 2: Spielen): "))
    except:
        print("Falsche Eingabe")
        continue

    # Aufruf einer Funktion oder Ende
    if menu == 0:
        break
    elif menu == 1:
        hs_anzeigen()
    elif menu == 2:
        spiel()
    else:
        print("Falsche Eingabe")

# Highscore aus Liste in Datei schreiben
hs_schreiben()

```

**Listing 8.29** Datei spiel\_datei.py, Teil 5 von 5

Zur Erläuterung:

- ▶ Nach der Initialisierung des Zufallsgenerators wird die Highscore-Datei eingelesen.
- ▶ Es beginnt die Endlosschleife mit dem Hauptmenü. Der Benutzer kann 0, 1 oder 2 eingeben. Alle anderen Eingaben werden als falsch gewertet.
- ▶ Nach der Eingabe von 1 wird die Highscore-Liste angezeigt, anschließend erscheint wieder das Hauptmenü.
- ▶ Nach der Eingabe von 2 beginnt das Spiel, anschließend erscheint wieder das Hauptmenü.
- ▶ Nach der Eingabe von 0 wird die Endlosschleife verlassen. Die Highscore-Liste wird in die Datei geschrieben, und das Programm endet.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen `\` für den Umbruch von langen Programmzeilen eingesetzt.

## 8.11 Spiel, objektorientierte Version mit Highscore-Datei

**CSV-Datei** Es folgt eine weitere objektorientierte Version des Spiels. Diesmal werden die Daten des Spielers (Name und Zeit) in einer CSV-Datei gespeichert. Diese CSV-Datei können Sie sich mithilfe von Microsoft Excel unter Windows oder mit *LibreOffice Calc* unter Ubuntu Linux und OS X ansehen.

Neben den beiden Klassen `Spiel` und `Aufgabe` benötigen Sie die Klasse `Highscore`. Zunächst die Import-Anweisung und das Hauptprogramm:

```
import random, time, glob

# Hauptprogramm
while True:
    # Hauptmenue, Auswahl
    try:
        menu = int(input("Bitte eingeben "
                         "(0: Ende, 1: Highscores, 2: Spielen): "))
    except:
        print("Falsche Eingabe")
        continue

    # Anlegen eines Objekts oder Ende
    if menu == 0:
        break
    elif menu == 1:
        hs = Highscore()
        print(hs)
    elif menu == 2:
        s = Spiel()
        s.messen(True)
        s.spielen()
        s.messen(False)
        print(s)
    else:
        print("Falsche Eingabe")
```

**Listing 8.30** Datei `spiel_datei_oop.py`, Hauptprogramm

Zur Erläuterung:

- ▶ Die Module `random`, `time` und `glob` werden für Zufallsgenerator, Zeitmessung und eine Dateiprüfung benötigt.
- ▶ Falls der Benutzer die Highscore-Liste sehen möchte, so wird ein Objekt der Klasse `Highscore` erzeugt und ausgegeben.
- ▶ Falls der Benutzer spielen möchte, so wird ein Objekt der Klasse `Spiel` erzeugt. Nach der Messung der Zeit und dem eigentlichen Spielvorgang werden die Ergebnisse ausgegeben.

Es folgt die Klasse Spiel:

Klasse »Spiel«

```
class Spiel:
    def __init__(self):
        # Start des Spiels
        random.seed()
        self.richtig = 0
        self.anzahl = 5
        s = input("Bitte geben Sie Ihren "
                  "Namen ein (max. 10 Zeichen): ")
        self.spieler = s[0:10]

    def spielen(self):
        # Spielablauf
        for i in range(1,self.anzahl+1):
            a = Aufgabe(i, self.anzahl)
            print(a)
            self.richtig += a.beantworten()

    def messen(self, start):
        # Zeitmessung
        if start:
            self.startzeit = time.time()
        else:
            endzeit = time.time()
            self.zeit = endzeit - self.startzeit

    def __str__(self):
        # Ergebnis
        ausgabe = "Richtig: {0:d} von {1:d} in " \
                  "{2:.2f} Sekunden".format(self.richtig, \
                  self.anzahl, self.zeit)
        if self.richtig == self.anzahl:
            ausgabe += ", Highscore"
            hs = Highscore()
            hs.speichern(self.spieler, self.zeit)
            print(hs)
        else:
            ausgabe += ", leider kein Highscore"
        return ausgabe
```

**Listing 8.31** Datei spiel\_datei\_oop.py, Klasse »Spiel«

Zur Erläuterung:

- Im Konstruktor der Klasse wird der Zufallsgenerator initialisiert. Der Zähler für die richtig gelösten Aufgaben wird zunächst auf 0, die Anzahl der Aufgaben auf 5 gesetzt. Es wird der Name des Spielers ermittelt. Dabei werden drei Eigenschaften der Klasse Spiel gesetzt: richtig, anzahl und spieler.

Konstruktor,  
Eigenschaften

- Methoden**
- ▶ In der Methode `spielen()` werden insgesamt fünf Aufgaben gestellt und beantwortet. Die Methode `messen()` dient zur Zeitmessung. Es wird die Eigenschaft `zeit` der Klasse `Spiel` gesetzt.
  - ▶ Falls der Spieler alle Aufgaben richtig gelöst hat, wird in der Ausgabemethode ein Objekt der Klasse `Highscore` erzeugt. Der Highscore wird gespeichert. Anschließend wird die Liste ausgegeben.

Die Klasse `Aufgabe` hat sich gegenüber der Version in der Datei `spiel_oop.py` nicht verändert.

**Klasse »Highscore«** Die neue Klasse `Highscore` sieht wie folgt aus:

```
class Highscore:
    # Liste aus Datei lesen
    def __init__(self):
        self.liste = []
        if not glob.glob("highscore.csv"):
            return
        d = open("highscore.csv")
        zeile = d.readline()
        while(zeile):
            teil = zeile.split(";")
            name = teil[0]
            zeit = teil[1][0:len(teil[1])-1]
            zeit = zeit.replace(",",".")
            self.liste.append([name, float(zeit)])
            zeile = d.readline()
        d.close()
    # Liste ändern
    def aendern(self, name, zeit):
        # Mitten in Liste schreiben
        gefunden = False
        for i in range(len(self.liste)):
            # Einsetzen in Liste
            if zeit < self.liste[i][1]:
                self.liste.insert(i, [name, zeit])
                gefunden = True
                break
        # Ans Ende der Liste schreiben
        if not gefunden:
            self.liste.append([name, zeit])
    # Liste ändern, in Datei speichern
    def speichern(self, name, zeit):
        self.aendern(name, zeit)
        d = open("highscore.csv", "w")
        for element in self.liste:
            name = element[0]
            zeit = str(element[1]).replace(".", ",")
            d.write(name + ";" + zeit + "\n")
```

```

        d.close()
    # Liste anzeigen
    def __str__(self):
        # Highscore nicht vorhanden
        if not self.liste:
            return "Keine Highscores vorhanden"
        # Ausgabe Highscore
        ausgabe = "P. Name           Zeit\n"
        for i in range(len(self.liste)):
            ausgabe += "{0:2d}. {1:10} {2:5.2f} sec\n". \
                format(i+1, self.liste[i][0], \
                        self.liste[i][1])
        if i >= 9:
            break
    return ausgabe

```

**Listing 8.32** Datei `spiel_datei_oop.py`, Klasse »Highscore«

Zur Erläuterung:

- ▶ Im Konstruktor der Klasse wird die wichtigste Eigenschaft gesetzt: die Highscore-Liste mit dem Namen `liste`. Sie ist zunächst leer. Falls keine CSV-Datei existiert, bleibt sie leer. Falls es eine CSV-Datei gibt, so werden alle Zeilen daraus gelesen. Anschließend werden die Zeilen zerlegt. Aus dem Dezimalkomma (für Excel oder LibreOffice Calc) wird ein Dezimalpunkt. Die beiden Bestandteile werden der Highscore-Liste als Unterliste angehängt. Konstruktor, Eigenschaft
- ▶ Die Methode `aendern()` wird klassenintern von der Methode `speichern()` benötigt. Ein neu ermittelter Highscore wird entweder in die Liste eingefügt oder der Liste angehängt. Liste ändern
- ▶ In der Methode `speichern()` wird zunächst die Highscore-Liste geändert. Anschließend wird aus dem Dezimalpunkt ein Dezimalkomma, für Excel oder LibreOffice Calc. Die gesamte Liste wird dann in der CSV-Datei gespeichert. Dezimalzeichen
- ▶ In der Ausgabemethode wird die gesamte Liste veröffentlicht, falls sie nicht leer ist.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.



# Kapitel 9

## Internet

Dieses Kapitel beschäftigt sich mit dem Laden und dem Senden von Internetdaten und mit der Erstellung von Programmen, die auf einem Webserver laufen. Sie können mithilfe von Python auf Daten im Internet zugreifen, Daten ins Internet senden und Daten anderen Benutzern im Internet zur Verfügung stellen.

Zum Testen der Programme dieses Kapitels wird ein lokaler Webserver benötigt. XAMPP ist ein vorkonfiguriertes und einfach zu installierendes Paket, das neben einem *Apache*-Webserver weitere Software umfasst, z. B. die Webserver-Sprache *PHP* und das Datenbanksystem *MySQL*.

XAMPP

Sie können XAMPP unter <http://www.apachefriends.org> sowohl für Windows als auch für Ubuntu Linux und für OS X herunterladen. Die jeweils aktuellen Versionen stehen auch auf dem Datenträger zum Buch zur Verfügung. Die Installation von XAMPP wird in Anhang A.1 beschrieben.

Am Ende dieses Kapitels wird das bereits bekannte Spiel in einer Version vorgestellt, die das Spielen im Internet ermöglicht. Dabei werden Aufgabendaten aus dem Internet abgerufen, Benutzereingaben ins Internet gesendet und Spielergebnisse auf einem Webserver im Internet gespeichert.

### 9.1 Laden und Senden von Internetdaten

Das Modul `urllib` mit den Untermodulen `urllib.request` und `urllib.parse` kann zum Laden von Daten aus dem Internet und zum Senden von Daten in das Internet verwendet werden.

Modul `urllib`

Bei existierender Verbindung zu einem Webserver, ob lokal oder im Internet, haben Sie folgende Möglichkeiten:

- ▶ Öffnen einer Verbindung zu einer URL mit der Funktion `urlopen()` und anschließendes Einlesen der Inhalte in die Variable eines Python-Programms
- ▶ direktes Kopieren der Daten von einer URL in eine Datei auf der Festplatte mit der Funktion `urlretrieve()`

`urlopen()`

`urlretrieve()`

Mithilfe der Funktion `urlopen()` können Sie auch Daten an eine URL zur weiteren Verarbeitung auf dem Webserver senden.

Daten senden

## Unterschiede in Python 2

Die benötigten Funktionen haben dieselben Namen wie in Python 3, stehen aber im Modul `urllib` und nicht in den Untermodulen `urllib.request` und `urllib.parse`.

### 9.1.1 Daten lesen

#### HTML-Datei lesen

Im folgenden Programm wird mithilfe der Funktion `urlopen()` aus dem Modul `urllib.request` der Inhalt der Internetseite `http://localhost/Python34/url_leSEN.htm` in eine Liste gelesen. Diese Liste wird anschließend ausgegeben.

Bei der Datei `url_leSEN.htm` handelt es sich um eine einfache HTML-Datei. Zunächst wird der Inhalt dieser Datei aufgeführt:

```
<html>
<head>
<title>Titelzeile</title>
</head>

<body>
<b>Hallo Python</b>
</body>

</html>
```

**Listing 9.1** Datei `url_leSEN.htm`

**Speicherort** Zur Erläuterung:

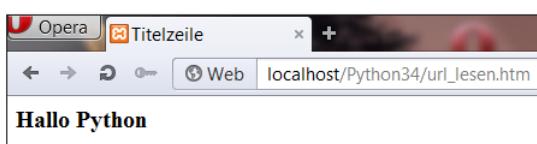
- ▶ Die Datei sollten Sie im Festplattenverzeichnis `C:\xampp\htdocs\Python34` speichern; dies setzt eine Standardinstallation von XAMPP voraus.

#### HTML lernen

- ▶ Es werden nur einfache HTML-Codebeispiele gegeben, da eine tiefergehende Einführung in HTML nicht Gegenstand dieses Buchs ist. Wenn Sie sich diesbezüglich genauer informieren möchten, so finden Sie ein empfehlenswertes Online-Buch unter <http://de.selfhtml.org>.

#### Aufruf über Browser

Geben Sie `http://localhost/Python34/url_leSEN.htm` in einem Webbrowser ein, so erscheint die in Abbildung 9.1 gezeigte Ansicht.



**Abbildung 9.1** Erstes HTML-Dokument

Das Python-Programm zum Lesen dieser Internetseite sieht wie folgt aus:

```
import sys, urllib.request

# Verbindung zu einer URL
try:
    u = urllib.request.urlopen \
        ("http://localhost/Python34/url_leSEN.htm")
except:
    print("Fehler")
    sys.exit(0)

# Liest alle Zeilen in eine Liste
li = u.readlines()

# Schliesst die Verbindung
u.close()

# Ausgabe der Liste
for element in li:
    print(element)
```

**Listing 9.2** Datei url\_leSEN.py

Die Ausgabe lautet:

```
b'<html>\r\n'
b'\r\n'
b'<head>\r\n'
b'<title>Titelzeile</title>\r\n'
b'</head>\r\n'
b'\r\n'
b'<body>\r\n'
b'<b>Hallo Python</b>\r\n'
b'</body>\r\n'
b'\r\n'
b'</html>\r\n'
```

Zur Erläuterung:

- ▶ Erscheint eine Fehlermeldung, so haben Sie eventuell vergessen, den lokalen Webserver zu starten (siehe Abschnitt 9.1.1, »Daten lesen«, und Abschnitt 9.1.2, »Daten kopieren«). Webserver gestartet?
- ▶ Mithilfe der Funktion `urlopen()` wird die Verbindung zur URL geöffnet. Der Rückgabewert der Funktion ähnelt einem Dateiobjekt. Sie können lesend auf die Inhalte der geöffneten Datei zugreifen, wie in Abschnitt 8.3.2, »Sequentielles Lesen«, beschrieben. `urlopen()`
- ▶ Die Funktion `readlines()` liefert eine Liste. Jedes Listenelement umfasst eine Zeile des HTML-Codes der Internetseite. `readlines()`

- close()** ► Nach dem Abspeichern in der Liste können Sie die Datei mit der Funktion `close()` wieder schließen.
- Byte-Literale** ► Jedes Element der Liste ist eine Zeichenkette, die als Byte-Literal ausgegeben wird. Byte-Literale sind Zeichenketten des Datentyps `bytes` (siehe [Abschnitt 4.2.7, »Datentyp »bytes««](#)).

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion `urlopen()` steht im Modul `urllib` und nicht im Untermodul `urllib.request`. Die Ausgabe erfolgt in »normaler« Textform und nicht als Folge von Byte-Literalen. Der Ablauf der Ausgabe wurde verändert, da ansonsten jeder Zeilenumbruch verdoppelt würde. Das Programm in Python 2:

```
import sys, urllib

# Verbindung zu einem URL
try:
    u = urllib.urlopen \
        ("http://localhost/Python27/url_lesen.htm")
except:
    print "Fehler"
    sys.exit(0)

# Liest alle Zeilen in eine Liste
li = u.readlines()

# Schliesst die Verbindung
u.close()

# Ausgabe der Liste
ausgabe = ""
for element in li:
    ausgabe += element
print ausgabe
```

**Listing 9.3** Datei `url_lesen.py` in Python 2

Die Ausgabe lautet:

```
<html>

<head>
<title>Titelzeile</title>
</head>

<body>
<b>Hallo Python</b>
</body>

</html>
```

### 9.1.2 Daten kopieren

Das folgende Programm kopiert den Inhalt der Internetseite `http://localhost/Python34/url_leSEN.htm` direkt in eine Datei auf der Festplatte.

HTML-Datei  
kopieren

```
import urllib.request

# Liest Inhalt von URL in eine Datei
urllib.request.urlretrieve \
    ("http://localhost/Python34/url_leSEN.htm", "url_kopieren.htm")
```

**Listing 9.4** Datei url\_kopieren.py

Zur Erläuterung:

- ▶ Die Funktion `urlretrieve()` hat zwei Parameter: die URL und den Namen der Datei, in der der Quellcode gespeichert werden soll (hier `url_kopieren.htm`).
- ▶ Diese Datei kann anschließend beispielsweise offline mit einem Browser gelesen oder mit einem Editor bearbeitet werden.

`urlretrieve()`

Offline browsen

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion `urlretrieve()` steht im Modul `urllib` und nicht im Untermodul `urllib.request`.

### 9.1.3 Daten senden per GET

Sie können dem Betrachter einer Website auch ermöglichen, spezifische Daten an den Webserver zu senden. Dies geschieht über Formulare, die der Betrachter ausfüllt und an den Webserver sendet.

Formular

Auf dem Webserver werden dann häufig zwei Vorgänge gestartet:

- ▶ Weiterverarbeitung oder Speicherung der Daten (z. B. in einer Datenbank)
- ▶ Senden einer individuellen Antwort an den Benutzer

#### HTML-PHP-Variante

Im folgenden Beispiel soll der Betrachter seinen Vor- und Nachnamen eingeben und diese Informationen an den Webserver senden. Er erhält daraufhin eine Bestätigung. Dies wird, zunächst für die Ein- und Ausgabe in einem Browser, mithilfe einer HTML-Datei und einem zugehörigen PHP-Programm realisiert. Abbildung 9.2 zeigt zunächst das Formular.

HTML-PHP-Variante

The screenshot shows a browser window with the URL `http://localhost/Python34/senden_get.htm`. The page content is:

**Bitte senden Sie Ihre Daten:**

Nachname  
 Vorname

Abbildung 9.2 Formular mit Beispieleingabe

Die Antwort des Webservers sehen Sie in [Abbildung 9.3](#).

The screenshot shows a browser window with the URL `http://localhost/Python34/senden_get.php`. The page content is:

**Ihre folgenden Daten wurden registriert:**

Nachname: Maier  
Vorname: Werner

Abbildung 9.3 Antwort des PHP-Programms

Der Quellcode des Formulars lautet:

```
<html>
<body>
<b>Bitte senden Sie Ihre Daten:</b><p>

<form action="senden_get.php" method="get">
<input name="nm"> Nachname<p>
<input name="vn"> Vorname<p>
<input type="submit">
</form>

</body>
</html>
```

Listing 9.5 Datei `senden_get.htm`

Zur Erläuterung:

- ▶ Das Formular wird mit `form` markiert, ein Eingabefeld mit `input`.
- ▶ Die im Formular verwendete GET-Methode ist wichtig für das Senden und Empfangen der Daten, sowohl für die HTML-PHP-Variante als auch für die später gezeigte Python-PHP-Variante.

#### GET-Methode

Es folgt der PHP-Quellcode der zugehörigen Antwort:

```
<html>
<body>
<b>Ihre folgenden Daten wurden registriert:</b><p>

<?php
echo "Nachname: " . $_GET["nn"] . "<br />";
echo "Vorname: " . $_GET["vn"];
?>

</body>
</html>
```

#### **Listing 9.6 Datei senden\_get.php**

Zur Erläuterung:

- ▶ Die aus dem Formular übernommenen Daten stehen in PHP im globalen Array `$_GET` zur Verfügung, weil sie mit der GET-Methode gesendet wurden.

#### **Python-PHP-Variante**

Mithilfe des folgenden Python-Formulars können diese Daten ebenfalls gesendet werden. Dabei wird wiederum das oben angegebene PHP-Programm *senden\_get.php* auf dem Webserver angesprochen.

Python-PHP-  
Variante

```
import urllib.request

# Eingabedaten
pnn = input("Bitte den Nachnamen eingeben: ")
pvn = input("Bitte den Vornamen eingeben: ")

# sendet Daten
u = urllib.request.urlopen \
    ("http://localhost/Python34/senden_get.php?nn="
     + pnn + "&vn=" + pvn)
# Empfang der Antwort und Ausgabe
li = u.readlines()
u.close()
for element in li:
    print(element)
```

#### **Listing 9.7 Datei senden\_get.py**

Zur Erläuterung:

- ▶ Zunächst werden die Eingabedaten vom Benutzer erfragt und in den beiden Variablen `pvn` und `pnn` gespeichert.

**urlencoded-Format**

- ▶ Mithilfe der Funktion `urlopen()` wird die betreffende URL angesprochen. Die Eingabeinformationen werden im Format *urlencoded* angehängt. Dazu werden nach einem Fragezeichen (?) ein oder mehrere Paare nach dem Muster `key=value` angehängt. Die einzelnen Paare werden jeweils durch ein Ampersand (&) voneinander getrennt. Der Schlüssel (`key`) entspricht der Variablen, der `value` dem Wert der Variablen.
- ▶ Anschließend wird der Code der Antwort des Webservers wie im ersten Programm gelesen und ausgegeben.

Die Ausgabe (für das gleiche Beispiel) lautet nun:

```
Bitte den Nachnamen eingeben: Maier
Bitte den Vornamen eingeben: Werner
b'<html>\r\n'
b'<body>\r\n'
b'<b>Ihre folgenden Daten wurden registriert:</b><p>\r\n'
b'\r\n'
b'Nachname: Maier<br />Vorname: Werner\r\n'
b'</body>\r\n'
b'</html>\r\n'
```

Zur Erläuterung:

**Daten zum Webserver gesendet**

- ▶ Wie Sie sehen, sind die Daten auf dem Webserver angekommen und wurden vom PHP-Programm weiterverarbeitet. In diesem Fall hätte man z. B. Nachname und Vorname in einer Datenbank auf dem Webserver speichern können.

**Unterschiede in Python 2**

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Die Funktion `urlopen()` steht im Modul `urllib` und nicht im Untermodul `urllib.request`. Die Ausgabe erfolgt in »normaler« Textform und nicht als Folge von Byte-Literalen. Der Ablauf der Ausgabe wurde verändert, da ansonsten jeder Zeilenumbruch verdoppelt würde.

**Python 2-Version**

Das Programm in Python 2:

```
import urllib

# Eingabedaten
pnn = raw_input("Bitte den Nachnamen eingeben: ")
pvn = raw_input("Bitte den Vornamen eingeben: ")

# sendet Daten
u = urllib.urlopen \
    (http://localhost/Python27/senden_get.php?nn=
     + pnn + "&vn=" + pvn)
```

```
# Empfang der Antwort und Ausgabe
li = u.readlines()
u.close()
ausgabe = ""
for element in li:
    ausgabe += element
print ausgabe
```

**Listing 9.8** Datei senden\_get.py in Python 2

Die Ausgabe lautet:

```
Bitte den Nachnamen eingeben: Maier
Bitte den Vornamen eingeben: Werner
<html>
<body>
<b>Ihre folgenden Daten wurden registriert:</b><p>
Nachname: Maier<br />Vorname: Werner
</body>
</html>
```

### 9.1.4 Daten senden per POST

Eine andere Möglichkeit bietet die Funktion `urlencode()`. Sie dient zur Codierung der Sendedaten und benötigt als Parameter ein Dictionary mit den zu sendenden Daten. Rückgabewert ist eine Zeichenkette im Format *urlencoded*, die an die URL angehängt wird.

`urlencode()`

#### HTML-PHP-Variante

Im folgenden Beispiel sehen Ein- und Ausgabe wie im vorherigen Abschnitt aus. Das HTML-Formular gestaltet sich wie folgt:

`HTML-PHP-Variante`

```
<html>
<body>
<b>Bitte senden Sie Ihre Daten:</b><p>

<form action="senden_post.php" method="post">
<input name="nn"> Nachname<p>
<input name="vn"> Vorname<p>
<input type="submit">
</form>

</body>
</html>
```

**Listing 9.9** Datei senden\_post.htm

Zur Erläuterung:

**POST-Methode**

- ▶ Im Unterschied zum vorherigen Beispiel wurde hier die sogenannte POST-Methode verwendet. Dies ist wichtig für das Senden und Empfangen der Daten, sowohl für die HTML-PHP-Variante als auch für die Python-PHP-Variante.

Es folgt der PHP-Quellcode der zugehörigen Antwort:

```
<html>
<body>
<b>Ihre folgenden Daten wurden registriert:</b><p>

<?php
echo "Nachname: " . $_POST["nn"] . "<br />";
echo "Vorname: " . $_POST["vn"];
?>

</body>
</html>
```

**Listing 9.10** Datei *senden\_post.php*

Zur Erläuterung:

**Array `$_POST`**

- ▶ Die aus dem Formular übernommenen Daten stehen in PHP im globalen Array `$_POST` zur Verfügung, weil sie mit der POST-Methode gesendet wurden.

### Python-PHP-Variante

**Python-PHP-Variante**

Mithilfe des folgenden Python-Programms können diese Daten ebenfalls gesendet werden. Dabei wird auch das obige PHP-Programm *senden\_post.php* auf dem Webserver angesprochen.

```
import urllib.request, urllib.parse

# Eingabedaten
pnn = input("Bitte den Nachnamen eingeben: ")
pvn = input("Bitte den Vornamen eingeben: ")

# Dictionary mit Sendedaten, Codierung
dc = {b"nn":pnn, b"vn":pvn}
data = bytes(urllib.parse.urlencode(dc), "UTF-8")
# sendet Daten
u = urllib.request.urlopen \
    ("http://localhost/Python34/senden_post.php", data)

# Empfang der Antwort und Ausgabe
li = u.readlines()
```

```
u.close()
for element in li:
    print(element)
```

**Listing 9.11** Datei senden\_post.py

Zur Erläuterung:

- ▶ Aus den eingegebenen Daten wird ein Dictionary erzeugt. Schlüssel ist Dictionary jeweils der Name der Variablen, Wert ist der Eingabewert des Benutzers.
- ▶ Dieses Dictionary wird mithilfe der Funktion `urlencode()` in ein Format umgewandelt, das zum Senden geeignet ist. Sendedaten
- ▶ Der zweite, optionale Parameter der Funktion `urlopen()` enthält diese Sendedaten.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Die Funktionen stehen im Modul `urllib` und nicht in den Untermodulen `urllib.request`, `urllib.parse`. Die Ausgabe erfolgt in »normaler« Textform und nicht als Folge von Byte-Literalen. Der Ablauf der Ausgabe wurde verändert, da sonst jeder Zeilenumbruch verdoppelt würde.

Das Programm in Python 2:

```
import urllib

# Eingabedaten
pnn = raw_input("Bitte den Nachnamen eingeben: ")
pvn = raw_input("Bitte den Vornamen eingeben: ")

# Dictionary mit Sendedaten, Codierung
dc = {"nn":pnn, "vn":pvn}
data = urllib.urlencode(dc)

# sendet Daten
u = urllib.urlopen \
    ("http://localhost/Python27/senden_post.php", data)

# Empfang der Antwort und Ausgabe
li = u.readlines()
u.close()
ausgabe = ""
for element in li:
    ausgabe += element
print ausgabe
```

**Listing 9.12** Datei senden\_post.py in Python 2

Python 2-Version

## 9.2 Webserver-Programmierung

<b>CGI-Skript</b>	Das Modul <code>cgi</code> wird zur Webserver-Programmierung verwendet. Mit seiner Hilfe können Sie sogenannte CGI-Skripte erzeugen, die auf einem Webserver ausgeführt werden und ihre Ergebnisse an den Betrachter senden.
	Basis für die gesendeten, dynamisch erzeugten Ergebnisse sind unter anderem Benutzereingaben, Inhalte von Dateien und Datenbanken, die auf dem Webserver liegen und mithilfe der CGI-Skripte gelesen und bearbeitet werden.
<b>Lokaler Webserver</b>	Voraussetzung für die Entwicklung und den Test der folgenden Programme ist ein lokaler Webserver, wie er bereits für die Programme im vorherigen Abschnitt benötigt wurde.
<b>Verzeichnis cgi-bin</b>	Die CGI-Skripte werden im Browser über das Verzeichnis mit der Adresse <code>http://localhost/cgi-bin/Python34</code> (bzw. <code>http://localhost/cgi-bin/Python27</code> ) abgerufen. Dieses Verzeichnis entspricht bei einer Standardinstallation von XAMPP unter Windows dem Festplattenverzeichnis <code>C:\xampp\cgi-bin\Python34</code> (bzw. <code>C:\xampp\cgi-bin\Python27</code> ).
<b>Servermodule</b>	Es gibt auch Möglichkeiten, Python-Programme über bestimmte Module auszuführen, die im Webserver eingebettet werden. Hier sind <code>mod_python</code> und sein Nachfolger <code>mod_wsgi</code> zu nennen. Die Python-Programme, die diese Module nutzen, sind schneller als CGI-Skripte, allerdings gestaltet sich die Installation dieser Module auf dem lokalen Webserver als recht aufwendig. Im Rahmen dieses Einsteigerbuchs wird auf diese Themen nicht eingegangen.
<b>CGI unter Linux</b>	Die CGI-Skripte in diesem Abschnitt werden unter Windows vorgeführt. Die Konfiguration der CGI-Schnittstelle ist unter Ubuntu Linux und unter OS X ebenfalls recht aufwendig. Auch darauf wird hier nicht eingegangen.

### 9.2.1 Erstes Programm

<b>Statisches CGI-Skript</b>	Im Folgenden wird zunächst ein einfaches, statisches CGI-Skript mit Python vorgeführt. Das Modul <code>cgi</code> wird hier noch nicht benötigt. In diesem Beispiel sollen nur der Python-Programmaufbau und der HTML-Dokumentaufbau gezeigt werden:
	<pre>#!C:\Python34\python.exe print("Content-type: text/html") print() print("&lt;html&gt;") print("&lt;body&gt;") print("&lt;h1&gt;Hallo Python 3.4&lt;/h1&gt;") print("&lt;/body&gt;") print("&lt;/html&gt;")</pre>

**Listing 9.13** Datei server\_hallo.cgi

Zur Erläuterung:

- ▶ In der ersten Zeile wird dem Webserver (unter Windows) die Information gegeben, dass die folgenden Zeilen mithilfe von Python ausgeführt werden sollen. Der Webserver weiß damit, dass er keine normale HTML-Datei vor sich hat, deren Daten einfach an den Browser des Benutzers gesendet werden sollen, sondern dass der Dateiinhalt erst von Python vorverarbeitet werden muss. Erst Python erstellt dann die HTML-Daten.
- ▶ Gleichzeitig wird der Ort mitgeteilt, an dem sich der Python-Interpreter auf dem Rechner befindet, auf dem auch der Webserver läuft. Im vorliegenden Fall handelt es sich um die Datei *C:\Python34\python.exe*, also lautet die Zeile: `#!C:\Python34\python.exe`.
- ▶ In der nächsten Zeile wird im sogenannten Header das Format des Dokumentinhalts (`Content-type: text/html`) festgelegt. Anschließend muss eine Leerzeile ausgegeben werden!
- ▶ Es folgt die Ausgabe der Zeilen des eigentlichen HTML-Quellcodes.

Python auf dem  
Webserver

`python.exe`

Header

Die Ausgabe im Browser, nach Eingabe der Adresse `http://localhost/cgi-bin/Python34/server_hallo.cgi`, sehen Sie in Abbildung 9.4.



Abbildung 9.4 Erstes Python-HTML-Dokument

Falls eine Python-Datei Umlaute oder Eszetts enthält, kann es passieren, dass beim Aufruf der Datei ein Fehler gemeldet wird. Um dies zu vermeiden, geben Sie als zweite Zeile in der Datei ein:

```
# -*- coding: cp1252 -*-
```

encoding/decoding

Dadurch wird eine Codepage eingestellt, die die richtige Darstellung und Nutzung der Umlaute und Eszetts ermöglicht.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die erste Zeile lautet `#!C:\Python27\python.exe`. Der ausgegebene Text lautet `Hallo Python 2.7.`

## 9.2.2 Beantworten einer Benutzereingabe

### Dynamisches CGI-Skript

Im folgenden Beispiel soll der Betrachter seinen Vor- und Nachnamen eingeben und diese Informationen an den Webserver senden. Er erhält daraufhin eine Bestätigung. Abbildung 9.5 zeigt das Formular.

The screenshot shows a web browser window with the URL `http://localhost/Py...x`. The title bar says "Opera". The page content is a form with the heading "Bitte senden Sie Ihre Daten:". It contains two input fields: one for "Nachname" with the value "Maier" and one for "Vorname" with the value "Werner". Below the inputs is a "Senden" button.

**Abbildung 9.5** Formular mit Beispieleingabe

Die Antwort des Webservers sehen Sie in Abbildung 9.6.

The screenshot shows a web browser window with the URL `http://localhost/c...x`. The title bar says "Opera". The page content displays the registered data under the heading "Registrierte Daten:". It shows "Nachname: Maier" and "Vorname: Werner".

**Abbildung 9.6** Antwort des Python-Programms

### HTML-Python-Variante

Unser bekanntes Beispiel realisieren wir nun mithilfe einer HTML-Datei und einem zugehörigen Python-Programm als CGI-Skript.

Zunächst der Code der HTML-Datei mit dem Formular:

```
<html>
<body>
<b>Bitte senden Sie Ihre Daten:</b><p>

<form action="/cgi-bin/Python34/server_antworten.cgi">
<input name="nn"> Nachname<p>
<input name="vn"> Vorname<p>
<input type="submit">
</form>

</body>
</html>
```

**Listing 9.14** Datei `server_antworten.htm`

Zur Erläuterung:

- ▶ In der `form`-Markierung ist der Name des Python-Programms angegeben, das die Antwort zu diesem Formular sendet. Es handelt sich um die Datei `server_antworten.cgi` im Unterverzeichnis `Python34` des Standardverzeichnisses für CGI-Skripte (`cgi-bin`) auf dem Webserver. Formularziel

Das zugehörige CGI-Skript sieht wie folgt aus:

```
#!C:\Python34\python.exe

# Modul cgi
import cgi, cgitb
# Ausgabe bei Fehler
cgitb.enable()

# Objekt der Klasse FieldStorage
form = cgi.FieldStorage()

# Einzelne Elemente des Objekts
if "nn" in form:
    nn = form["nn"].value
if "vn" in form:
    vn = form["vn"].value

# HTML-Dokument mit Variablen
print("Content-type: text/html")
print()

print("<html>")
print("<body>")

print("<p><b>Registrierte Daten:</b></p>")
print("<p>Nachname:", nn, "</p>")
print("<p>Vorname:", vn, "</p>")

print("</body>")
print("</html>")
```

#### **Listing 9.15 Datei server\_antworten.cgi**

Zur Erläuterung:

- ▶ Es werden die Module `cgi` und `cgitb` eingebunden. Module `cgi`, `cgitb`
- ▶ Das Modul `cgi` wird für die eigentliche Datenübermittlung benötigt. Das Modul `cgitb` können Sie während der Entwicklungszeit zur Fehlersuche nutzen.
- ▶ Die Funktion `enable()` des Moduls `cgitb` sorgt dafür, dass Fehler während der Entwicklung des Python-Programms mit Kommentar im Browser ausgegeben werden. Ein Beispiel sehen Sie am Ende des Abschnitts. Nach Fertigstellung `cgitb.enable()`

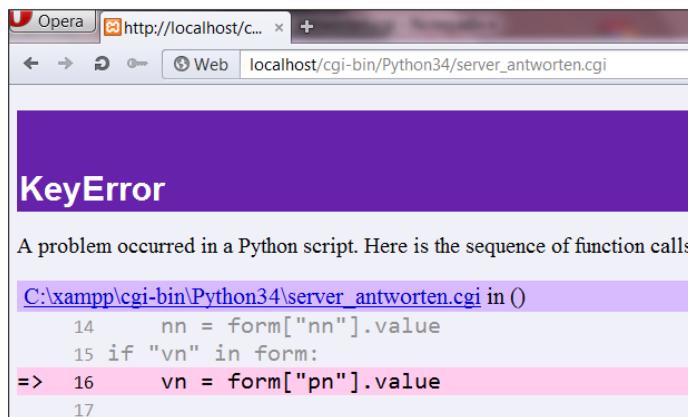
lung eines Programms sollten Sie die Zeile mit `cgitb.enable()` wieder auskommentieren.

- FieldStorage** ► Es wird ein Objekt (hier mit dem Namen `form`) der Klasse `FieldStorage` des Moduls `cgi` erzeugt. Dieses Objekt ähnelt einem Dictionary und enthält alle ausgefüllten Elemente des Formulars mit ihren jeweiligen Werten.
- Formularelement prüfen** ► Es muss zunächst geprüft werden, ob der Benutzer einen Wert für ein bestimmtes Formularelement eingegeben hat oder nicht. Dies geschieht mithilfe des Operators `in` (Beispiel: `if "nn" in form`). Formularelemente ohne Wert werden nicht an das CGI-Programm übermittelt. Ihre Verwendung würde dann zu einem Laufzeitfehler führen.
- Wert des Formularelements** ► Die Anweisung `nn = form["nn"].value` weist darauf hin der Python-Variablen `nn` den Wert des Formularelements `nn` zu, das in dem Objekt `form` gespeichert wurde.
- Header** ► Es folgt die Ausgabe des HTML-Codes, in dem die ermittelten Python-Variablen eingesetzt werden. Der Code wird eingeleitet durch die Header-Zeile, in der das Format des Dokumentinhalts (`text/html`) festgelegt wird. Anschließend folgt eine Leerzeile.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die erste Zeile lautet `#!C:\Python27\python.exe`. Das Ziel des HTML-Formulars ist entsprechend `/cgi-bin/Python27/server_antworten.cgi`.

- Fehlerausgabe** Ergänzend zum Einsatz des Moduls `cgitb` veranschaulicht das Beispiel in Abbildung 9.7 ein fehlerhaftes Python-Programm und die Ausgabe des Fehlers über den Browser.



The screenshot shows a browser window with the address bar containing `localhost/cgi-bin/Python34/server_antworten.cgi`. The main content area has a purple header with the text **KeyError**. Below it, the error message reads: "A problem occurred in a Python script. Here is the sequence of function calls". Underneath, the Python code is displayed with line numbers:

```

C:\xampp\cgi-bin\Python34\server_antworten.cgi in ()
 14     nn = form["nn"].value
 15 if "vn" in form:
=> 16     vn = form["pn"].value
 17

```

Abbildung 9.7 Fehlermeldung im Browser

Zur Erläuterung:

- Statt der Variablen `vn` für den Vornamen wurde versehentlich die Variable `pn` gewählt.

### 9.2.3 Formularelemente mit mehreren Werten

Formularelemente können mehrere verschiedene Werte enthalten. Dies ist z. B. bei einem Auswahlmenü vom Typ `multiple` der Fall, in dem der Benutzer mehrere Werte markieren kann.

Zur vereinfachten Formularauswertung können aber auch mehrere Formularelemente den gleichen Namen tragen. Auch in diesem Fall wird zu einem Namen mehr als ein Wert übermittelt.

Wird nur ein Wert zu einem solchen Formularelement übermittelt, so geschieht die Auswertung in der bereits bekannten Form. Falls mehrere Werte zu einem Formularelement übermittelt wurden, so stehen diese Werte in einer Liste zur Verfügung. Bei der Auswertung muss also zunächst festgestellt werden, ob

Auswertung

- überhaupt ein Wert vorhanden ist oder
- genau ein Wert vorhanden ist oder
- mehrere Werte vorhanden sind.

Das Beispieldokument in Abbildung 9.8 enthält drei gleichnamige Textfelder.

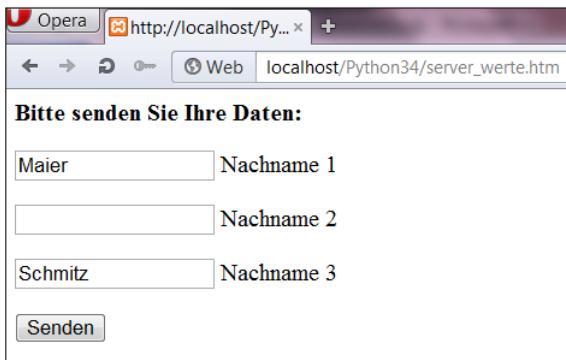


Abbildung 9.8 Formular mit Beispieleingabe

Die Antwort des Webservers zeigt Abbildung 9.9.

Der HTML-Code des Formulars:

```
<html>
<body>
<b>Bitte senden Sie Ihre Daten:</b><p>
```

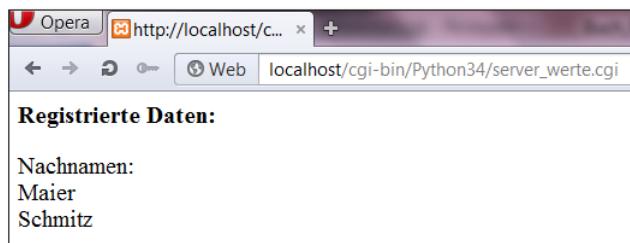
```

<form action="/cgi-bin/Python34/server_werte.cgi">
<input name="nn"> Nachname 1<p>
<input name="nn"> Nachname 2<p>
<input name="nn"> Nachname 3<p>
<input type="submit">
</form>

</body>
</html>

```

**Listing 9.16** Datei server\_werte.htm



**Abbildung 9.9** Antwort des Python-Programms

Zur Erläuterung:

- ▶ Das Formular wird zur Auswertung an *server\_werte.cgi* gesendet. Es enthält dreimal das Formularelement `nn`. Es können also bis zu drei verschiedene Nachnamen eingegeben werden.

Das zugehörige CGI-Skript sieht wie folgt aus:

```

#!/usr/bin/python

# Modul cgi
import cgi, cgitb

# Ausgabe bei Fehler
cgitb.enable()

# Objekt der Klasse FieldStorage
form = cgi.FieldStorage()

# HTML-Dokument mit Variablen
print("Content-type: text/html")
print()

# Auswertung Formularfeld
if "nn" in form:
    print("<p><b>Registrierte Daten:</b></p>")
    print("<p>Nachnamen:<br />")

```

```

try:
    print(form["nn"].value)
except:
    for element in form["nn"]:
        print(element.value, "<br />")
    print("</p>")
else:
    print("<p><b>Keine Daten gesendet</b></p>")

print("</body>")
print("</html>")

```

**Listing 9.17** Datei server\_werte.cgi

Zur Erläuterung:

- ▶ Zunächst wird mithilfe des Operators `in` festgestellt, ob überhaupt ein Nachname eingegeben wurde. Kein Wert
- ▶ Ist dies nicht der Fall, so wird ausgegeben: Keine Daten gesendet. Ein Wert
- ▶ Innerhalb eines `try-except`-Blocks wird versucht, einen einzelnen Nachnamen auszugeben. Falls mehrere Nachnamen übermittelt wurden, schlägt dies fehl.
- ▶ In diesem Fall werden alle Elemente der übermittelten Liste von Nachnamen ausgegeben. Mehrere Werte

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die erste Zeile lautet `#!C:\Python27\python.exe`. Das Ziel des HTML-Formulars ist entsprechend `/cgi-bin/Python27/server_werte.cgi`.

## 9.2.4 Typen von Formularelementen

Im Folgenden demonstriere ich die Auswertung verschiedener Typen von Formularelementen an einem größeren Beispiel – einer Online-Bestellung beim Python Pizza Service (siehe [Abbildung 9.10](#)).

**Online-Formular**

Das Formular enthält die folgenden Typen von Formularelementen:

**Elementtypen**

- ▶ zwei Radiobuttons für den Kundentyp
- ▶ ein Eingabefeld vom Typ `password` für den Stammkunden-Code
- ▶ sechs einzeilige Eingabefelder für die Adresse
- ▶ ein einfaches Auswahlmenü für die Pizzasorte
- ▶ ein mehrfaches Auswahlmenü für die Zusätze zur Pizza
- ▶ ein Kontrollkästchen (`checkbox`) für den Express-Service

- ▶ ein mehrzeiliges Eingabefeld (textarea) für Bemerkungen
- ▶ je einen Button zum Absenden und zum Zurücksetzen der Bestellung

The screenshot shows a web page titled "Python Pizza Service". At the top, there are two radio buttons: "Stammkunde (5% Rabatt)" (selected) and "Neukunde". Below them are four input fields arranged in a grid: "Maier" (Name), "Nachname Werner" (Last Name), "Vorname" (First Name), "Pepperoniweg" (Street), "Straße 4" (House Number), "Hausnr." (House Number), "63999" (Postcode), "PLZ Teigdorf" (City), and "Ort". A dropdown menu for "Sorte" (Type) shows "Pizza Python (8.50 €)". Below it, a dropdown menu for "Zusätze" (Add-ons) shows "Extra Pepperoni (1.00 €)" and "Extra Oliven (1.20 €)". A checked checkbox for "Express-Service (max. 30 Minuten, 1.50 € extra)" is present. A text area for "Bemerkung" (Remark) contains the placeholder "Bitte zweimal klingeln". At the bottom are two buttons: "Senden" (Send) and "Zurücksetzen" (Reset).

Abbildung 9.10 Python Pizza Service, Bestellung

**Vorbesetzung** Die Formularelemente, die eine einfache Auswahl verlangen, sind schon mit einem Standardwert vorbesetzt. Darauf sollten Sie beim Formularentwurf unbedingt achten, damit der Benutzer nur sinnvolle Inhalte absenden kann. Gleichzeitig muss im auswertenden Programm nicht mehr geprüft werden, ob für diese Elemente ein Wert vorhanden ist.

**HTML-Formular** Das HTML-Formular sieht folgendermaßen aus:

```
<html>
<body>
<p><b>Python Pizza Service</b></p>

<form action="/cgi-bin/Python34/server_pizza.cgi">

<p><input type="radio" name="ku" value="S" checked="checked">
Stammkunde (5 % Rabatt)
<input type="password" name="kc" size="6" maxlength="6">
Code<br />
<input type="radio" name="ku" value="N"> Neukunde</p>
```

```

<table>
<tr>
  <td><input name="nn" size="12"> Nachname</td>
  <td><input name="vn" size="12"> Vorname</td>
</tr>
<tr>
  <td><input name="st" size="12"> Stra&szlig;e</td>
  <td><input name="hn" size="12"> Hausnr.</td>
</tr>
<tr>
  <td><input name="pz" size="12"> PLZ</td>
  <td><input name="or" size="12"> Ort</td>
</tr>
</table>

<p>Sorte:
<select name="pt">
<option value="Salami">
Pizza Salami (6.00 &euro;)</option>
<option value="Thunfisch">
Pizza Thunfisch (6.50 &euro;)
<option value="Quattro Stagioni">
Pizza Quattro Stagioni (7.50 &euro;)</option>
<option value="Python" selected="selected">
Pizza Python (8.50 &euro;)</option>
</select></p>

<p>Zus&auml;tze:
<select name="zu" multiple size="2">
<option value="Pepperoni">Extra Pepperoni (1.00 &euro;)</option>
<option value="Oliven">Extra Oliven (1.20 &euro;)</option>
<option value="Sardellen">Extra Sardellen (1.50 &euro;)</option>
</select></p>

<p><input type="checkbox" name="ex">
Express-Service (max. 30 Minuten, 1.50 &euro; extra)</p>

<p>Bemerkung:
<textarea name="bm" rows="3" cols="25"></textarea></p>

<p><input type="submit"> <input type="reset"></p>
</form>

</body>
</html>

```

**Listing 9.18** Datei server\_pizza.htm

Zur Erläuterung:

- |                               |   |
|-------------------------------|---|
| <b>Radiobuttons</b>           | ► Eingabefelder vom Typ <code>radio</code> (Radiobuttons), die den gleichen Namen haben (im vorliegenden Beispiel "ku"), bieten eine einfache Auswahl zwischen zwei Alternativen, hier zwischen <i>Stammkunde</i> und <i>Neukunde</i> . Die Vorauswahl wird durch <code>checked</code> gekennzeichnet.  |
| <b>Passwortfeld</b>           | ► Eingaben in Eingabefeldern vom Typ <code>password</code> sind nicht erkennbar.  |
| <b>Einfaches Auswahlmenü</b>  | ► Einfache Auswahlmenüs mit <code>select</code> dienen zur einfachen Auswahl zwischen verschiedenen Möglichkeiten, hier zwischen den unterschiedlichen Pizzasorten. Die Vorauswahl wird durch <code>selected</code> gekennzeichnet.   |
| <b>Mehrfaches Auswahlmenü</b> | ► Mehrfache Auswahlmenüs mit <code>select</code> und <code>multiple</code> bieten eine mehrfache Auswahl zwischen den verschiedenen Möglichkeiten, hier den unterschiedlichen Zusätzen zu den Pizzen. Eine mehrfache Auswahl erfolgt mithilfe der Tasten <code>[Shift]</code> und <code>[Strg]</code> . |
| <b>Checkbox</b>               | ► Ein Kontrollkästchen (engl.: <i>checkbox</i> ) dient zur Übermittlung einer zusätzlichen Information, hier dem Wunsch nach Express-Service.   |
| <b>Textarea</b>               | ► Ein mehrzeiliges Eingabefeld ( <code>Textarea</code> ) bietet die Möglichkeit, längere Texte einzutragen und zu übermitteln.  |

Die Antwort des Webservers auf die obige Bestellung (siehe Abbildung 9.11) erfolgt mit einem Python-CGI-Skript.



**Abbildung 9.11** Python Pizza Service, Bestätigung der Bestellung

- |                          |  |
|--------------------------|--|
| <b>Python-CGI-Skript</b> | Das zugehörige CGI-Skript sieht wie folgt aus: |
|--------------------------|--|

```
#!C:\Python34\python.exe
```

```

# Modul cgi
import cgi, cgitb
# Allgemeine Check-Funktion
def chk(element):
    global form

```

```

if element in form:
    erg = form[element].value
else:
    erg = ""
return erg

# Ausgabe bei Fehler
cgitb.enable()

# Objekt der Klasse FieldStorage
form = cgi.FieldStorage()

print("Content-type: text/html")
print()
print("<html>")
print("<body>")

# Anrede
print("<p>Guten Tag", chk("vn"), chk("nn"), "</p>")

# Adresse
print("<p>Ihre Adresse:", chk("st"), chk("hn"),
      "in", chk("pz"), chk("or"), "</p>")

# Pizza-Typ
print("<p>Ihre Bestellung: Pizza", chk("pt"))

# Beginn Berechnung Preis
preisliste = {"Salami":6, "Thunfisch":6.5,
              "Quattro Stagioni":7.5, "Python":8.5}
preis = preisliste[form["pt"].value]

# Zusatz
zusatzliste = {"Pepperoni":1, "Oliven":1.2,
                "Sardellen":1.5}
if "zu" in form:
    try:
        print("mit", form["zu"].value)
        preis += zusatzliste[form["zu"].value]
    except:
        for element in form["zu"]:
            print(", mit", element.value)
            preis += zusatzliste[element.value]
print("</p>")

# Express-Service
if "ex" in form:
    print("<p>Mit Express-Service</p>")
    preis += 1.5

```

```

# Bemerkung
if "bm" in form:
    print("<p>Bemerkung:", form["bm"].value, "</p>")

# Rabatt
if "kc" in form:
    if form["ku"].value == "S"
        and form["kc"].value == "Bingo":
        preis = preis * 0.95;
    print("<p>Rabatt 5 %</p>")

# Preis
print("Preis (ohne Bemerkung): {0:.2f} &euro;".format(preis))
print("</body>")
print("</html>")

```

**Listing 9.19** Datei server\_pizza.cgi

Zur Erläuterung:

- |                                  |   |
|----------------------------------|---|
| <b>Allgemeine Check-Funktion</b> | ► Es wird zunächst eine allgemeine Check-Funktion für alle Formularelemente definiert, die maximal einen Wert übermitteln können. Diese Funktion liefert den Wert des überprüften Elements oder eine leere Zeichenkette zurück. |
| <b>Eingabefelder</b>             | ► Nach der Überschrift folgt die Anrede. Für die Angabe von Vorname und Nachname wird die Check-Funktion genutzt.<br>► Es folgt die Bestätigung der Adresse, ebenfalls mit Nutzung der Check-Funktion.                          |
| <b>Einfaches Auswahlmenü</b>     | ► Die Pizzasorte wird angegeben, und der zugehörige Preis wird über ein Dictionary berechnet. Die Existenz des Elements muss nicht überprüft werden, da bereits eine Vorauswahl getroffen wurde.                                |
| <b>Mehrfaches Auswahlmenü</b>    | ► Die optionalen Zusätze für die Pizza werden in der Ausgabe und bei der Berechnung des Preises (über ein Dictionary) hinzugefügt. Hier muss überprüft werden, ob der Benutzer Zusätze gewählt hat.                             |
| <b>Checkbox, Textarea</b>        | ► Es erfolgt die Prüfung und gegebenenfalls Ausgabe bzw. Preisberechnung für Express-Service und die Ausgabe der Bemerkung.   |
| <b>Passwortfeld</b>              | ► Nach Berechnung eines eventuell zu gewährenden Rabatts für Stammkunden wird der Preis ausgegeben. Dies geschieht allerdings nur, falls der Code »Bingo« im Passwortfeld eingegeben wurde.                                     |

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt. Die erste Zeile lautet `#!/C:\Python27\python.exe`. Das Ziel des HTML-Formulars ist entsprechend `/cgi-bin/Python27/server_pizza.cgi`.

### 9.3 Browser aufrufen

Das Modul `webbrowser` enthält Funktionen, um Internetseiten über einen Browser abzurufen. Es genügt, die Funktion `open()` aufzurufen. Als Parameter geben Sie die URL der gewünschten Internetseite an:

```
import webbrowser
webbrowser.open("http://www.galileo-press.de")
print("Ende")
```

**Listing 9.20** Datei `browser.py`

Zur Erläuterung:

- ▶ Der Standard-Webbrowser des Systems wird als unabhängiges Programm aufgerufen. Danach läuft das Python-Programm weiter.

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 9.4 Spiel, Version für das Internet

In dieser Spielversion kann der Spieler in einem HTML-Formular auf einer Internetseite zunächst seinen Namen eingeben. Anschließend werden ihm, wiederum in einem HTML-Formular, fünf Additionsaufgaben mit Zahlen aus dem Bereich von 10 bis 30 gestellt.

Der Spieler gibt seine Lösungen in die fünf zugehörigen Eingabefelder ein. Pro Aufgabe hat er nur einen Versuch. Am Ende wird die Gesamtzeit ermittelt, die er für seine Lösungsversuche benötigt hat.

Löst der Spieler alle fünf Aufgaben richtig, so werden sein Name und die benötigte Zeit in eine Highscore-Liste aufgenommen, die in einer Datei auf dem Webserver abgespeichert wird.

`webbrowser.  
open()`

**Aufgaben im  
Browser**

**Highscore speichern**

#### 9.4.1 Eingabebeispiel

Es folgt ein typischer Durchlauf durch das Programm. Abbildung 9.12 zeigt zunächst das Formular für die Eingabe des Namens.



Abbildung 9.12 Beginn des Spiels

Es folgen die fünf Aufgaben, siehe Abbildung 9.13.

The screenshot shows a web page titled "Kopfrechnen". It greets "Hallo Mario, Ihre Aufgaben" and lists five addition problems:

1.  $27 + 29 =$
2.  $15 + 14 =$
3.  $24 + 29 =$
4.  $30 + 27 =$
5.  $16 + 14 =$

At the bottom is a "Fertig" button.

Abbildung 9.13 Fünf Aufgaben mit Lösungsversuchen

Die Bewertung des Ergebnisses und die Highscore-Liste sehen Sie in Abbildung 9.14.

The screenshot shows a web page titled "Kopfrechnen". It greets "Hallo Mario, Ihr Ergebnis" and displays "5 von 5 in 9.53 Sekunden, Highscore". Below is a table of results:

Platz	Name	Zeit
1	Mario	9.53 sec
2	Stefan	11.35 sec

At the bottom is a "Zum Start" button.

Abbildung 9.14 Bewertung, Highscore

Zur Erläuterung:

- ▶ Der Spieler gibt seinen Namen ein (»Mario«) und betätigt den Button UND LOS ...
- ▶ Er bekommt fünf Aufgaben gestellt, gibt seine Lösungen für die Aufgaben ein und betätigt den Button FERTIG.
- ▶ Die Eingaben werden bewertet, und es erscheint die Highscore-Liste, gegebenenfalls mit einem neuen Eintrag für den aktuellen Spieler.
- ▶ Das Programm hat einen kleinen Schönheitsfehler: Es misst nicht die Zeit, die der Spieler zum Lösen der Aufgaben tatsächlich benötigt, sondern die Zeit, die zwischen dem Erscheinen der fünf Aufgaben und dem Erscheinen der Auswertungsseite verstreicht. Somit fließt neben der Schnelligkeit des Spielers auch die Qualität der Internetverbindung in das Ergebnis ein. Eine aufwendigere Lösung würde dieses Problem umgehen.

## 9.4.2 Aufbau des Programms

Das Programm besteht aus drei Dateien:

### Erstes Formular

- ▶ Bei der ersten Datei handelt es sich um eine reine HTML-Datei (*http://localhost/Python34/spiel\_server.htm*). Darin steht das Formular mit dem Feld zur Eingabe des Namens.

Eingabe Name

### Zweites Formular

- ▶ Bei der zweiten Datei handelt es sich um ein Python-CGI-Skript (*http://localhost/cgi-bin/Python34/spiel\_server\_a.cgi*). In diesem CGI-Skript wird der Eingabewert aus dem ersten Formular entgegengenommen, und das zweite Formular wird dynamisch aufgebaut. Das zweite Formular enthält fünf Eingabefelder für die Lösungsversuche der fünf Aufgaben.
- ▶ Außerdem enthält das zweite Formular insgesamt sieben versteckte Felder. In diesen Feldern werden die fünf richtigen Lösungen der Aufgaben, der Name des Spielers aus dem ersten Formular und die ermittelte Startzeit unsichtbar an die dritte Datei übermittelt.

Fünf Aufgaben

Versteckte Felder

### Auswertung

- ▶ Die dritte Datei ist wiederum ein Python-CGI-Skript (*http://localhost/cgi-bin/Python34/spiel\_server\_b.cgi*). In diesem Skript werden insgesamt zwölf Werte aus dem zweiten Formular entgegengenommen. Dies sind:
  - der Spielername
  - die ermittelte Startzeit

- die fünf richtigen Lösungen
- die fünf Eingaben des Benutzers

<b>Ergebniszeit</b>	► Es wird die Endzeit festgestellt. Aus Startzeit und Endzeit wird die Ergebniszeit ermittelt, die der Benutzer benötigt hat.
<b>Auswertung</b>	<ul style="list-style-type: none"> <li>► Die fünf richtigen Lösungen und die fünf Eingaben des Benutzers werden miteinander verglichen. Sind alle Lösungen richtig, werden Name und Zeit in die Highscore-Liste eingetragen. Die Liste wird anschließend angezeigt.</li> <li>► Zuletzt wird ein Hyperlink angezeigt, mit dem der Benutzer wieder zum Start des Programms gelangt.</li> </ul>

### 9.4.3 Code des Programms

#### Erstes Formular

```
<html>
<body>
<p><b>Kopfrechnen</b></p>

<form action="/cgi-bin/Python34/spiel_server_a.cgi">
<p><input name="spielername" size="12"> Ihr Name</p>
<p><input type="submit" value="Und los ..."></p>
</form>

</body>
</html>
```

**Listing 9.21** Datei spiel\_server.htm

Zur Erläuterung:

- Dieses erste Formular verweist auf das zweite Formular. Es enthält das Element spielername und den Button zum Absenden.

#### Zweites Formular

```
#!/C:/Python34/python.exe

# Module
import cgi, cgitb, sys, random, time

# Ausgabe bei Fehler
cgitb.enable()
# Initialisierung Zufallsgenerator
random.seed()

# Objekt der Klasse FieldStorage
form = cgi.FieldStorage()
```

```

# Header
print("Content-type: text/html")
print()

# HTML-Dokumentbeginn
print("<html>")
print("<body>")
print("<p><b>Kopfrechnen</b></p>")
# Falls kein Name eingetragen
if not "spielername" in form:
    print("<p>Kein Name, kein Spiel</p>")
    print("<a href='http://localhost/Python34/>
          "spiel_server.htm'>Zur&ampuumlck</a>")
    print("</body>")
    print("</html>")
    sys.exit(0)

# Formularbeginn
print("<form action='spiel_server_b.cgi'>")

# Spielername
print("<p>Hallo <b>", form["spielername"].value,
      "</b>, Ihre Aufgaben</p>")
print("<td><input name='spielername' type='hidden' "
      "value='" + form["spielername"].value + "'>")

# Startzeit
startzeit = time.time()
print("<td><input name='startzeit' type='hidden' "
      "value='" + str(startzeit) + "'>")

# Tabellenbeginn
print("<table border='0'>")

# 5 Aufgaben
for aufgabe in range(5):
    # Aufgabe mit Ergebnis
    a = random.randint(10,30)
    b = random.randint(10,30)
    c = a + b

    # Tabellenzeile
    print("<tr>")
    print("<td>&nbsp;" + str(aufgabe+1)
          + ".&nbsp;</td>")
    print("<td>&nbsp;" + str(a) + "&nbsp;</td>")
    print("<td>&nbsp;&nbsp;</td>")
    print("<td>&nbsp;" + str(b) + "&nbsp;</td>")
    print("<td>&nbsp;&nbsp;</td>")

```

```

print("<td><input name='ein" + str(aufgabe)
      + "' size='12'></td>")
print("</tr>")

# Richtiges Ergebnis senden
print("<input name='erg" + str(aufgabe) + "'"
      + "type='hidden' value='" + str(c) + "'>")

# Tabellenende
print("</table>")

# Absenden
print("<p><input type='submit' value='Fertig'></p>")

# HTML-Dokumentende
print("</form>")
print("</body>")
print("</html>")

```

**Listing 9.22** Datei spiel\_server\_a.cgi

Zur Erläuterung:

- Python-Interpreter**
  - ▶ Dem Webserver wird mitgeteilt, wo er den Python-Interpreter findet. Nach Einbindung einiger Module wird der Zufallsgenerator initialisiert.
  - ▶ Eingaben aus dem ersten Formular werden in die Python-Variable `form` übernommen.
- HTML-Dokument**
  - ▶ Nach dem HTML-Header beginnt das eigentliche HTML-Dokument.
- Hyperlink**
  - ▶ Falls im ersten Formular kein Spielername eingegeben wurde, wird nur noch ein Hyperlink angeboten, der zurück zum ersten Formular führt. Ohne Spielername geht es nicht weiter, das HTML-Dokument und das Python-CGI-Skript enden.
- Versteckte Felder**
  - ▶ Der übermittelte Spielername wird angezeigt und für die spätere Auswertung versteckt (`type='hidden'`) im Formular eingetragen.
  - ▶ Die Startzeit wird aufgezeichnet und für die spätere Auswertung ebenfalls versteckt im Formular eingetragen.
- HTML-Tabelle**
  - ▶ Es wird eine HTML-Tabelle aufgebaut. Darin werden in einer `for`-Schleife die fünf Aufgaben und die fünf Eingabefelder für die Lösungsversuche eingetragen. Der Name der fünf Eingabefelder wird dynamisch zusammengesetzt: `ein0` bis `ein4`.
  - ▶ Parallel dazu werden die fünf richtigen Lösungen für die spätere Auswertung versteckt im Formular eingetragen. Die Felder erhalten die Namen `erg0` bis `erg4`.
  - ▶ Es folgen das Ende der Tabelle, der Button zum Absenden und das Ende des HTML-Dokuments.

## Auswertungsseite

Die Auswertungsseite enthält ein Hauptprogramm und vier Funktionen:

**Modularer Aufbau**

- ▶ Funktion `hs_leSEN()`: Lesen der Highscores aus der Datei in eine Liste
- ▶ Funktion `hs_schreiben()`: Schreiben der Highscore-Liste in die Datei
- ▶ Funktion `hs_anzeigen()`: Anzeigen der Highscore-Liste im Browser
- ▶ Funktion `hs_hinzu()`: ermittelte Zeit in Highscore-Liste einfügen

Die ersten beiden Funktionen stimmen mit den entsprechenden Funktionen der letzten Spielversion überein. Das Lesen und Schreiben der Datei haben sich nicht geändert, daher wird der Code hier nicht eigens abgebildet.

Das Anzeigen der Highscore-Liste wurde an die Ausgabe im Browser angepasst. Statt einer formatierten Aufgabe auf dem Bildschirm wird eine HTML-Tabelle erzeugt. Die Funktion `hs_hinzu()` ist ein Teil der Funktion `spiel()` aus einer anderen Spielversion. Sie erkennen daran die leichte Wiederverwendbarkeit von modular aufgebauten Programmen. Der Code lautet:

**HTML-Tabelle**

```
#!C:\Python34\python.exe
# Module
import cgi, cgitb, time, glob, pickle
# Ausgabe bei Fehler
cgitb.enable()

# Funktion Highscore lesen
def hs_leSEN(): [ .....

```

```

# Funktion Highscore hinzufügen
def hs_hinzufügen(name, differenz):
    # Mitten in Liste schreiben
    gefunden = False
    for i in range(len(hs_liste)):
        # Einsetzen in Liste
        if differenz < hs_liste[i][1]:
            hs_liste.insert(i, [name, differenz])
            gefunden = True
            break

    # Ans Ende der Liste schreiben
    if not gefunden:
        hs_liste.append([name, differenz])

# Hauptprogramm

# Objekt der Klasse FieldStorage
form = cgi.FieldStorage()

# Zeit berechnen
endzeit = time.time()
differenz = endzeit - float(form["startzeit"].value)

# Header
print("Content-type: text/html")
print()

# HTML-Dokumentbeginn
print("<html>")
print("<body>")
print("<p><b>Kopfrechnen</b></p>")
# Spielername
print("<p>Hallo <b>", form["spielername"].value,
      "</b>, Ihr Ergebnis</p>")

# Auswertung
richtig = 0
for aufgabe in range(5):
    if "ein" + str(aufgabe) in form:
        if form["ein" + str(aufgabe)].value == \
           form["erg" + str(aufgabe)].value:
            richtig += 1

# Ausgabe
print("<p>{0:d} von 5 in {1:.2f} Sekunden". \
      format(richtig, differenz), end = "")

```

```

# Highscore
if richtig == 5:
    print(", Highscore</p>")
    hs_leSEN()
    hs_hinzu(form["spielername"].value, differenz)
    hs_schreiben()
    hs_anzeigen()
else:
    print(", leider kein Highscore</p>")

# Hyperlink zum Anfang
print("<p><a href='http://localhost/Python34/'"
      "'spiel_server.htm'>Zum Start</a></p>")

# HTML-Dokumentende
print("</body>")
print("</html>")

```

**Listing 9.23** Datei spiel\_server\_b.cgi

Zur Erläuterung:

- ▶ Die Funktion `hs_anzeigen()` enthält eine HTML-Tabelle mit Überschrift und einzelnen Zeilen für die Highscore-Daten. Die Zahlen werden für die Anzeige mithilfe der Funktionen `str()` und `format()` in Zeichenketten umgewandelt.
- ▶ An die Funktion `hs_hinzu()` werden Name und ermittelte Zeit des Spielers übergeben. Diese beiden Werte werden an passender Stelle in die Highscore-Liste eingefügt oder hinten angehängt.
- ▶ Im Hauptprogramm wird die Endzeit ermittelt. Aus der übermittelten Startzeit und der Endzeit wird die Ergebniszeit errechnet.
- ▶ Die fünf Lösungen werden mit den fünf Eingaben verglichen, und die Anzahl der gelösten Aufgaben wird berechnet. Wurden alle Aufgaben richtig gelöst, so werden
  - die Highscores aus der Datei eingelesen,
  - der neue Highscore mit Name und Ergebniszeit hinzugefügt,
  - die Highscore-Liste wieder in die Datei geschrieben und
  - die Highscores angezeigt.
- ▶ Schließlich wird der Hyperlink zum Start des Programms angegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt. Die erste Zeile lautet `#!C:\Python27\python.exe`. Das Ziel des HTML-Formulars ist `/cgi-bin/Python27/spiel_server_a.cgi`. Zurück zum Start des Spiels gelangen Sie wieder über die Adresse `http://localhost/Python27/spiel_server.htm`.

# Kapitel 10

## Datenbanken

Eine Datenbank dient zur Speicherung größerer Datenmengen und zur übersichtlichen Darstellung bestimmter Daten aus diesen Datenmengen.

In diesem Buch arbeiten wir mit zwei verschiedenen SQL-basierten Datenbanksystemen:

- ▶ SQLite (auf Basis von Textdateien, für kleine Datenmengen) und SQLite
- ▶ MySQL (komplexes Datenbanksystem, für große Datenmengen) MySQL

Bei SQL (*Structured Query Language*) handelt es sich um die meist verwendete Datenbanksprache.

### 10.1 Aufbau von Datenbanken

Innerhalb einer Datenbank befinden sich verschiedene Tabellen. Im Folgenden sehen Sie eine einfache Beispieldatenebene.

Name	Vorname	Personalnummer	Gehalt	Geburtstag
Maier	Hans	6714	3.500,00	15.03.62
Schmitz	Peter	81343	3.750,00	12.04.58
Mertens	Julia	2297	3.621,50	30.12.59

Tabelle 10.1 Beispiel mit Personaldaten

Die Begriffe in der ersten Zeile bezeichnet man als die *Felder* der Tabelle. Es folgen die einzelnen *Datensätze* der Tabelle – in diesem Fall sind es drei.

**Feld, Datensatz**

Natürlich legt niemand für drei Datensätze eine Datenbank mit einer Tabelle an, aber die vorliegende Struktur könnte auch für mehrere Tausend Datensätze verwendet werden. Die Felder haben jeweils einen bestimmten Datentyp. In diesem Fall gibt es die Datentypen *Text*, *Ganze Zahl*, *Zahl mit Nachkommastellen* und *Datumsangabe*.

**Struktur, Datentyp**

Eine Datenbank wird in den folgenden Schritten erzeugt:

**Reihenfolge**

- ▶ Anlegen der Datenbank
- ▶ Anlegen von Tabellen durch Angabe der Struktur
- ▶ Eingeben der Datensätze in die Tabellen

- Strukturänderung** Die Struktur einer existierenden Datenbank oder Tabelle kann auch dann noch verändert werden, wenn die Datenbank bereits Daten enthält. Allerdings sollten Sie die Struktur Ihrer Datenbank stets gründlich planen, da es bei einer nachträglichen Veränderung leicht zu Datenverlusten kommt.
- SQL** Sie werden mit SQL-Anweisungen arbeiten. Diese dienen
- ▶ zur Erzeugung der Struktur von Datenbanken und Tabellen und
  - ▶ zum Bearbeiten der Datensätze (Erzeugen, Anzeigen, Ändern, Löschen).

## 10.2 SQLite

- Modul sqlite3** SQLite ist ein einfacher Ersatz für ein komplexes Datenbankmanagement-System. Es wird über das Modul `sqlite3` direkt in Python eingebunden, arbeitet auf der Basis von Textdateien und kann bei kleineren Datenmengen eingesetzt werden. Es erfordert keine zusätzlichen Installationen.
- Datentypen** SQLite bietet standardmäßig die folgenden Datentypen:
- ▶ `TEXT`, für Zeichenketten
  - ▶ `INTEGER`, für ganze Zahlen
  - ▶ `FLOAT`, für Zahlen mit Nachkommastellen
  - ▶ `BLOB`, für *binary large objects*, also große binäre Datenmengen
  - ▶ `NULL`, entspricht `None` in Python

Für die Kontrolle und die richtige Konvertierung anderer Datentypen von SQLite nach Python, z. B. einer SQLite-Zeichenkette in eine Python-Variable für eine Datumsangabe, muss der Entwickler selbst sorgen.

### Hinweis

Die oben genannten Datentypen und andere SQL-spezifische Angaben werden in Großbuchstaben geschrieben. Dies ist für SQL nicht notwendig, dient aber zur deutlicheren Darstellung.

### 10.2.1 Datenbank, Tabelle und Datensätze

- Erzeugen** Im folgenden Programm wird zunächst eine SQLite-Datenbank erzeugt und eine Tabelle mit einem eindeutigen Index angelegt. Anschließend werden drei Datensätze in der Tabelle angelegt. Als Beispiel dient Tabelle 10.1 am Anfang dieses Kapitels.
- Eindeutiger Index** Der eindeutige Index dient dazu, Datensätze zu erzeugen, die jeweils ein unverwechselbares Merkmal haben und somit eindeutig identifiziert werden

können. In der Personentabelle ist dazu das Feld `personalnummer` geeignet. Das Programm lautet:

```
import os, sys, sqlite3

# Existenz feststellen
if os.path.exists("firma.db"):
    print("Datei bereits vorhanden")
    sys.exit(0)

# Verbindung zur Datenbank erzeugen
connection = sqlite3.connect("firma.db")

# Datensatzcursor erzeugen
cursor = connection.cursor()

# Tabelle erzeugen
sql = "CREATE TABLE personen(\" \
        \"name TEXT, \" \
        \"vorname TEXT, \" \
        \"personalnummer INTEGER PRIMARY KEY, \" \
        \"gehalt FLOAT, \" \
        \"geburtstag TEXT)"
cursor.execute(sql)

# Datensatz erzeugen
sql = "INSERT INTO personen VALUES('Maier', \" \
        '\"Hans', 6714, 3500, '15.03.1962')"
cursor.execute(sql)
connection.commit()

# Datensatz erzeugen
sql = "INSERT INTO personen VALUES('Schmitz', \" \
        '\"Peter', 81343, 3750, '12.04.1958')"
cursor.execute(sql)
connection.commit()

# Datensatz erzeugen
sql = "INSERT INTO personen VALUES('Mertens', \" \
        '\"Julia', 2297, 3621.5, '30.12.1959')"
cursor.execute(sql)
connection.commit()

# Verbindung beenden
connection.close()
```

**Listing 10.1** Datei `sqlite_erzeugen.py`

Zur Erläuterung:

- ▶ Zunächst wird geprüft, ob die Datenbankdatei *firma.db* bereits existiert. Ist dies der Fall, dann wurde das Programm bereits einmal aufgerufen, die Tabelle `personen` existiert bereits und ist mit Daten gefüllt. Ein zweiter Aufruf würde zu einem Fehler führen.
- connect()** ▶ Dann wird mithilfe der Funktion `connect()` aus dem Modul `sqlite3` eine Verbindung zur Datenbank hergestellt. Da die Datenbankdatei noch nicht existiert, wird sie nun erzeugt.
- Connection-Objekt** ▶ Der Rückgabewert der Funktion `connect()` ist ein Objekt der Klasse `sqlite3.connection`. Über diese Verbindung kann auf die Datenbank zugegriffen werden.
- cursor()** ▶ Mithilfe der Methode `cursor()` des Connection-Objekts wird ein Cursor-Objekt erzeugt. Über diesen Cursor können SQL-Abfragen an die Datenbank gesendet und die Ergebnisse empfangen werden.
- Cursor-Objekt** ▶ Der Rückgabewert der Methode `cursor()` ist ein Objekt der Klasse `sqlite3.cursor`. Über diesen Cursor können SQL-Abfragen an die Datenbank gesendet und die Ergebnisse empfangen werden.
- execute()** ▶ Es wird eine Zeichenkette zusammengesetzt, die einen SQL-Befehl enthält. Dieser SQL-Befehl wird anschließend mit der Methode `execute()` des Cursor-Objekts an die Datenbank gesendet.
- CREATE TABLE** ▶ Die SQL-Anweisung `CREATE TABLE` erzeugt eine Tabelle in einer Datenbank. Hinter dem Namen der Tabelle (hier: `personen`) werden in Klammern die einzelnen Felder der Tabelle jeweils mit Datentyp angegeben.
- Datentypen** ▶ Die Felder `name`, `vorname` und `geburtstag` sind vom Typ `TEXT`. Das Feld `gehalt` ist vom Typ `FLOAT`, das Feld `personalnummer` vom Typ `INTEGER`.
- Eindeutiger Index** ▶ Auf dem Feld `personalnummer` wird mithilfe der Anweisung `PRIMARY KEY` ein Primärschlüssel definiert. Dabei handelt es sich um einen eindeutigen Index, der dafür sorgt, dass es keine zwei Datensätze geben kann, die dieselbe Personalnummer besitzen.
- INSERT INTO** ▶ Mithilfe der SQL-Anweisung `INSERT INTO` werden Datensätze angelegt. Es handelt sich um eine sogenannte Aktionsabfrage, die zu einer Änderung in der Tabelle führt.
- commit()** ▶ Auf eine Aktionsabfrage sollte immer ein Aufruf der Methode `commit()` des Connection-Objekts folgen, damit die Änderungen unmittelbar ausgeführt werden.
- Felder, Werte** ▶ In der SQL-Anweisung `INSERT INTO` werden nach dem Namen der Tabelle und dem Schlüsselwort `VALUES` die einzelnen Werte des Datensatzes für die Felder der Tabelle in Klammern angegeben. Dabei ist es wichtig, Zeichenketten in einfache Anführungsstriche zu setzen. Außerdem müssen Sie die Reihenfolge der Felder wie bei der Erzeugung der Tabelle einhalten.

- Zuletzt wird die Verbindung zur Datenbank mithilfe der Methode `close()` wieder geschlossen.

### 10.2.2 Daten anzeigen

Es sollen alle Datensätze der Tabelle mit allen Inhalten angezeigt werden. Das Programm:

```
import sqlite3

# Verbindung, Cursor
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()

# SQL-Abfrage
sql = "SELECT * FROM personen"

# Kontrollausgabe der SQL-Abfrage
# print(sql)

# Absenden der SQL-Abfrage
# Empfang des Ergebnisses
cursor.execute(sql)

# Ausgabe des Ergebnisses
for dsatz in cursor:
    print(dsatz[0], dsatz[1], dsatz[2],
          dsatz[3], dsatz[4])

# Verbindung beenden
connection.close()
```

**Listing 10.2** Datei `sqlite_anzeigen.py`

Das Programm erzeugt die Ausgabe:

Mertens Julia 2297 3621.5 30.12.1959  
 Maier Hans 6714 3500.0 15.03.1962  
 Schmitz Peter 81343 3750.0 12.04.1958

Zur Erläuterung:

- Nach Erzeugung von Datenbankverbindung und Datensatz-Cursor wird mithilfe der Methode `execute()` eine SQL-Abfrage gesendet.
- Die Abfrage `SELECT FROM` ist eine sogenannte Auswahlabfrage. Solche Abfragen dienen nur zum Sichten, nicht zum Ändern der Datenbankinhalte.
- Die Anweisung `SELECT * FROM personen` liefert alle Felder und alle Datensätze der Tabelle. Der Stern (\*) steht für »alle Felder«.

- Kontrollausgabe**
- Die SQL-Anweisung wurde zunächst in einer Zeichenkette gespeichert. Dies ist nicht zwingend notwendig, ermöglicht aber eine Ausgabe der Zeichenkette zur Kontrolle mit `print(sql)`. Dies ist besonders vorteilhaft für die Fehlersuche bei komplexeren SQL-Befehlen.
- Ergebnis der Abfrage**
- Nach Ausführung der Methode `execute()` steht im Cursor-Objekt das Ergebnis der Abfrage zur Verfügung. Dabei handelt es sich um die Datensätze, die zur Abfrage passen. Diese Reihe kann mithilfe einer Schleife durchlaufen werden.
- Tupel mit Datensatz**
- Jeder Ergebnis-Datensatz steht in einem Tupel, das aus den Werten für die einzelnen Felder in der gleichen Reihenfolge wie bei der Erzeugung der Tabelle besteht.
  - Da es sich bei `select` nicht um eine Aktionsabfrage handelt (die eine Änderung in der Datenbank vornimmt), sondern nur um eine Auswahlabfrage (die Informationen ermittelt), muss hier die Methode `commit()` nicht aufgerufen werden.
  - Zuletzt wird die Verbindung zur Datenbank wieder beendet.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen `\` für den Umbruch von langen Programmzeilen eingesetzt.

### 10.2.3 Daten auswählen, Operatoren

- SELECT WHERE** Mithilfe von `SELECT` in Verbindung mit der `WHERE`-Klausel wählen Sie einzelne oder mehrere Datensätze aus. Einige Beispiele:

```
import sqlite3

# Verbindung, Cursor
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()

# SQL-Abfragen
# Einzelne Felder
sql = "SELECT name, vorname FROM personen"
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1])
print()

# Auswahl mit WHERE-Klausel und Vergleichsoperator
sql = "SELECT * FROM personen " \
      "WHERE gehalt > 3600"
cursor.execute(sql)
```

```

for dsatz in cursor:
    print(dsatz[0], dsatz[3])
print()

# Auswahl mit Zeichenkette
sql = "SELECT * FROM personen " \
      "WHERE name = 'Schmitz'"
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1])
print()

# Auswahl mit logischen Operatoren
sql = "SELECT * FROM personen " \
      "WHERE gehalt >= 3600 AND gehalt <= 3650"
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[3])

# Verbindung beenden
connection.close()

```

**Listing 10.3** Datei `sqlite_auswaehlen.py`

Erzeugt wird die Ausgabe:

Mertens Julia  
 Maier Hans  
 Schmitz Peter

**Mertens 3621.5**  
**Schmitz 3750.0**

**Schmitz Peter**

**Mertens 3621.5**

Zur Erläuterung:

- ▶ Es handelt sich um vier verschiedene SQL-Abfragen, die nacheinander ausgeführt werden.
- ▶ Wenn Sie nur die Werte einzelner Felder sehen möchten, notieren Sie die Namen dieser Felder zwischen `SELECT` und `FROM`. Die Anweisung `SELECT name, vorname FROM personen` liefert nur die Inhalte der beiden genannten Felder aller Datensätze der Tabelle.
- ▶ Mithilfe von `WHERE` können Sie die Auswahl der Datensätze einschränken, indem Sie eine Bedingung mit Vergleichsoperatoren erstellen (wie bei den Anweisungen `if` oder `while` in Python).

Felder auswählen

Datensätze auswählen

- |                             |   |
|-----------------------------|---|
| <b>Vergleichsoperatoren</b> | ► Die Anweisung <code>SELECT * FROM personen WHERE gehalt &gt; 3600</code> liefert die Inhalte aller Felder, aber (mithilfe eines Vergleichsoperators) nur die Datensätze, bei denen der Wert des Feldes <code>gehalt</code> oberhalb der genannten Grenze liegt (siehe auch <a href="#">Tabelle 10.2</a> ).  |
| <b>Anführungsstriche</b>    | ► Auch bei Zeichenketten kann ein Vergleich stattfinden. Dabei ist allerdings auf die <i>einfachen</i> Anführungsstriche zu achten. Die SQL-Anweisung <code>SELECT * FROM personen WHERE name = 'Schmitz'</code> liefert nur die Datensätze, bei denen der Name gleich »Schmitz« ist.   |
| <b>Logische Operatoren</b>  | ► Logische Operatoren ermöglichen, wie in Python, die Verknüpfung mehrerer Bedingungen. Die Anweisung <code>SELECT * FROM personen WHERE gehalt &gt;= 3600 AND gehalt &lt;= 3650</code> liefert nur die Datensätze, bei denen der Wert des Feldes <code>gehalt</code> innerhalb der genannten Grenzen liegt (siehe auch <a href="#">Tabelle 10.3</a> ). |

Operator	Erläuterung	Operator	Erläuterung
=	gleich	>=	größer als oder gleich
<>	ungleich	<	kleiner als
>	größer als	<=	kleiner als oder gleich

**Tabelle 10.2** Vergleichsoperatoren in SQL

Operator	Erläuterung
NOT	Logisches NICHT: Der Wahrheitswert einer Bedingung wird umgekehrt.
AND	Logisches UND: Beide Bedingungen müssen zutreffen.
OR	Logisches ODER: Nur eine der Bedingungen muss zutreffen.

**Tabelle 10.3** Logische Operatoren in SQL

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 10.2.4 Operator »LIKE«

- |                    |  |
|--------------------|--|
| <b>Platzhalter</b> | Der Operator <code>LIKE</code> dient zur Auswahl von Datensätzen über Zeichenketten, in denen Platzhalter vorkommen dürfen. Sie können also Abfragen bilden wie: |
|--------------------|--|

- ▶ Suche alle Datensätze, die mit ... beginnen.
- ▶ Suche alle Datensätze, die mit ... enden.
- ▶ Suche alle Datensätze, die ... enthalten.

Einige Beispiele:

```
import sqlite3

# Verbindung, Cursor
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()

# SQL-Abfragen

# Beliebig viele beliebige Zeichen
sql = "SELECT * FROM personen WHERE name LIKE 'm%'"
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1])
print()

# Beinhaltet ...
sql = "SELECT * FROM personen WHERE name LIKE '%i'"
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1])
print()

# Einzelne beliebige Zeichen
sql = "SELECT * FROM personen WHERE name LIKE 'M_\_er'"
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1])

# Verbindung beenden
connection.close()
```

#### **Listing 10.4 Datei sqlite\_like.py**

Es wird die Ausgabe erzeugt:

**Mertens Julia**

**Maier Hans**

**Maier Hans**

**Schmitz Peter**

**Maier Hans**

Zur Erläuterung:

- Platzhalter %**
- ▶ Der Platzhalter % (Prozentzeichen) steht für eine unbestimmte Anzahl beliebiger Zeichen. Die Anweisung `SELECT * FROM personen WHERE name LIKE '%m'` liefert nur die Datensätze, bei denen der Name mit dem Buchstaben »m« beginnt (unabhängig von Groß- und Kleinschreibung).
  - ▶ Steht das Prozentzeichen vor *und* nach einem bestimmten Zeichen, so werden alle Datensätze gesucht, in denen dieses Zeichen vorkommt. Die Anweisung `SELECT * FROM personen WHERE name LIKE '%i%` liefert nur die Datensätze, in denen der Buchstabe »i« (oder »I«) vorkommt.
- Platzhalter \_**
- ▶ Der Platzhalter \_ (Unterstrich) steht für ein einzelnes, beliebiges Zeichen. Die Anweisung `SELECT * FROM personen WHERE name LIKE 'M_ er'` liefert nur die Datensätze, bei denen der Name mit »M« beginnt und nach zwei weiteren Zeichen mit »er« endet. Dies trifft z. B. auf »Maier«, »Mayer« oder »Meyer« zu.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### 10.2.5 Sortierung der Ausgabe

- ORDER BY** Mithilfe von ORDER BY werden die Datensätze in sortierter Form geliefert. Es kann nach einem oder mehreren Feldern, aufsteigend oder absteigend sortiert werden. Ein Beispiel:

```
import sqlite3

# Verbindung, Cursor
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()
# Sortierung fallend
sql = "SELECT * FROM personen ORDER BY gehalt DESC"
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1], dsatz[3])
print()

# Sortierung nach mehreren Feldern
sql = "SELECT * FROM personen ORDER BY name, vorname"
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1])

connection.close()
```

**Listing 10.5** Datei `sqlite_sortieren.py`

Die Ausgabe lautet:

**Maier Wolfgang** 3810.0  
**Schmitz Peter** 3750.0  
**Mertens Julia** 3621.5  
**Maier Hans** 3500.0

**Maier Hans**  
**Maier Wolfgang**  
**Mertens Julia**  
**Schmitz Peter**

Zur Erläuterung:

- ▶ Zur deutlicheren Darstellung wurde vor Ausführung dieses Programms noch ein weiterer Datensatz hinzugefügt.
- ▶ Die Anweisung `SELECT * FROM personen ORDER BY gehalt DESC` liefert alle Datensätze, absteigend nach Gehalt sortiert. Der Zusatz `DESC` steht für *descending* (absteigend). Für eine aufsteigende Sortierung könnten Sie den Zusatz `ASC` (*ascending*, deutsch: aufsteigend) verwenden. Dies ist allerdings der Standard, daher kann der Zusatz entfallen.
- ▶ Die Anweisung `SELECT * FROM personen ORDER BY name, vorname` liefert alle Datensätze, aufsteigend nach Name sortiert. Bei gleichem Namen wird nach Vorname sortiert.

`DESC, ASC`

**Mehrere Felder**

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 10.2.6 Auswahl nach Eingabe

Der Benutzer kann Datensätze in eingeschränkter Form auch selbst auswählen. Mit dem folgenden Programm werden nur die Datensätze angezeigt,

**Eingabe des Benutzers**

- ▶ die dem eingegebenen Namen entsprechen oder
- ▶ die den eingegebenen Namensteil enthalten.

```
import sqlite3
```

```
# Verbindung, Cursor
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()
# Eingabe Name
eingabe = input("Bitte den gesuchten Namen eingeben: ")
sql = "SELECT * FROM personen WHERE name LIKE '" \
      + eingabe + "'"
print(sql)
```

```

cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1])
print()

# Eingabe Teil des Namens
eingabe = input("Bitte den gesuchten Namensteil eingeben: ")
sql = "SELECT * FROM personen WHERE name LIKE '%" \
    + eingabe + "%'"

print(sql)
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1])
print()

connection.close()

```

**Listing 10.6** Datei sqlite\_eingabe.py

Die Ausgabe (mit Eingabe des Benutzers) sieht wie folgt aus:

```

Bitte den gesuchten Namen eingeben: Maier
SELECT * FROM personen WHERE name LIKE 'Maier'
Maier Hans
Maier Wolfgang

```

```

Bitte den gesuchten Namensteil eingeben: r
SELECT * FROM personen WHERE name LIKE '%r%'
Mertens Julia
Maier Hans
Maier Wolfgang

```

Zur Erläuterung:

- ▶ Beim ersten Mal hat der Benutzer den Namen »Maier« eingegeben. Es wird ein SQL-Befehl zusammengesetzt, der diese Benutzereingabe enthält. Sodann werden alle Datensätze mit dem Namen »Maier« geliefert.
- ▶ Gerade bei zusammengesetzten SQL-Anweisungen lohnt sich eine Ausgabe des SQL-Befehls während der Entwicklungszeit. So können Sie beispielsweise kontrollieren, ob die einfachen Anführungsstriche gesetzt wurden.
- ▶ Beim zweiten Mal hat der Benutzer nur das Zeichen »r« eingegeben. Diese Eingabe wird in Prozentzeichen (und einfache Anführungsstriche) gesetzt. Es werden alle Datensätze geliefert, deren Name ein »r« enthält.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

### 10.2.7 Datensätze ändern

Die Anweisung `UPDATE` dient zur Änderung von Feldinhalten in einem oder mehreren Datensätzen. Sie ähnelt in ihrem Aufbau der Anweisung `SELECT`. Wählen Sie Ihre Auswahlkriterien sorgfältig, um nicht versehentlich andere als die gemeinten Datensätze in die Änderung einzubeziehen.

`UPDATE`

```
SQLCommand = "UPDATE personen SET gehalt = 3800"
```

Diese Anweisung würde bei sämtlichen Datensätzen der Tabelle `personen` den Wert für das Feld `gehalt` auf den Wert 3800 setzen – diese Aktion ist sicherlich nicht beabsichtigt.

Nehmen Sie daher eine Einschränkung vor, am besten über das Feld mit dem eindeutigen Index (hier das Feld `personalnummer`).

`Eindeutiger Index`

```
SQLCommand = "UPDATE personen SET gehalt = 3780 " \
"WHERE personalnummer = 81343"
```

Das Beispiel mit einer Ausgabe vor der Änderung und einer Ausgabe nach der Änderung sieht wie folgt aus:

`Vorher, nachher`

```
import sqlite3

def ausgabe():
    # SQL-Abfrage, senden, Ausgabe
    sql = "SELECT * FROM personen"
    cursor.execute(sql)
    for dsatz in cursor:
        print(dsatz[0], dsatz[1], dsatz[2], dsatz[3])
    print()

# Verbindung, Cursor
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()

# Vorher
ausgabe()

# Datensatz aktualisieren
sql = "UPDATE personen SET gehalt = 3780 " \
      "WHERE personalnummer = 81343"
cursor.execute(sql)
connection.commit()

# Nachher
ausgabe()

connection.close()
```

**Listing 10.7** Datei `sqlite_aendern.py`

Folgende Ausgabe wird erzeugt:

```
Mertens Julia 2297 3621.5
Maier Hans 6714 3500.0
Maier Wolfgang 8339 3810.0
Schmitz Peter 81343 3750.0
```

```
Mertens Julia 2297 3621.5
Maier Hans 6714 3500.0
Maier Wolfgang 8339 3810.0
Schmitz Peter 81343 3780.0
```

Zur Erläuterung:

- ▶ Wie Sie sehen, wurde nur ein Datensatz verändert. Aufbau und Auswirkung des SQL-Befehls wurden bereits erläutert.

**Mehrere Felder ändern** Sollen mehrere Feldinhalte eines Datensatzes gleichzeitig geändert werden, könnte dies wie folgt aussehen:

```
sql = "UPDATE personen SET gehalt = 3780, " \
      "vorname = 'Hans-Peter' " \
      "WHERE personalnummer = 81343"
```

**Unerlaubte Änderung** Zu den Änderungen, die nicht durchgeführt werden dürfen, gehört der Eintrag eines Datensatzes mit einem bereits vorhandenen Wert für das Feld mit dem eindeutigen Index. In der vorliegenden Tabelle kann also kein neuer Datensatz eingetragen werden, der im Feld `personalnummer` einen Wert hat, der bereits in einem anderen Datensatz vorhanden ist.

Das Gleiche gilt für die entsprechende Änderung eines Datensatzes. Der Wert für das Feld `personalnummer` eines Datensatzes darf also nur einmal vorkommen. So würde die Anweisung ...

```
sql = "UPDATE personen SET personalnummer = 6714 " \
      "WHERE personalnummer = 81343"
```

... zur folgenden Fehlermeldung führen:

**sqlite3.IntegrityError: PRIMARY KEY must be unique**

**unique** Dies liegt daran, dass für das Feld `personalnummer` mit dem Zusatz `PRIMARY KEY` ein eindeutiger Index definiert wurde. Daher ist jede Personalnummer einzigartig (englisch: *unique*). Die Personalnummer 6714 gehört aber bereits zu einem anderen Datensatz.

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 10.2.8 Datensätze löschen

Mit der SQL-Anweisung `DELETE` löschen Sie Datensätze. Achten Sie auch hier darauf, welche Datensätze Sie genau löschen möchten, damit Sie nicht unabsichtigt alle Datensätze auf einmal löschen. Ein Beispiel:

```
import sqlite3

def ausgabe():
    # SQL-Abfrage, senden, Ausgabe
    sql = "SELECT * FROM personen"
    cursor.execute(sql)
    for dsatz in cursor:
        print(dsatz[0], dsatz[1], dsatz[2], dsatz[3])
    print()

# Verbindung, Cursor
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()

# Vorher
ausgabe()

# Datensatz entfernen
sql = "DELETE FROM personen " \
      "WHERE personalnummer = 8339"
cursor.execute(sql)
connection.commit()

# Nachher
ausgabe()

connection.close()
```

**Listing 10.8** Datei `sqlite_loeschen.py`

Die Ausgabe lautet:

Mertens Julia 2297 3621.5  
 Maier Hans 6714 3500.0  
 Maier Wolfgang 8339 3810.0  
 Schmitz Peter 81343 3780.0

Mertens Julia 2297 3621.5  
 Maier Hans 6714 3500.0  
 Schmitz Peter 81343 3780.0

Zur Erläuterung:

- Der Datensatz mit der Personalnummer 8339 wurde gelöscht, wie der Vergleich der Ausgabe vor dem Löschen mit der Ausgabe nach dem Löschen zeigt.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 10.3 SQLite auf dem Webserver

Sie können eine SQLite-Datenbank natürlich auch auf einem Webserver zur dauerhaften Speicherung von Daten einsetzen.

**Python-CGI-Skript** In diesem Abschnitt machen wir eine SQLite-Datenbank über ein Python-CGI-Interface zugänglich. Hierzu werden Sie Ihre Kenntnisse aus [Abschnitt 9.2, »Webserver-Programmierung«, und Abschnitt 10.2, »SQLite«](#), nutzen.

**Datenbankdatei schützen** Da es sich bei einer SQLite-Datenbank um eine einfache Textdatei handelt, sollte diese in einem eigenen, per Passwort geschützten Verzeichnis liegen. Damit verhindern Sie, dass die Datei einfach komplett heruntergeladen werden kann. Sie soll nur über das bereitgestellte Python-CGI-Interface genutzt werden. Das Programm:

```
#!C:\Python34\python.exe
import sqlite3

# Dokumenttyp
print("Content-type: text/html")
print()

# Dokumentbeginn
print("<html>")
print("<body>")

# Verbindung, Cursor
connection = sqlite3.connect("sqlite/firma.db")
cursor = connection.cursor()

# SQL-Abfrage, Absenden, Ergebnis
sql = "SELECT * FROM personen"
cursor.execute(sql)

# Tabellenausgabe des Ergebnisses
print("<table border='1'>")
```

```

# Tabellenkopf
print("<tr><td>Name</td><td>Vorname</td>""
      "<td align='right'>PNr.</td>""
      "<td align='right'>Gehalt</td>""
      "<td>Geburtstag</td></tr>")

# Tabellenzeile
for dsatz in cursor:
    print("<tr> " \
          + "<td>" + dsatz[0] + "</td>""
          + "<td>" + dsatz[1] + "</td>""
          + "<td align='right'>" + str(dsatz[2]) + "</td>""
          + "<td align='right'>""
          + "{0:.2f}".format(dsatz[3]) + "</td>""
          + "<td>" + dsatz[4] + "</td>""
          + "</tr>")

# Tabellenende
print("</table>")

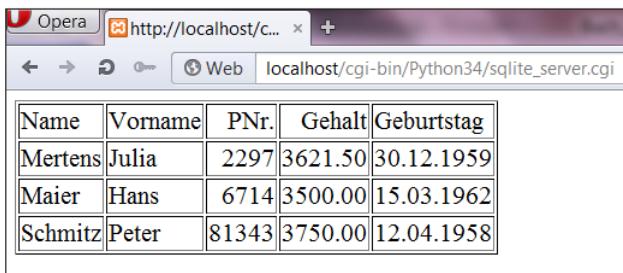
# Verbindung beenden
connection.close()

# Dokumentende
print("</body>")
print("</html>")

```

**Listing 10.9** Datei `sqlite_server.cgi`

Abbildung 10.1 zeigt die Ausgabe im Browser nach Eingabe der Adresse `http://localhost/cgi-bin/Python34/sqlite_server.cgi`.



The screenshot shows a Microsoft Internet Explorer browser window. The address bar displays the URL `http://localhost/cgi-bin/Python34/sqlite_server.cgi`. The main content area of the browser shows a table with five columns: Name, Vorname, PNr., Gehalt, and Geburtstag. The table contains four rows of data:

Name	Vorname	PNr.	Gehalt	Geburtstag
Mertens	Julia	2297	3621.50	30.12.1959
Maier	Hans	6714	3500.00	15.03.1962
Schmitz	Peter	81343	3750.00	12.04.1958

**Abbildung 10.1** Datenbankinhalte auf dem Webserver

Zur Erläuterung:

- Das Python-Programm `sqlite_server.cgi` befindet sich (bei einer Standard-installation von XAMPP) im Festplattenverzeichnis `C:\xampp\cgi-bin\Python34`. XAMPP

- |  |  |
|--|--|
| <b>Datenbankdatei</b>                  | ► Der Inhalt der SQLite-Datenbankdatei <i>firma.db</i> ist aus dem vorherigen Abschnitt bekannt. Die Datei liegt im Verzeichnis: <i>C:\xampp\cgi-bin\Python34\sqlite</i> . Dieses Verzeichnis sollte geschützt werden. Die Standard-Konfigurationsprogramme für eigene Websites bieten unkomplizierte Möglichkeiten zum Schutz von Verzeichnissen.   |
| <b>HTML-Dokument, Datenbankabfrage</b> | ► Das HTML-Dokument wird begonnen. Es wird Verbindung zur Datenbank aufgenommen. Eine Datenbankabfrage wird abgesendet, das Ergebnis steht über den Datensatz-Cursor zur Verfügung.  |
| <b>HTML-Tabelle</b>                    | ► Es wird eine HTML-Tabelle mit Rahmen begonnen.<br>► Es folgt die erste Tabellenzeile mit der Überschrift. Sie enthält die Bezeichnungen der einzelnen Felder.  |
| <b>Datensatz</b>                       | ► Für jeden Datensatz wird eine Tabellenzeile erzeugt. In den einzelnen Zellen der Tabellenzeile stehen die Werte der Datenbankfelder.   |
| <b>Formatierung</b>                    | ► Die Personalnummer und das Gehalt werden rechtsbündig in die Zelle geschrieben. Die Personalnummer wird mithilfe der Funktion <code>str()</code> in eine Zeichenkette umgewandelt. Das Gehalt wird mithilfe der Funktion <code>format()</code> mit zwei Nachkommastellen formatiert.<br>► Die HTML-Tabelle wird beendet, die Verbindung zur Datenbank geschlossen und das HTML-Dokument beendet. |

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt. Die erste Zeile lautet `#!C:\Python27\python.exe`. Das Python-Programm liegt im Festplattenverzeichnis *C:\xampp\cgi-bin\Python27*, die SQLite-Datenbankdatei im Unterverzeichnis *sqlite*. Die einzugebende Adresse im Browser lautet `http://localhost/cgi-bin/Python27/sqlite_server.cgi`.

## 10.4 MySQL

MySQL ist ein komplexes Datenbankmanagement-System. In diesem Abschnitt lernen Sie es mithilfe einiger Programme kennen. Die SQL-Befehle zur Auswahl bzw. Änderung von Daten kennen Sie bereits aus dem Abschnitt zu SQLite. Daher wird in diesem Abschnitt nur eine Datenbank mit einer Tabelle und Datensätzen erzeugt und auf einfache Weise abgefragt.

- |                   |   |
|-------------------|---|
| <b>Datentypen</b> | MySQL bietet eine ganze Reihe von Datentypen, unter anderem:                        |
|                   | ► <code>varchar</code> , für Zeichenketten<br>► <code>int</code> , für ganze Zahlen |

- ▶ `double`, für Zahlen mit Nachkommastellen
- ▶ `date`, für Datumsangaben

### 10.4.1 XAMPP und Connector / Python

Zum Testen der Programme wird ein lokaler MySQL-Datenbankserver benötigt. Diesen finden Sie im Paket XAMPP, das bereits vorkonfiguriert ist und einfach zu installieren ist. Es umfasst neben einem MySQL-Datenbankserver einen Apache-Webserver, die Webserver-Sprache *PHP*, das Datenbankverwaltungsprogramm *phpMyAdmin* und weitere Software.

Sie können XAMPP unter <http://www.apachefriends.org> sowohl für Windows als auch für Ubuntu Linux und für OS X herunterladen. Die jeweils aktuellen Versionen stehen auch auf dem Datenträger zum Buch zur Verfügung. Die Installation von XAMPP wird in Anhang A.1 beschrieben.

Eine Schnittstelle zwischen Python und MySQL bietet der Treiber *Connector / Python*. In diesem Buch werden die Treiber für Python 2.7 und Python 3.3 genutzt. Achten Sie darauf, den passenden Treiber zu verwenden. Der Treiber für Python 3.3 kann nicht zusammen mit Python 3.4 oder Python 3.2 genutzt werden.

**Schnittstelle**

Sie können die MSI-Installationsdatei für Windows mit diesem Treiber von MySQL (<http://dev.mysql.com/downloads/connector/python>) herunterladen. Sie befindet sich aber auch auf dem Datenträger zum Buch. Die Installation verläuft in der Regel problemlos.

**MSI-Datei**

Sie finden auf der Website auch einen Treiber für Ubuntu Linux. Allerdings gestaltet sich die Installation recht aufwendig. Im Rahmen dieses Einsteigerbuchs wird darauf nicht eingegangen. Für OS X gibt es keinen Treiber.

### 10.4.2 Datenbank erzeugen

Nach der Installation und dem Start von XAMPP und der Installation des Treibers *Connector / Python* können Sie mit folgendem Programm zunächst eine Datenbank erzeugen:

```
# Connector importieren
import sys, mysql.connector

# Verbindung zum Datenbankserver erstellen
try:
    connection = mysql.connector.connect \
        (host = "localhost", user = "root", passwd = "")
except:
    print("Keine Verbindung zum Server")
    sys.exit(0)
```

```

# Execution-Objekt erzeugen
cursor = connection.cursor()

# Datenbank erzeugen
cursor.execute("CREATE DATABASE IF NOT EXISTS firma")
connection.commit()

# Execution-Objekt schliessen
cursor.close();

# Verbindung schliessen
connection.close()

```

**Listing 10.10** Datei mysql\_db.py

Zur Erläuterung:

- connect() ▶ Nach dem Import des Moduls `mysql.connector` wird mithilfe der Methode `connect()` eine Verbindung zum Datenbankserver hergestellt. Dabei werden der Name des Servers (hier: `localhost`), der Name des Benutzers (hier: `root`) und das Passwort (hier: kein Passwort) angegeben. Sollte der MySQL-Datenbankserver nicht gestartet worden sein, dann wird die Verbindungsauftnahme fehlschlagen.
- cursor() ▶ Der Rückgabewert der Methode `connect()` ist ein `connection`-Objekt. Über diese Verbindung kann auf die Datenbank zugegriffen werden.
- execute() ▶ Die Methode `cursor()` wird anschließend genutzt, um ein `cursor`-Objekt zu erstellen. Über diesen Cursor können SQL-Abfragen an die Datenbank gesendet und die Ergebnisse empfangen werden.
- commit() ▶ Der Aufruf der Methode `commit()` des `connection`-Objekts sorgt dafür, dass der Befehl innerhalb dieser Aktionsabfrage unmittelbar ausgeführt wird.
- phpMyAdmin ▶ Da keine weiteren Aktionen erfolgen sollen, können `cursor`-Objekt und `connection`-Objekt wieder geschlossen werden, jeweils mithilfe der Methode `close()`.

### 10.4.3 Tabelle anlegen

Als Nächstes wird mithilfe des nachfolgenden Programms eine Tabelle innerhalb der Datenbank angelegt. Das Programm lautet:

```

# Connector importieren
import sys, mysql.connector

# Verbindung zur Datenbank auf dem Datenbankserver erstellen
try:
    connection = mysql.connector.connect(host = "localhost", \
        user = "root", passwd = "", db = "firma")
except:
    print("Keine Verbindung zum Server")
    sys.exit(0)

# Execution-Objekt erzeugen
cursor = connection.cursor()

# Tabelle erzeugen
cursor.execute("CREATE TABLE IF NOT EXISTS personen ("\
    "name varchar(30), vorname varchar(25),"\
    "personalnummer int(11), gehalt double, geburtstag date,"\
    "PRIMARY KEY (personalnummer))")
connection.commit()

# Execution-Objekt schliessen
cursor.close()

# Verbindung schliessen
connection.close()

```

**Listing 10.11** Datei mysql\_tabelle.py

Zur Erläuterung:

- ▶ Wiederum wird die Methode `connect()` genutzt. Als vierter Parameter wird nun zusätzlich der Name der Datenbank angegeben (hier: `firma`). Damit wird nicht nur eine Verbindung zum Datenbankserver hergestellt, sondern auch zu der Datenbank auf dem Server.
- ▶ Mithilfe der Methode `execute()` wird die Tabelle `personen` erzeugt, mit fünf Spalten. Die Felder `name` und `vorname` sind vom Typ `varchar`. Es wird eine Länge von maximal 30 bzw. 25 Zeichen gewählt. Das Feld `personalnummer` ist vom Typ `int`. Die Angabe 11 ist dabei der Standardwert. Das Feld `gehalt` ist vom Typ `double`, für Zahlen mit Nachkommastellen. Das Feld `geburtstag` ist vom Typ `date`, für Datumsangaben.
- ▶ Auf dem Feld `personalnummer` wird ein Primärschlüssel definiert. Damit wird dafür gesorgt, dass es keine zwei Einträge geben kann, die dieselbe Personalnummer haben.
- ▶ Der Zusatz `if not exists` bewirkt auch hier, dass die Tabelle nur erzeugt wird, falls sie noch nicht existiert.
- ▶ Das Programm hat keine Ausgabe. Im Programm *phpMyAdmin* können Sie sich von der Existenz der neuen Tabelle überzeugen.

Felder und Typen

Primärschlüssel

#### 10.4.4 Datensätze anlegen

In diesem Abschnitt werden drei Datensätze in der Tabelle angelegt.

Das Programm lautet:

```
# Connector importieren
import sys, mysql.connector

# Verbindung zur Datenbank auf dem Datenbankserver erstellen
connection = mysql.connector.connect(host = "localhost", \
    user = "root", passwd = "", db = "firma")

# Execution-Objekt erzeugen
cursor = connection.cursor()

# Datensatz erzeugen
sql = "INSERT INTO personen VALUES('Maier', " \
    "'Hans', 6714, 3500, '1962-03-15')"
cursor.execute(sql)
connection.commit()

# Datensatz erzeugen
sql = "INSERT INTO personen VALUES('Schmitz', " \
    "'Peter', 81343, 3750, '1958-04-12')"
cursor.execute(sql)
connection.commit()

# Datensatz erzeugen
sql = "INSERT INTO personen VALUES('Mertens', " \
    "'Julia', 2297, 3621.5, '1959-12-30')"
cursor.execute(sql)
connection.commit()

# Execution-Objekt schliessen
cursor.close()

# Verbindung schliessen
connection.close()
```

**Listing 10.12** Datei mysql\_datensatz.py

Zur Erläuterung:

- insert into**
- ▶ Nach Aufnahme der Verbindung zur Datenbank auf dem Server werden mithilfe des SQL-Befehls `insert into` drei Datensätze erzeugt.
  - ▶ Die Daten werden in der Reihenfolge der Felder angegeben. Zeichenketten stehen in einfachen Anführungsstrichen. Nachkommastellen werden mit einem Dezimalpunkt abgetrennt. Eine Datumsangabe erfolgt in der Form JJJJ-MM-TT, ebenfalls innerhalb von Anführungsstrichen.

- ▶ Das Programm hat keine Ausgabe. Im Programm *phpMyAdmin* können Sie sich von der Existenz der Datensätze überzeugen.
- ▶ Ein zweiter Aufruf desselben Programms würde zu einem Abbruch führen, da auf dem Feld `personalnummer` ein Primärschlüssel definiert wurde und somit kein Eintrag erzeugt werden kann, der dieselbe Personalnummer wie ein bereits existierender Datensatz hat.

Primärschlüssel

#### 10.4.5 Daten anzeigen

Im nachfolgenden Programm sollen alle Datensätze der Tabelle mit allen Inhalten angezeigt werden. Das Programm:

```
# Connector importieren
import sys, mysql.connector

# Verbindung zur Datenbank auf dem Datenbankserver erstellen
try:
    connection = mysql.connector.connect(host = "localhost", \
                                          user = "root", passwd = "", db = "firma")
except:
    print("Keine Verbindung zum Server")
    sys.exit(0)

# Execution-Objekt erzeugen
cursor = connection.cursor()

# Daten auslesen
cursor.execute("SELECT * from personen")
result = cursor.fetchall()

# Execution-Objekt schliessen
cursor.close()

# Verbindung schliessen
connection.close()

# Daten ausgeben
for data in result:
    print(str(data[0]) + ", " + str(data[1]) + ", " +
          str(data[2]) + ", " + str(data[3]) + ", " +
          str(data[4]))
```

**Listing 10.13** Datei `mysql_anzeigen.py`

Das Programm erzeugt die Ausgabe:

Mertens, Julia, 2297, 3621.5, 1959-12-30  
 Maier, Hans, 6714, 3500.0, 1962-03-15  
 Schmitz, Peter, 81343, 3750.0, 1958-04-12

Zur Erläuterung:

- select**
  - ▶ Nach Aufnahme der Verbindung zur Datenbank werden mithilfe des SQL-Befehls `select` alle Datensätze abgefragt.
  - ▶ Da es sich nicht um eine Aktionsabfrage handelt (die eine Änderung in der Datenbank vornimmt), sondern nur um eine Auswahlabfrage (die Informationen ermittelt), muss hier die Methode `commit()` nicht aufgerufen werden.
- fetchall()**
  - ▶ Die Methode `fetchall()` des `cursor`-Objekts liefert als Rückgabewert eine Liste, die das Ergebnis der Abfrage beinhaltet.
  - ▶ Als Letztes werden die Elemente dieser Liste einzeln durchlaufen. Jedes Element besteht wiederum aus Elementen, die einzeln in Zeichenketten umgewandelt und ausgegeben werden.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

## 10.5 Spiel, Version mit Highscore-Datenbank

Es folgt eine weitere Version des Spiels. Darin werden die Daten des Spielers (Name und Zeit) in einer SQLite-Datenbank dauerhaft gespeichert, sodass eine Highscore-Liste geführt werden kann.

### Direkter Zugriff

Der entscheidende Unterschied zur Version mit Highscore-Datei (siehe Abschnitt 8.10, »Spiel, Version mit Highscore-Datei«) liegt darin, dass keine separate Liste geführt wird, sondern

- ▶ die neuen Werte direkt und »unsortiert« in die Datenbank geschrieben werden,
- ▶ die vorhandenen Werte direkt und sortiert aus der Datenbank gelesen und auf dem Bildschirm ausgegeben werden.

### Vorteile

Dies hat einige Vorteile:

- ▶ Es muss keine eigene Funktion zum Austausch der Daten zwischen Liste und Datei geschrieben werden.
- ▶ Die Sortierung wird von SQL übernommen und muss nicht innerhalb des Python-Programms erfolgen.
- ▶ Bei einem vorzeitigen Abbruch des Programms gehen die bisher ermittelten Daten nicht verloren, da sie nach jedem Spiel gespeichert werden.

Es folgt der erste Teil des Programms:

```
# Module
import random, time, glob, sqlite3

# Funktion Highscore anzeigen
def hs_anzeigen():
    # Highscore-DB nicht vorhanden
    if not glob.glob("highscore.db"):
        print("Keine Highscores vorhanden")
        return

    # Highscores laden
    con = sqlite3.connect("highscore.db")
    cursor = con.cursor()
    sql = "SELECT * FROM daten ORDER BY zeit LIMIT 10"
    cursor.execute(sql)

    # Ausgabe Highscore
    print(" P. Name          Zeit")
    i = 1
    for dsatz in cursor:
        print("{0:2d}. {1:10} {2:5.2f} sec".format
              (i, dsatz[0], dsatz[1]))
        i = i+1

    # Verbindung beenden
    con.close()
```

**Listing 10.14** Datei spiel\_sqlite.py, Teil 1 von 3

Zur Erläuterung:

- ▶ In der Funktion `hs_anzeigen()` wird zunächst festgestellt, ob die Datenbank-Datei existiert. Ist dies der Fall, wird eine Verbindung aufgenommen.
- ▶ Es werden die ersten zehn Datensätze mithilfe der Anweisung `SELECT` ausgewählt, aufsteigend sortiert nach der Spieldauer. Der Zusatz `LIMIT` in der SQL-Abfrage begrenzt dabei die Anzahl.
- ▶ Nach der Ausgabe der Datensätze wird die Verbindung wieder beendet.

Der zweite Teil des Programms:

```
# Funktion Spiel
def spiel():
    . . .
    # Highscore-DB nicht vorhanden, erzeugen
    if not glob.glob("highscore.db"):
        con = sqlite3.connect("highscore.db")
```

```

cursor = con.cursor()
sql = "CREATE TABLE daten(" \
      "name TEXT, " \
      "zeit FLOAT)"
cursor.execute(sql)
con.close()

# Datensatz in DB schreiben
con = sqlite3.connect("highscore.db")
cursor = con.cursor()
sql = "INSERT INTO daten VALUES('' \
      + name + ',' + str(differenz) + ")"
cursor.execute(sql)
con.commit()
con.close()

# Highscoreliste anzeigen
hs_anzeigen()

```

Listing 10.15 Datei spiel\_sqlite.py, Teil 2 von 3

Zur Erläuterung:

- ▶ Der Anfang der Funktion `spiel()` unterscheidet sich nicht von der Version mit Highscore-Datei (siehe [Abschnitt 8.10](#), »Spiel, Version mit Highscore-Datei«). Erst bei der Speicherung zeigen sich die Änderungen.

**CREATE TABLE**

- ▶ Es wird zunächst festgestellt, ob die Datenbank-Datei existiert. Ist dies nicht der Fall, wird die Datenbank mit der Tabelle `daten` mithilfe der Anweisung `CREATE TABLE` erzeugt.

**INSERT**

- ▶ Die neu ermittelten Werte werden mithilfe der Anweisung `INSERT` als neuer Datensatz in die Datenbank geschrieben.
- ▶ Anschließend wird die Highscore-Liste angezeigt.

Das Hauptprogramm:

```

# Endlosschleife
while True:
    # Hauptmenue, Auswahl
    try:
        menu = int(input("Bitte eingeben "
                         "(0: Ende, 1: Highscores, 2: Spielen): "))
    except:
        print("Falsche Eingabe")
        continue

    # Aufruf einer Funktion oder Ende
    if menu == 0:
        break
    elif menu == 1:
        hs_anzeigen()

```

```

elif menu == 2:
    spiel()
else:
    print("Falsche Eingabe")

```

**Listing 10.16** Datei spiel\_sqlite.py, Teil 3 von 3

Zur Erläuterung:

- ▶ Gegenüber der Version mit Highscore-Datei (siehe [Abschnitt 8.10](#)) fehlen die beiden Aufrufe der Funktionen zum Austausch der Daten zwischen Liste und Datei.
- ▶ Im Hauptprogramm läuft eine Endlosschleife. Der Benutzer kann sich die Highscores anzeigen lassen, spielen oder das Programm beenden.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen `\` für den Umbruch von langen Programmzeilen eingesetzt. Das Weglassen des Zeilenendes wird erreicht, indem die Ausgabe stückweise in einer Variablen zusammengesetzt und erst anschließend vollständig ausgegeben wird.

## 10.6 Spiel, objektorientierte Version mit Highscore-Datenbank

Diese Version ist auf der objektorientierten Version mit Highscore-Datei aufgebaut. Nur die Klasse `Highscore` wurde verändert, um die in [Abschnitt 10.5, »Spiel, Version mit Highscore-Datenbank«](#), erwähnten Vorteile einer Datenbank (hier SQLite) gegenüber einer Datei zu nutzen.

Die Klasse `Highscore`:

```

class Highscore:
    # Highscore in DB speichern
    def speichern(self, name, zeit):
        # Highscore-DB nicht vorhanden, erzeugen
        if not glob.glob("highscore.db"):
            con = sqlite3.connect("highscore.db")
            cursor = con.cursor()
            sql = "CREATE TABLE daten(" \
                  "name TEXT, " \
                  "zeit FLOAT)"
            cursor.execute(sql)
            con.close()

    # Datensatz in DB schreiben
    con = sqlite3.connect("highscore.db")

```

**Klasse »Highscore«**

```

cursor = con.cursor()
sql = "INSERT INTO daten VALUES('" \
      + name + "','" + str(zeit) + "')"
cursor.execute(sql)
con.commit()
con.close()

# Highscore anzeigen
def __str__(self):
    # Highscore-DB nicht vorhanden
    if not glob.glob("highscore.db"):
        return "Keine Highscores vorhanden"

# Highscores laden
con = sqlite3.connect("highscore.db")
cursor = con.cursor()
sql = "SELECT * FROM daten" \
      " ORDER BY zeit LIMIT 10"
cursor.execute(sql)

# Ausgabe Highscore
ausgabe = " P. Name           Zeit\n"
i = 1
for dsatz in cursor:
    ausgabe += "{0:2d}. {1:10} {2:5.2f} sec\n". \
               format(i, dsatz[0], dsatz[1])
    i = i+1

# Verbindung beenden
con.close()
return ausgabe

```

**Listing 10.17** Datei spiel\_sqlite\_oop.py

Zur Erläuterung:

- ▶ In der Klasse `Highscore` gibt es nur noch die beiden Methoden zum Speichern und zum Ausgeben des Highscores.
- CREATE, INSERT** ▶ In der Methode `speichern()` wird geprüft, ob es eine Datenbank-Datei gibt. Ist dies nicht der Fall, werden die Datenbank und die Tabelle mithilfe von `CREATE TABLE` erzeugt. Falls es die Datenbank bereits gibt, werden die neu ermittelten Daten mithilfe der Anweisung `INSERT` hinzugefügt.
- SELECT** ▶ In der Ausgabemethode werden die Datensätze mithilfe der Anweisung `SELECT` sortiert ausgewählt und anschließend formatiert ausgegeben.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

# Kapitel 11

## Benutzeroberflächen

Die bisher dargestellten Programme wurden über Eingaben in der Kommandozeile bedient. Möchten Sie dem Benutzer dagegen eine komfortable grafische Oberfläche zur Bedienung zur Verfügung stellen, so bietet sich bei Python die Nutzung der Bibliothek *Tk* an.

Tk

Das Modul *tkinter* stellt eine Schnittstelle (engl.: *interface*) zu dieser Bibliothek dar. Es steht nach der Installation von Python unter den verschiedenen Betriebssystemen standardmäßig zur Verfügung. Es bietet eine Reihe von Klassen zur Erzeugung der einzelnen Elemente der Oberfläche. Die ereignisorientierte Programmierung rückt dabei in den Vordergrund.

tkinter

Es gibt weitere Bibliotheken für grafische Benutzeroberflächen mit einer Schnittstelle zu Python. Allerdings gestaltet sich die Einbindung teilweise recht aufwendig und unterscheidet sich in den verschiedenen Betriebssystemen. Hier sind z.B. das Gimp-Toolkit (*GTK+*) mit der Schnittstelle *pyGTK*, *wxWidgets* mit der Schnittstelle *wxPython* oder *QT* mit der Schnittstelle *PyQt* zu nennen. Im Rahmen dieses Einsteigerbuchs wird auf diese Themen nicht eingegangen.

GTK+, wxWidgets,  
QT

### 11.1 Einführung

Die Erstellung einer Oberflächen-Anwendung, einer sogenannten *GUI-Anwendung* (*GUI* = *Graphical User Interface*), umfasst die folgenden wichtigen Schritte:

GUI

- ▶ Erzeugung eines Hauptfensters, das alle Elemente der Anwendung enthält
- ▶ Erzeugung und Anordnung von einzelnen Steuerelementen (hier *Widgets* genannt) innerhalb des Hauptfensters zur Bedienung der Anwendung
- ▶ Starten einer *Endlosschleife*, in der die Bedienung der Steuerelemente durch den Benutzer zu Ereignissen und damit zum Ablauf einzelner Programmteile führt

Hauptfenster

Widgets

Endlosschleife

In den folgenden Abschnitten werden diese drei Schritte genauer erläutert.

#### 11.1.1 Eine erste GUI-Anwendung

Das folgende einfache Programm enthält bereits alle notwendigen Elemente einer GUI-Anwendung:

```
import tkinter

# Funktion zu Button Ende
def ende():
    main.destroy()

# Hauptfenster
main = tkinter.Tk()

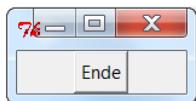
# Button Ende
b = tkinter.Button(main, text = "Ende", command = ende)
b.pack()

# Endlosschleife
main.mainloop()
```

**Listing 11.1 Datei gui\_ertes.py**

### Benutzeroberfläche

Das Ergebnis sieht abhängig vom verwendeten Betriebssystem unterschiedlich aus. In jedem Fall sieht ein Benutzer eine Oberfläche in gewohntem Design mit den ihm bekannten Bedienungselementen. Das Ergebnis für Windows, Ubuntu Linux und OS X zeigen [Abbildung 11.1](#) bis [Abbildung 11.3](#).



**Abbildung 11.1** Erstes Programm unter Windows



**Abbildung 11.2** Erstes Programm unter Ubuntu Linux



**Abbildung 11.3** Erstes Programm unter OS X

Zur Erläuterung:

- |                      |  |
|----------------------|--|
| <b>Modul tkinter</b> | <ul style="list-style-type: none"><li>▶ Das Modul <code>tkinter</code> mit den Klassen zur Erzeugung der Oberflächenelemente wird importiert.</li><li>▶ Die selbst geschriebene Funktion <code>ende()</code> wird definiert, die beim Betätigen des gleichnamigen Buttons aufgerufen wird.</li></ul> |
| <b>Klasse Tk</b>     | <ul style="list-style-type: none"><li>▶ Es wird ein Objekt der Klasse <code>Tk</code> mit dem Namen <code>main</code> erzeugt. Dabei handelt es sich um das Hauptfenster, das alle Elemente der Anwendung enthält.</li></ul>   |

- ▶ Als Beispiel für ein Steuerelement (ein Widget) wird ein Objekt der Klasse `Button` mit dem Namen `b` erzeugt. Dabei werden drei Parameter angegeben:
  - Der erste Parameter gibt an, zu welchem Hauptelement das Widget gehört – in diesem Fall zum Hauptfenster.
  - Der zweite, benannte Parameter `text` legt die Aufschrift des Buttons fest.
  - Der dritte, ebenfalls benannte Parameter `command` bestimmt den Namen der Funktion (ohne Klammern), die ausgeführt werden soll, wenn der Button betätigt wird. Eine solche Kommandofunktion wird *Callback* genannt.
  - Rückgabewert ist eine Referenz auf das Button-Objekt.
- ▶ Es handelt sich um die Funktion `ende()`. In der Funktion wird die Methode `destroy()` für das Hauptfenster ausgeführt. Diese Methode schließt das Fenster. Da es sich um das einzige Fenster der Anwendung handelt, wird damit auch die Anwendung geschlossen.
- ▶ Es wird die Methode `pack()` auf den Button `b` angewendet. Diese Methode dient als Geometriemanager zur geometrischen Anordnung eines Widgets innerhalb des Hauptfensters. Gleichzeitig dient der Geometriemanager dazu, das Widget sichtbar zu machen. In [Abschnitt 11.3, »Geometrische Anordnung von Widgets«](#), werden noch andere Geometriemanager erläutert.
- ▶ Zu guter Letzt wird mit der Funktion `mainloop()` die Endlosschleife für das Hauptfenster aufgerufen, die dafür sorgt, dass alle Benutzerereignisse empfangen und weitergeleitet werden.

## Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

### Hinweis

Sie können GUI-Anwendungen direkt aus IDLE heraus starten. Alternativ können Sie sie auch per Eingabe über die Kommandozeile starten, also mit `python gui_erstes.py` oder `python3 gui_erstes.py`, siehe auch [Abschnitt 2.3.2, »Ausführen unter Windows«, und Abschnitt 2.3.3, »Ausführen unter Ubuntu Linux und unter OS X«](#).

## 11.1.2 Ändern von Eigenschaften

Im ersten Beispiel wurden die Eigenschaften des Button-Widgents unmittelbar bei der Erzeugung festgelegt. Diese Eigenschaften können aber auch später eingestellt oder verändert werden. Ein Beispiel:

```

import tkinter

# Funktion zu drei Buttons
def ende():
    main.destroy()

# Hauptfenster
main = tkinter.Tk()

# Button Ende 1
b1 = tkinter.Button(main, text = "Ende", command = ende)

# Button Ende 2
b2 = tkinter.Button(main)
b2["text"] = "Auch Ende"
b2["command"] = ende

# Button Ende 3
b3 = tkinter.Button(main)
b3.configure(text = "Ebenfalls Ende", command = ende)

# Buttons 1 bis 3 anzeigen
b1.pack()
b2.pack()
b3.pack()

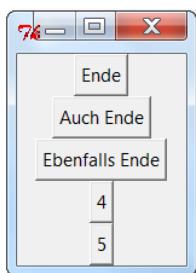
# Buttons Ende 4 und 5
b4 = Tkinter.Button(main,text="4",command=ende).pack()
Tkinter.Button(main,text="5",command=ende).pack()

# Endlosschleife
main.mainloop()

```

**Listing 11.2** Datei gui\_eigenschaft.py

Bei Ausführung des Programms werden mehrere Buttons angezeigt, die auf unterschiedliche Art und Weise erzeugt wurden, siehe Abbildung 11.4.

**Abbildung 11.4** Mehrere unterschiedlich erzeugte Buttons

Zur Erläuterung:

- ▶ Der Button `b1` wird bei Erzeugung mit allen Eigenschaften versorgt.
- ▶ Der Button `b2` wird zunächst nur erzeugt und dem Hauptfenster zugeordnet. Anschließend wird das Widget wie ein Dictionary genutzt. Die jeweilige Eigenschaft wird als Schlüssel verwendet, dem ein Wert zugewiesen wird.
- ▶ Der Button `b3` wird ebenfalls zunächst nur erzeugt und dem Hauptfenster zugeordnet. Anschließend wird die Methode `configure()` mit benannten Parametern auf das Widget angewendet, um ihm Eigenschaften zuzuweisen.
- ▶ Beim Button `b4` wird die Tatsache ausgenutzt, dass bei Erzeugung des Buttons eine Referenz auf den Button zurückgegeben wird. Diese Referenz wird unmittelbar genutzt, um die Methode `pack()` auf den Button anzuwenden.
- ▶ Der fünfte Button wird erzeugt, ohne eine Referenz auf ihn zu speichern. Dies ist nur möglich, falls die Referenz auf das Objekt im weiteren Verlauf des Programms nicht mehr benötigt wird. Natürlich muss nun die Methode `pack()` unmittelbar aufgerufen werden.

Widget-Dictionary

`configure()`

Rückgabe

Referenz nicht speichern

### Unterschiede in Python 2

Das Modul heißt Tkinter statt tkinter.

## 11.2 Widget-Typen

In diesem Abschnitt stelle ich eine Reihe von Widget-Typen für die verschiedenen Einsatzzwecke vor.

### 11.2.1 Anzeigefeld, Label

Ein Label-Widget wird zur Anzeige von Informationen verwendet. Dabei kann es sich um ein Bild oder um einen unveränderlichen Text handeln. Die Texte dienen in GUI-Anwendungen zur Erläuterung der Oberfläche und deren Bedienelemente für den Benutzer.

Label-Widget

Anhand von drei Labels sollen im folgenden Programm eine Reihe von allgemeinen Widget-Eigenschaften gezeigt werden, die bei allen Arten von Widgets zur Anwendung kommen können.

Allgemeine Eigenschaften

Dies sind die Eigenschaften:

- ▶ `font` (Schrift)
- ▶ `height` (Höhe)
- ▶ `width` (Breite)

- ▶ borderwidth (Randbreite)
- ▶ relief (Randart)
- ▶ image (Bild)
- ▶ bg (*Background*, Hintergrundfarbe)
- ▶ fg (*Foreground*, Vordergrundfarbe)
- ▶ anchor (Ausrichtung)

Das Programm sieht folgendermaßen aus:

```
import tkinter

def ende():
    main.destroy()

main = tkinter.Tk()

# Erstes Label, mit Text
lb1 = tkinter.Label(main, text = "groove")
lb1["font"] = "Courier 16 italic"
lb1["height"] = 2
lb1["width"] = 20
lb1["borderwidth"] = 5
lb1["relief"] = "groove"
lb1["bg"] = "#FFFFFF"
lb1["fg"] = "#000000"
lb1["anchor"] = "w"
lb1.pack()
# Ende-Button
b = tkinter.Button(main, text = "Ende", command = ende)
b.pack()

# Zweites Label, mit Text
lb2 = tkinter.Label(main, text = "ridge")
lb2["font"] = "Arial 11 bold"
lb2["height"] = 2
lb2["width"] = 20
lb2["borderwidth"] = 5
lb2["relief"] = "ridge"
lb2["bg"] = "#FFFFFF"
lb2["fg"] = "#000000"
lb2["anchor"] = "e"
lb2.pack()

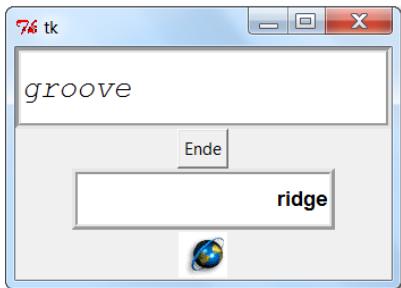
# Drittes Label, mit Bild
lb3 = tkinter.Label(main)
im = tkinter.PhotoImage(file="globus.gif")
lb3["image"] = im
```

```
lb3.pack()
```

```
main.mainloop()
```

**Listing 11.3** Datei `gui_label.py`

Das Ergebnis dieses Programms zeigt [Abbildung 11.5](#).



**Abbildung 11.5** Label

Es werden insgesamt drei Objekte der Klasse `Label` (`lb1`, `lb2` und `lb3`) erzeugt und angezeigt. Zur Erläuterung des ersten Labels:

- ▶ Es zeigt den Text »groove«. Dieser Text ist in der Schriftart Courier, Größe 16, **Schriftart** kursiv (*italic*) dargestellt.
- ▶ Die Größe des Labels beträgt 2 mal 20 und ist abhängig von der verwendeten **Größe**. Diese Abhängigkeit der Größe von der Schriftgröße gilt für alle Widgets mit Textinhalt. Bei Widgets mit grafischem Inhalt bezieht sich die Größenangabe auf Pixel.
- ▶ Die Randbreite beträgt 5, und die Randart ist groove. Es gibt außerdem die **Rand**arten raised (erhoben), sunken (eingesunken), flat (flach) und ridge (mit einem Grat umrandet, siehe zweites Label).
- ▶ Die Ausrichtung innerhalb des Labels hat den Wert "w", dies steht für den westlichen, also linken Rand des Labels. Erlaubte Werte sind: w, e, s, n, nw, ne, sw, se und center. **Ausrichtung**
- ▶ Die Hintergrundfarbe ist Weiß, die Vordergrundfarbe Schwarz. Die Farben können Sie unter anderem in hexadezimalen Ziffern im Format #RRGGBB angeben. Dabei gilt:
  - Die beiden ersten Ziffern stehen für den Rot-Anteil der Farbe.
  - Die folgenden beiden Ziffern stehen für den Grün-Anteil der Farbe.
  - Die beiden letzten Ziffern stehen für den Blau-Anteil der Farbe.
 Hexadezimale Ziffern sind (in aufsteigender Reihenfolge): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (entspricht dem Dezimalwert 10), B (= 11), C (= 12), D (= 13), E (= 14), F (= 15). Siehe hierzu auch [Abschnitt 4.1.1](#), »Ganze Zahlen«.

Zur Erläuterung des zweiten Labels:

- ▶ Es zeigt den Text »ridge«. Dieser Text ist in der Schriftart Arial, Größe 11, fett (**bold**) dargestellt.
- ▶ Die Größe des Labels beträgt ebenfalls 2 mal 20. Da die Schriftgröße geringer ist, ist es kleiner als das erste Label.
- ▶ Die Ausrichtung innerhalb des Labels hat den Wert "e". Dies steht für den östlichen, also rechten Rand des Labels (»e« für »east«).

Zur Erläuterung des dritten Labels:

- |                   |  |
|-------------------|--|
| <b>Bild</b>       | ▶ Es zeigt das Bild aus der Datei <i>globus.gif</i> .  |
|                   | ▶ Zunächst wird ein Objekt der Klasse <code>PhotoImage</code> erzeugt. Die Eigenschaft <code>file</code> erhält als Wert den Dateinamen. |
| <b>PhotoImage</b> | ▶ Der Eigenschaft <code>image</code> des Labels wird anschließend als Wert das Objekt der Klasse <code>PhotoImage</code> zugeordnet.     |

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

## 11.2.2 Einzelige Textbox, Entry

**Entry-Widget** Ein Entry-Widget ist eine einzelige Textbox und dient im Allgemeinen zur Eingabe von kurzen Texten oder Zahlen. Das folgende Beispiel soll die Verarbeitung von eingegebenen Daten und die Ausgabe von berechneten Werten verdeutlichen.

Der Benutzer soll eine Zahl in ein Entry-Widget eintragen. Diese Zahl wird quadriert, und das Ergebnis wird in einem Label-Widget ausgegeben.

```
import tkinter

def ende():
    main.destroy()

# Funktion zum Quadrieren und Ausgeben
def quad():
    eingabe = e.get()
    try:
        zahl = float(eingabe)
        lb["text"] = "Ergebnis:" + str(zahl * zahl)
    except:
        lb["text"] = "Bitte Zahl eingeben"

main = tkinter.Tk()
```

```
# einzeiliges Eingabefeld
e = tkinter.Entry(main)
e.pack()

# Button zur Verarbeitung und Ausgabe
bquad = tkinter.Button(main, text = "Quadrieren",
                      command = quad)
bquad.pack()

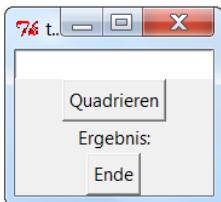
# Ausgabelabel
lb = tkinter.Label(main, text = "Ergebnis:")
lb.pack()

bende = tkinter.Button(main, text = "Ende",
                      command = ende)
bende.pack()

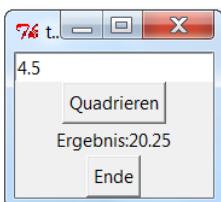
main.mainloop()
```

**Listing 11.4** Datei `gui_entry.py`

Zunächst sieht die Anwendung wie in Abbildung 11.6 gezeigt aus.

**Abbildung 11.6** Entry, Eingabeposition

Nach Eingabe einer Zahl durch den Benutzer (hier: 4.5) und Betätigung des Verarbeitungsbuttons wird das Ergebnis (hier: 20.25) im Label angezeigt, siehe Abbildung 11.7.

**Abbildung 11.7** Entry, Ergebnis

Zur Erläuterung:

- ▶ Insgesamt werden zwei Objekte der Klasse `Button` (`bquad` und `bende`), ein Objekt der Klasse `Label` (`lb`) und ein Objekt der Klasse `Entry` (`e`) erzeugt und angezeigt.
- ▶ Bei Betätigung des Buttons `QUADRIEREN` wird die Funktion `quad()` aufgerufen.
- get()**
- ▶ In der Funktion `quad()` wird die Methode `get()` auf das Eingabefeld angewendet. Sie liefert als Rückgabewert eine Zeichenkette, die die Benutzereingabe enthält. Diese Zeichenkette wird in der Variablen `eingabe` gespeichert.
- ▶ Mithilfe der Funktion `float()` wird die Zeichenkette in eine Zahl umgewandelt und in der Variablen `zahl` gespeichert.
- ▶ Die Zahl wird quadriert, das Ergebnis wird mit der Funktion `str()` in eine Zeichenkette umgewandelt und zusammen mit der Zeichenkette `Ergebnis:` im Label ausgegeben.

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

#### 11.2.3 Versteckte Eingabe

**show** Ein Entry-Widget kann auch zur Eingabe von versteckten Informationen, z. B. Passwörtern, dienen. Dazu dient die Eigenschaft `show`, in der festgelegt wird, welches Zeichen in einem Entry-Widget angezeigt wird, wenn der Benutzer Eingaben macht.

**Passworteingabe** Das folgende Beispiel veranschaulicht diese Nutzung. Die Eingabe in einem Entry-Widget wird untersucht. Gibt der Benutzer das richtige Wort ein, so erhält er die Rückmeldung *Zugang erlaubt*, anderenfalls die Meldung *Zugang verweigert*.

```
import tkinter
def ende():
    main.destroy()

# Untersuchung des Passwortes
def pwtest():
    eingabe = e.get()
    if eingabe == "Bingo":
        lb["text"] = "Zugang erlaubt"
    else:
        lb["text"] = "Zugang verweigert"

main = tkinter.Tk()
```

```
# Eingabefeld mit Zeichen * als Darstellung
e = tkinter.Entry(main, show = "*")
e.pack()

# Test der Eingabe
btest = tkinter.Button(main, text = "Login",
                      command = pwtest)
btest.pack()

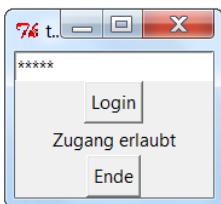
# Anzeige des Ergebnisses
lb = tkinter.Label(main, text = "Zugang")
lb.pack()

bende = tkinter.Button(main, text = "Ende",
                      command = ende)
bende.pack()

main.mainloop()
```

**Listing 11.5** Datei `gui_versteckt.py`

Das Ergebnis bei richtiger Eingabe (Bingo) sehen Sie in Abbildung 11.8.



**Abbildung 11.8** Richtig Eingabe

Das Ergebnis bei falscher Eingabe zeigt Abbildung 11.9.



**Abbildung 11.9** Falsche Eingabe

Zur Erläuterung:

- ▶ Die Eigenschaft `show` des Entry-Widgets wird auf den Wert `*` gesetzt. Daher `show = "***"` wird für jedes eingegebene Zeichen ein Stern angezeigt, wie bei Passwörtern üblich.

- ▶ Bei Betätigung des LOGIN-Buttons wird die Funktion `pwtest()` aufgerufen.
- ▶ In der Funktion `pwtest()` wird der Inhalt des Eingabefeldes mit der Funktion `get()` abgerufen und in der Variablen `eingabe` gespeichert. Dabei handelt es sich um den tatsächlich eingegebenen Inhalt; der Stern dient nur zur Darstellung auf dem Bildschirm.
- ▶ In der folgenden Verzweigung wird eine der beiden möglichen Ausgaben im Label erzeugt, abhängig vom Inhalt des Entry-Widgets.

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

#### 11.2.4 Mehrzeilige Textbox, Text

**Text-Widget** Ein Text-Widget ist eine mehrzeilige Textbox und dient im Allgemeinen zur Eingabe und Darstellung größerer Textmengen. Gegenüber dem Entry-Widget bietet es nicht nur mehr Platz, sondern auch mehr Möglichkeiten zur Verarbeitung von Daten. Das folgende Codebeispiel stellt den Inhalt einer Textdatei in einem Text-Widget dar.

```
import tkinter

def ende():
    main.destroy()

# Anzeigefunktion
def xshow():
    d = open("gui_text.txt")
    z = d.readline()
    while z:
        t.insert("end", z)
        z = d.readline()
    d.close()

main = tkinter.Tk()

# mehrzeiliges Eingabefeld
t = tkinter.Text(main, width=70, height=10)
t.pack()

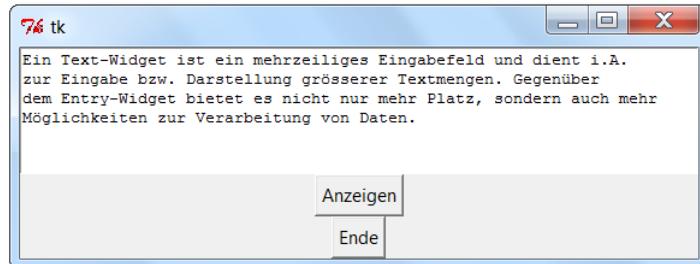
# Inhalt der Datei anzeigen
bshow = tkinter.Button(main, text = "Anzeigen",
                      command = xshow)
bshow.pack()
```

```
bende = tkinter.Button(main, text = "Ende",
                      command = ende)
bende.pack()

main.mainloop()
```

**Listing 11.6** Datei *gui\_text.py*

Abbildung 11.10 zeigt die Darstellung nach Aufruf des Programms und Betätigung des Buttons ANZEIGEN.

**Abbildung 11.10** Text-Widget mit Inhalt

Zur Erläuterung:

- ▶ Insgesamt werden zwei Objekte der Klasse `Button` (`bshow` und `bende`) und ein Objekt der Klasse `Text` (`t`) erzeugt und angezeigt.
- ▶ Bei Betätigung des Buttons ANZEIGEN wird die Funktion `xshow()` aufgerufen.
- ▶ In der Funktion `xshow()` wird die Datei *gui\_text.txt* geöffnet. Alle Zeilen der Datei werden gelesen und mithilfe der Methode `insert()` in der Textbox dargestellt.
- ▶ Die Methode `insert()` benötigt zwei Parameter. Der erste Parameter gibt die Eingabeposition als Zeichenkette an. Der zweite Parameter gibt die einzufügende Zeichenkette an.
- ▶ Im vorliegenden Beispiel wird für die Einfügeposition der festgelegte Begriff `end` genutzt, der die Position nach dem letzten Zeichen der letzten Zeile der Textbox bezeichnet.

`insert()``end`**Einfügeposition**

Zeichen können Sie aber auch an beliebigen Stellen in die Textbox einfügen. Die Einfügeposition geben Sie dabei in der Form `Zeile.Zeichen` an. Die Zeilennummerierung beginnt innerhalb der Textbox bei 1, die Zeichenummerierung innerhalb einer Zeile beginnt bei 0. Ein Beispiel:

Die Anweisung `t.insert("3.7", "hallo")` fügt den Text »hallo« in der dritten Zeile nach dem achten Zeichen ein. Dies gilt allerdings nur, falls diese Position existiert, falls also die Textbox bereits mindestens drei Zeilen Text und die dritte Zeile mindestens acht Zeichen enthält.

## Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

### 11.2.5 Scrollende Textbox, `ScrolledText`

#### `ScrolledText`-Widget

Falls die Breite der Textbox für den darzustellenden Text nicht ausreicht, wird der Text in der nächsten Zeile dargestellt. Wird die Anzahl der Zeilen zu groß, so muss der Cursor zur gewünschten Stelle bewegt werden, damit diese sichtbar wird.

#### Vertikales Scrollen

In diesem Zusammenhang bringt das `ScrolledText`-Widget einen Komfortgewinn: Es stellt eine Erweiterung einer mehrzeiligen Textbox dar und ermöglicht zumindest das vertikale Scrollen innerhalb der Textbox. Das Programm:

```
import tkinter, tkinter.scrolledtext

def ende():
    main.destroy()

# Anzeigefunktion
def xshow():
    d = open("gui_text.txt")
    z = d.readline()
    while z:
        t.insert("end",z)
        z = d.readline()
    d.close()

main = tkinter.Tk()

# mehrzeiliges Eingabefeld
t = tkinter.scrolledtext.ScrolledText(main, width=40,
                                         height=3)
t.pack()

# Inhalt der Datei anzeigen
bshow = tkinter.Button(main, text = "Anzeigen",
                       command = xshow)
bshow.pack()
bende = tkinter.Button(main, text = "Ende",
                       command = ende)
bende.pack()

main.mainloop()
```

**Listing 11.7** Datei `gui_scrolledtext.py`

Die Ausgabe sieht nach Aufruf, Anzeige und Betätigung der Scrollbar beispielsweise wie in Abbildung 11.11 aus.

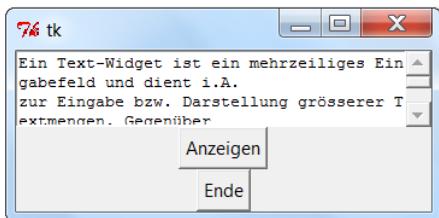


Abbildung 11.11 Scrollende TextBox

Zur Erläuterung:

- ▶ Es wird ein Objekt der Klasse `ScrolledText()` aus dem Modul `tkinter.scrolledtext` erzeugt und angezeigt.

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`. Das `ScrolledText`-Widget kommt aus dem Modul `ScrolledText`, nicht aus dem Modul `tkinter.scrolledtext`.

## 11.2.6 Listbox mit einfacher Auswahl

Ein Listbox-Widget (Auswahlmenü) dient zur Aufnahme einer Reihe von Begriffen, aus denen der Benutzer einen oder mehrere auswählen kann.

Listbox-Widget

Im folgenden Programm wird eine Listbox mit insgesamt vier Elementen versehen. Wenn der Benutzer ein Element auswählt und den Button ANZEIGEN betätigt, wird das betreffende Element im Ausgabelabel angezeigt.

```
import tkinter

def ende():
    main.destroy()

def anzeigen():
    lb["text"] = "Auswahl: " + li.get("active")

main = tkinter.Tk()

# Listbox mit vier Elementen
li = tkinter.Listbox(main, height=0)
li.insert("end","Hamburg")
li.insert("end","Stuttgart")
li.insert("end","Berlin")
li.insert("end","Dortmund")
```

```

        li.pack()

        # Auswahl anzeigen lassen
        bshow = tkinter.Button(main, text = "Anzeigen",
                               command = anzeigen)
        bshow.pack()

        # Anzeigelabel
        lb = tkinter.Label(main, text = "Auswahl:")
        lb.pack()

        bende = tkinter.Button(main, text = "Ende",
                               command = ende)
        bende.pack()

    main.mainloop()

```

**Listing 11.8** Datei `gui_listbox.py`

Nach Aufruf, Auswahl eines Elements (hier: Stuttgart) und Betätigung des Anzeigebuttons ergibt sich die Darstellung in [Abbildung 11.12](#).

Zur Erläuterung:

- ▶ Insgesamt werden zwei Objekte der Klasse `Button` (`bshow` und `bende`), ein Objekt der Klasse `Label` (`lb`) und ein Objekt der Klasse `Listbox` (`li`) erzeugt und angezeigt.
- ▶ Die Listbox erhält bei Erzeugung die Höhe 0. Damit erreichen wir, dass die Höhe immer der Anzahl der Elemente angepasst wird.

**Abbildung 11.12** Liste mit einfacher Auswahl

- `insert()`
- ▶ Die Methode `insert()` dient zur Erzeugung von Elementen in der Listbox. Sie benötigt zwei Parameter. Der erste Parameter gibt – ähnlich wie bei der Textbox – die Eingabeposition an. Der zweite Parameter bestimmt das einzufügende Element.
- `end`
- ▶ Als Einfügeposition wird der festgelegte Begriff `end` genutzt, der die Position am Ende der Liste bezeichnet. Die Elemente werden jeweils am Ende der

Liste eingefügt. Sie werden also in der Reihenfolge angezeigt, in der sie im Programm angegeben sind.

- ▶ In der Funktion `anzeigen()` wird die Methode `get()` auf das Label angewendet. Diese liefert das Listenelement zur angegebenen Nummer. Der festgelegte Begriff `active` liefert immer die Nummer des vom Benutzer ausgewählten Elements.
- ▶ Der über `get()` gelieferte Begriff wird im Ausgabelabel angezeigt.

`get("active")`

### Unterschiede in Python 2

Das Modul heißt Tkinter statt tkinter.

#### Hinweise

1. Elemente können Sie auch an beliebigen Stellen in die Listbox einfügen. Die Einfügeposition geben Sie dann als Zahl an. Die Nummerierung beginnt dabei bei 0.
2. Ein Beispiel: Hätten wir bei allen vier Aufrufen der Methode `insert()` die Nummer 0 anstelle der Zeichenkette `end` angegeben, so wäre jedes Element jeweils an erster Position eingefügt worden. Die Elemente wären also in umgekehrter Reihenfolge erschienen.

### 11.2.7 Listbox mit mehrfacher Auswahl

Im folgenden Programm wird das gleiche Listbox-Widget wie im vorherigen Abschnitt angezeigt. Allerdings kann der Benutzer nun mehrere Elemente auswählen. Nach Auswahl und Betätigung des Buttons ANZEIGEN werden alle betreffenden Elemente im Ausgabelabel angezeigt.

Listbox-Widget

```
import tkinter

def ende():
    main.destroy()

def anzeigen():
    lb["text"] = "Anzeige: "
    for x in li.curselection():
        lb["text"] = lb["text"] + li.get(x) + " "

main = tkinter.Tk()

# Listbox mit vier Elementen, mehrfache Auswahl
li = tkinter.Listbox(main, height=0,
                     selectmode="multiple")
```

```

        li.insert("end","Hamburg")
        li.insert("end","Stuttgart")
        li.insert("end","Berlin")
        li.insert("end","Dortmund")
        li.pack()

# Auswahl anzeigen lassen
bshow = tkinter.Button(main, text = "Anzeigen",
                      command = anzeigen)
bshow.pack()

# Anzeigelabel
lb = tkinter.Label(main, text = "Auswahl:")
lb.pack()

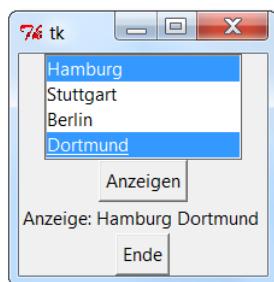
bende = tkinter.Button(main, text = "Ende",
                      command = ende)
bende.pack()

main.mainloop()

```

**Listing 11.9** Datei `gui_listbox_mehrfach.py`**Ohne Sondertasten**

Bei der Auswahl von mehreren Elementen ist zu beachten, dass einfach nur ein Element nach dem anderen angeklickt wird, ohne die gewohnte Betätigung der Sondertasten **Strg** oder **←**. Ein erneuter Klick auf ein ausgewähltes Element macht dessen Auswahl wieder rückgängig. Die Darstellung nach Aufruf, Auswahl von zwei Elementen (hier: Hamburg und Dortmund) und Betätigung des Anzeigebuttons zeigt [Abbildung 11.13](#).

**Abbildung 11.13** Liste mit Mehrfachauswahl

Zur Erläuterung:

**selectmode**

- ▶ Die Listbox wird zusätzlich mit der Eigenschaft `selectmode` und dem Eigenschaftswert `multiple` versehen. Bei `selectmode` ist der Wert `single` standardmäßig eingestellt und musste deshalb beim vorherigen Beispiel (Listbox mit einfacher Auswahl) nicht angegeben werden.

- Die Nummern der vom Benutzer ausgewählten Elemente stehen in einem Tupel zur Verfügung, das von der Methode `curselection()` zurückgeliefert wird.
- In der Funktion `anzeigen()` werden die einzelnen Elemente dieses Tupels ermittelt. Die zugehörigen Elemente aus der Listbox werden nacheinander im Ausgabelabel angezeigt.

`curselection()`

### Unterschiede in Python 2

Das Modul heißt Tkinter statt tkinter.

## 11.2.8 Scrollbar, scrollende Widgets

Einige Widgets können mit einer Scrollbar angezeigt werden. Dazu ist es notwendig, das betreffende Widget sowie eine Scrollbar zu erzeugen und miteinander zu verbinden.

Im folgenden Programm wird eine Listbox mit der Höhe 4 mit einer Scrollbar verbunden, sodass alle sieben Elemente der Listbox sichtbar gemacht werden können.

`Widget und  
Scrollbar verbinden`

```
import tkinter

def ende():
    main.destroy()

main = tkinter.Tk()

button = tkinter.Button(main, text="Ende",
                       command=ende)
button.pack()

# Erzeugen der Scrollbar
scb = tkinter.Scrollbar(main, orient="vertical")

# Erzeugen der Listbox, Verbindung mit der Scrollbar
li = tkinter.Listbox(main, height=4,
                     yscrollcommand=scb.set)
scb["command"] = li.yview

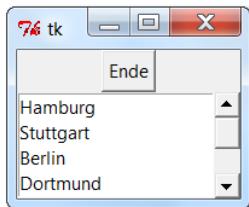
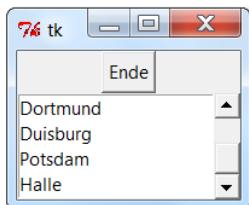
# Sieben Elemente
stadt = ["Hamburg", "Stuttgart", "Berlin", "Dortmund",
         "Duisburg", "Potsdam", "Halle"]
for s in stadt:
    li.insert("end", s)
```

```
# Anzeigen von Listbox und Scrollbar
li.pack(side="left")
scb.pack(side="left", fill="y")

main.mainloop()
```

**Listing 11.10** Datei `gui_scrollbar.py`

Abbildung 11.14 und Abbildung 11.15 zeigen die Darstellung mit zwei verschiedenen Stellungen der Scrollbar.

**Abbildung 11.14** Scrollendes Widget, Startzustand**Abbildung 11.15** Scrollendes Widget, nach Scrollvorgang

Zur Erläuterung:

- Scrollbar**
  - ▶ Zunächst wird ein Objekt der Klasse `Scrollbar` erzeugt (`scb`). Die Scrollbar ist vertikal ausgerichtet.
  - ▶ Eine Listbox (`li`) mit der Anzeigehöhe 4 wird erzeugt.
- Zweiseitige Zuweisung**
  - ▶ Wenn die Scrollbar bewegt wird, soll die Listbox entsprechend nach oben oder unten scrollen. Dies erreichen Sie durch zwei Maßnahmen:
    - Zuweisung der Methode `set()` der Scrollbar zur Eigenschaft `yscrollcommand` der Listbox
    - Zuweisung der Methode `yview()` der Listbox zur Eigenschaft `command` der Scrollbar
  - ▶ Die Listbox wird mit insgesamt sieben Elementen gefüllt.
- pack()**
  - ▶ Mithilfe des Geometrie-Managers `pack()` werden Listbox und Scrollbar nebeneinander angezeigt, beide nach links ausgerichtet. Die Scrollbar füllt den Platz in y-Richtung neben der Listbox aus, wird also genauso hoch wie

die Listbox. Weitere Angaben zu den Möglichkeiten von `pack()` erhalten Sie später in Abschnitt 11.3, »Geometrische Anordnung von Widgets«.

### Unterschiede in Python 2

Das Modul heißt Tkinter statt `tkinter`.

## 11.2.9 Radiobuttons zur Auswahl, Widget-Variablen

Radiobuttons (Optionsfelder) ermöglichen wie die einfache Listbox eine Auswahl durch den Benutzer. Sie werden allerdings als einzelne Widgets angezeigt und müssen vom Programmierer zu einer Gruppe zusammengefügt werden, damit sie gemeinsam reagieren können. Dazu ist die Verbindung zu einer sogenannten Widget-Variablen zwingend notwendig.

**Radiobutton-Widget**

### Hinweis

Sie können mehrere Arten von Widgets, nicht nur Radiobuttons, mit Widget-Variablen verbinden. Eine Änderung des Widget-Zustands führt dann zu einer Änderung des Werts der Widget-Variablen und umgekehrt.

**Widget-Variablen**

Im folgenden Beispiel wird eine Gruppe von drei Radiobuttons erzeugt und angezeigt, mit deren Hilfe der Benutzer eine Farbe auswählen kann. Betätigt er den Anzeigebutton, so wird die gewählte Farbe angezeigt.

```
import tkinter

def ende():
    main.destroy()

def anzeigen():
    lb["text"] = "Auswahl: " + farbe.get()

main = tkinter.Tk()

# Widget-Variable
farbe = tkinter.StringVar()
farbe.set("rot")

# Gruppe von Radiobuttons
rb1 = tkinter.Radiobutton(main, text="rot",
                          variable=farbe, value="rot")
rb1.pack()
rb2 = tkinter.Radiobutton(main, text="gelb",
                          variable=farbe, value="gelb")
```

```

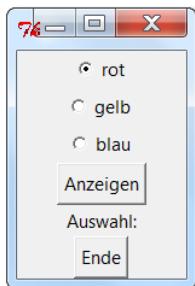
rb2.pack()
rb3 = tkinter.Radiobutton(main, text="blau",
                           variable=farbe, value="blau")
rb3.pack()

# Auswahl anzeigen lassen
bshow = tkinter.Button(main, text = "Anzeigen", command = anzeigen)
bshow.pack()
# Anzeigelabel
lb = tkinter.Label(main, text = "Auswahl:")
lb.pack()
bende = tkinter.Button(main, text = "Ende", command = ende)
bende.pack()
main.mainloop()

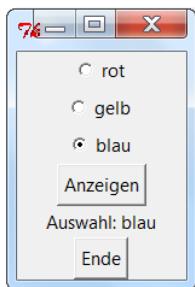
```

**Listing 11.11** Datei `gui_radio_auswahl.py`

Die Anzeige sieht zunächst aus wie in Abbildung 11.16.

**Abbildung 11.16** Radiobuttons, Startzustand

Wenn der Benutzer BLAU auswählt und den Anzeigebutton betätigt, sieht die Darstellung aus wie in Abbildung 11.17.

**Abbildung 11.17** Radiobuttons, Zustand nach Auswahl und Anzeige

Zur Erläuterung:

- ▶ Widget-Variablen müssen Objekte aus einer der folgenden Klassen sein: **Objekttypen**
  - StringVar (für Zeichenketten)
  - IntVar (für ganze Zahlen)
  - DoubleVar (für Zahlen mit Nachkommastellen)
- ▶ Im vorliegenden Beispiel wird die Widget-Variable `farbe` als Objekt der Klasse `StringVar` erzeugt. Mithilfe der Methode `set()` wird der Widget-Variablen der Wert `rot` zugewiesen. **StringVar**
- ▶ Es werden drei Radiobuttons erzeugt. Der gemeinsame Wert der Eigenschaft `variable` ist der Name der Widget-Variablen (`farbe`). Dadurch ist die Verbindung zwischen den drei Radiobuttons hergestellt. Die Buttons repräsentieren nun diese Widget-Variable und umgekehrt. **Gemeinsame Widget-Variable**
- ▶ Jeder der drei Radiobuttons erhält individuelle Werte für die Eigenschaften `text` und `value`. Der Text erscheint sichtbar neben dem Radiobutton. Der Wert der Eigenschaft `value` entspricht (unsichtbar) dem Wert der zugehörigen Widget-Variablen. **text, value**
- ▶ Wählt der Benutzer einen der drei Radiobuttons aus, so wird der Widget-Variablen der zugehörige Wert der Eigenschaft `value` zugewiesen und umgekehrt. **Auswahl = Zuweisung**
- ▶ Da zu Beginn für die Widget-Variable `farbe` der Wert `rot` eingestellt wurde, ist der zugehörige Radiobutton beim Start des Programms bereits ausgewählt. Eine solche Vorbelegung gehört zum guten Programmierstil und verhindert, dass der Benutzer gar keine Auswahl trifft. **Vorbelegung**
- ▶ In der Funktion `anzeigen()`, die über den Button aufgerufen wird, wird mithilfe der Methode `get()` der Wert der Widget-Variablen und damit die vom Benutzer getroffene Auswahl ermittelt. Dieser Wert wird im Label angezeigt.

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

#### 11.2.10 Radiobuttons zur Auswahl und Ausführung

Die Betätigung eines Radiobuttons kann auch unmittelbar zur Ausführung einer Funktion führen, da Radiobutton-Widgets wie die Standardbuttons über die Eigenschaft `command` verfügen.

Das oben aufgeführte Programm wurde in dieser Hinsicht verändert. Der Anzeigebutton entfällt. Stattdessen wurde die Eigenschaft `command` jedem der drei Radiobuttons hinzugefügt. Als Resultat wird die ausgewählte Farbe unmit-

**command**

telbar nach der Betätigung des Radiobuttons angezeigt. Das veränderte Programm sieht wie folgt aus:

```
import tkinter
def ende():
    main.destroy()
def anzeigen():
    lb["text"] = "Auswahl: " + farbe.get()
main = tkinter.Tk()
# Widget-Variable
farbe = tkinter.StringVar()
farbe.set("rot")
# Gruppe von Radiobuttons
rb1 = tkinter.Radiobutton(main, text="rot", variable=farbe,
                           value="rot", command=anzeigen)
rb1.pack()
rb2 = tkinter.Radiobutton(main, text="gelb", variable=farbe,
                           value="gelb", command=anzeigen)
rb2.pack()
rb3 = tkinter.Radiobutton(main, text="blau", variable=farbe,
                           value="blau", command=anzeigen)
rb3.pack()
# Anzeigelabel
lb = tkinter.Label(main, text = "Auswahl:")
lb.pack()
bende = tkinter.Button(main, text = "Ende",
                       command = ende)
bende.pack()
main.mainloop()
```

**Listing 11.12** Datei `gui_radio_aufuehrung.py`

Die Darstellung nach Auswahl von BLAU sehen Sie in [Abbildung 11.18](#).

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.



**Abbildung 11.18** Radiobuttons, nach Auswahl

### 11.2.11 Checkbuttons zur mehrfachen Auswahl

*Checkbuttons* (auch *Checkboxen* oder *Kontrollkästchen* genannt) bieten ebenso wie eine Listbox mit mehrfacher Auswahl dem Benutzer mehrere Entscheidungsmöglichkeiten. Checkbuttons werden als einzelne Widgets angezeigt und können mit einer Widget-Variablen verbunden werden. Dabei gibt es abhängig vom Zustand des Checkbuttons unterschiedliche Werte für die Variable.

Ihnen stehen wie beim Radiobutton zwei Möglichkeiten zur Verfügung:

- ▶ Sie lassen den Benutzer die Auswahl treffen und starten anschließend über einen Standardbutton eine Anzeige- oder Auswertungsfunktion.
- ▶ Sie verbinden die Betätigung eines Checkbuttons unmittelbar mit einer Aktion.

Im folgenden Programm wird eine Reservierung für ein Hotelzimmer durchgeführt. Mithilfe von zwei Checkbuttons kann der Benutzer entscheiden, ob er ein Zimmer mit oder ohne Dusche sowie mit oder ohne Minibar möchte. Jede Auswahl führt unmittelbar zur Anzeige.

```
import tkinter

def ende():
    main.destroy()

def anzeigen():
    lb["text"] = "Zimmer " + du.get() + " " + mb.get()

main = tkinter.Tk()

# Anzeigelabel
lb = tkinter.Label(main, text = "Zimmer ", width=40)
lb.pack()

# Widget-Variablen
du = tkinter.StringVar()
du.set("ohne Dusche")
mb = tkinter.StringVar()
mb.set("ohne Minibar")

# Zwei Checkbuttons
cb1 = tkinter.Checkbutton(main, text="Dusche",
                           variable=du, onvalue="mit Dusche",
                           offvalue="ohne Dusche", command=anzeigen)
cb1.pack()
cb2 = tkinter.Checkbutton(main, text="Minibar",
                           variable=mb, onvalue="mit Minibar",
                           offvalue="ohne Minibar", command=anzeigen)
```

Checkbutton-Widget

Bedienungsvarianten

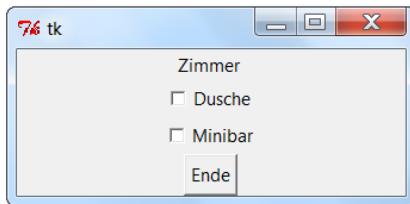
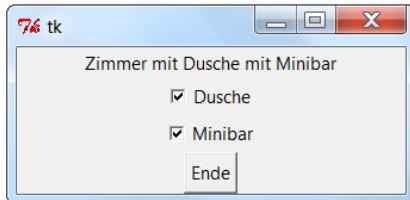
```

cb2.pack()

bende = tkinter.Button(main, text = "Ende",
                      command = ende)
bende.pack()

main.mainloop()

```

**Listing 11.13** Datei `gui_check.py`**Abbildung 11.19** zeigt den Startzustand des Formulars.**Abbildung 11.19** Checkbuttons, StartzustandNach Markierung beider Checkbuttons sieht das Formular aus wie in [Abbildung 11.20](#).**Abbildung 11.20** Checkbuttons, nach Markierung

Zur Erläuterung:

- Vorbelegung**
- ▶ Es werden zwei Widget-Variablen erzeugt (`du` und `mb`). Diese werden jeweils mit einem Wert vorbelegt, der dem nicht markierten Zustand (d. h. ohne Dusche bzw. Minibar) entspricht.

- ▶ Es werden zwei Checkbuttons mit den folgenden Eigenschaften erzeugt:

- `text` (zur Anzeige)
- `variable` (zur Verbindung mit der Widget-Variablen)
- `onvalue` (als Wert der Widget-Variablen für den markierten Zustand)
- `offvalue` (als Wert der Widget-Variablen für den nicht markierten Zustand)
- `command` (für die Funktion, die bei Betätigung des Checkbuttons aufgerufen wird)

- In der Funktion `anzeigen()` wird jeweils mithilfe der Methode `get()` der Wert der beiden Widget-Variablen und damit die vom Benutzer getroffene Auswahl ermittelt. Diese Werte werden im Label angezeigt.

### Unterschiede in Python 2

Das Modul heißt Tkinter statt tkinter.

#### 11.2.12 Schieberegler, Scale

Ein Scale-Widget entspricht einem Schieberegler, also der visuellen Anzeige eines Werts, der mithilfe der Maus verändert werden kann.

Scale-Widget

Im folgenden Beispiel wird ein Geschwindigkeitsmesser simuliert. Es können Geschwindigkeiten zwischen 0 und 200 km/h in Schritten von 5 km/h angezeigt und eingestellt werden. Der eingestellte Wert wird zusätzlich in einem Label angezeigt.

Visuelle Anzeige

```
import tkinter

def ende():
    main.destroy()

def anzeigen(self):
    lb["text"] = "Geschwindigkeit: " \
        + str(scwvert.get()) + " km/h"
main = tkinter.Tk()

# Anzeigelabel
lb = tkinter.Label(main,
    text = "Geschwindigkeit: 0 km/h", width=25)
lb.pack()

# Widget-Variablen
scwvert = tkinter.IntVar()
scwvert.set(0)

# Scale Widget
scv = tkinter.Scale(main, width=20, length=200,
    orient="vertical", from_=0, to=200,
    resolution=5, tickinterval=20, label="km/h",
    command=anzeigen, variable=scwvert)
scv.pack()

bende = tkinter.Button(main, text = "Ende",
    command = ende)
```

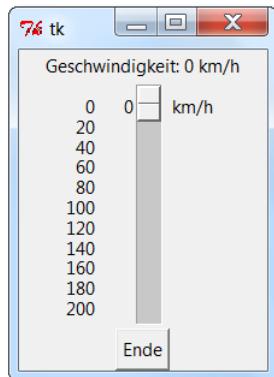
```
bende.pack()
```

```
main.mainloop()
```

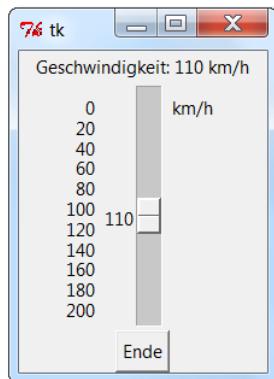
**Listing 11.14** Datei `gui_scale.py`

Die Darstellung zu Beginn sehen Sie in Abbildung 11.21.

Abbildung 11.22 zeigt den Geschwindigkeitsmesser nach einer Beschleunigungsphase.



**Abbildung 11.21** Scale, Startzustand



**Abbildung 11.22** Scale, nach Veränderung

Zur Erläuterung:

- ▶ Es wird eine Widget-Variable erzeugt (`scv`). Diese wird mit dem Wert 0 vorbelegt.
- ▶ Es wird ein Scale-Widget mit den Eigenschaften und Werten aus Tabelle 11.1 erzeugt.

Eigenschaft	Wert	Erklärung
width	20	Breite in Pixeln
length	200	Länge oder Höhe in Pixeln
orient	"vertical"	Orientierung, Ausrichtung
from_	0	Minimalwert; der Unterstrich dient dazu, Verwechslungen mit dem Schlüsselwort from zu vermeiden
to	200	Maximalwert
resolution	5	Auflösung; Abstand der Werte bei Bewegung des Schiebereglers
tickinterval	20	Anzeige der Werte im eingestellten Abstand
label	"km/h"	zusätzliche Anzeige neben dem Schieberegler
command	"anzeigen"	Funktion, die bei Betätigung des Schiebereglers aufgerufen wird
variable	scwert	Name der zugeordneten Widget-Variablen

Tabelle 11.1 Eigenschaften des Scale-Widgets

- In der Funktion `anzeigen()` wird jeweils mithilfe der Methode `get()` der Wert der Widget-Variablen und damit der vom Benutzer eingestellte Wert ermittelt. Dieser Wert wird im Label angezeigt.

`get()`

### Unterschiede in Python 2

Das Modul heißt Tkinter statt tkinter.

#### 11.2.13 Mausereignisse

Die Widgets, die in den bisherigen Programmen zu Aktionen geführt haben, enthalten die Eigenschaft `command`. Über diese Eigenschaft wird das Widget mit einer Funktion verbunden, die bei dem entsprechenden Ereignis – also beim Anklicken des Buttons, Radiobuttons oder Checkbuttons – aufgerufen wird.

`command`

Widgets können Sie mit vielen Events, also Mausereignissen oder Tastaturereignissen (siehe nächster Abschnitt 11.2.14), verbinden. Zu den Mausereignissen gehören:

`Event`

- ▶ Anklicken des Widgets mit der linken oder rechten Maustaste (Ereignis `<Button1>` bzw. `<Button3>`)
- ▶ doppeltes oder dreifaches Anklicken mit einer Maustaste (`<Double-Button>` und `<Triple-Button>`)
- ▶ gleichzeitige Betätigung einer oder mehrerer der Sondertasten `Strg`, `Alt` oder `Shift` (die Ereignisse `<Control-Button>`, `<Alt-Button>`, `<Shift-Button>`)
- ▶ Loslassen der Maustaste in einem Widget (Ereignis `<ButtonRelease>`)
- ▶ Betreten eines Widgets, also Bewegung der Maus in ein Widget hinein (Ereignis `<Enter>`)
- ▶ Verlassen eines Widgets, also Bewegung der Maus aus einem Widget heraus (Ereignis `<Leave>`)
- ▶ Bewegung der Maus innerhalb eines Widgets (Ereignis `<Motion>`)

In dem folgenden Programm werden die genannten Möglichkeiten vorgeführt:

```
import tkinter

def ende():
    main.destroy()

def mlinks(e):
    lbanz["text"] = "Linke Maustaste"

def mrechts(e):
    lbanz["text"] = "Rechte Maustaste"
def mstrglinks(e):
    lbanz["text"] = "Strg-Taste und linke Maustaste"

def maltlinks(e):
    lbanz["text"] = "Alt-Taste und linke Maustaste"

def mshiftlinks(e):
    lbanz["text"] = "Shift-Taste und linke Maustaste"

def mlinkslos(e):
    lbanz["text"] = "Linke Maustaste losgelassen"

def mbetreten(e):
    lbanz["text"] = "Button betreten"

def mverlassen(e):
    lbanz["text"] = "Button verlassen"

def mbewegt(e):
    lbanz["text"] = \
        "Koordinaten: x=" + str(e.x) + ", y=" + str(e.y)
```

```

main = tkinter.Tk()

# Label mit Events
im = tkinter.PhotoImage(file="figur.gif")
lbn = tkinter.Label(main, image=im)
lbn.bind("<Button 1>", mlinks)
lbn.bind("<Button 3>", mrechts)
lbn.bind("<Control-Button 1>", mstrglinks)
lbn.bind("<Alt-Button 1>", maltlinks)
lbn.bind("<Shift-Button 1>", mshiftlinks)
lbn.bind("<ButtonRelease 1>", mlinkslos)
lbn.bind("<Motion>", mbewegt)
lbn.pack()

# Anzeigelabel
lbanz = tkinter.Label(main, width=35)
lbanz.pack()

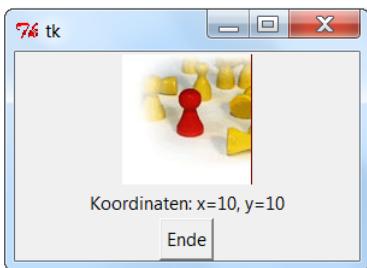
# Ende-Button, mit Events
bende = tkinter.Button(main, text = "Ende",
                       command = ende)
bende.bind("<Enter>", mbetreten)
bende.bind("<Leave>", mverlassen)
bende.pack()

main.mainloop()

```

**Listing 11.15** Datei `gui_maus.py`

In Abbildung 11.23 befindet sich die Maus in der Nähe der linken oberen Bild-ecke.



**Abbildung 11.23** Mausereignis, Maus über Widget bewegt

Hält der Benutzer die rechte Maustaste über dem Bild gedrückt, so ergibt sich **Rechte Maustaste** die Darstellung in Abbildung 11.24.



Abbildung 11.24 Mausereignis, rechte Maustaste über Widget betätigt

Zur Erläuterung:

- ▶ Es wurden insgesamt neun Funktionen definiert, die mit bestimmten Ereignissen verbunden sind. In einem zusätzlichen Label wird das jeweilige Ereignis für den Benutzer angezeigt.
- bind()** ▶ Zur Verbindung zwischen einem bestimmten Ereignis und einem Widget müssen Sie die Methode `bind()` für das betreffende Widget aufrufen.
  - Der erste Parameter der Methode ist eine Zeichenkette, in der das Ereignis bezeichnet wird.
  - Der zweite Parameter ist der Name der Funktion, die aufgerufen wird, wenn das Ereignis eintritt.
- Ereignisse verbinden** ▶ Die beiden Ereignisse *Betreten* und *Verlassen* eines Widgets wurden mit dem ENDE-Button verbunden. Die übrigen sieben Ereignisse wurden mit einem Label verbunden, in dem ein Bild dargestellt wird.
- Event-Objekt** ▶ Jede Ereignisfunktion übermittelt Informationen über das Ereignis mithilfe eines Event-Objekts (hier: `e`). Bei einem Mausereignis sind dabei die Objekteigenschaften `x` und `y` interessant, also die Koordinaten des Mauszeigers innerhalb des Widgets. Diese werden beim Ereignis *Bewegen* zusätzlich angezeigt (siehe Abbildung 11.23).

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

### Unterschiede unter Ubuntu Linux und OS X

Die `[Alt]`-Taste wird bereits vom Betriebssystem abgefangen, sie dient in diesem Fall zum Verschieben des GUI-Programms. Im Python-Programm kommt das Ereignis »Alt-Taste betätigt« daher nicht mehr an.

### 11.2.14 Tastaturereignisse

Die Verbindung zu Tastaturereignissen bauen Sie ähnlich wie die Verbindung zu Mausereignissen auf. Ein Beispiel:

```
import tkinter

def ende():
    main.destroy()

def kev(e):
    lbanz["text"] = "Zeichen:" + e.char \
        + ", Beschreibung: " + e.keysym \
        + ", Codezahl: " + str(e.keycode)

main = tkinter.Tk()

# Key-Events
e = tkinter.Entry(main)
e.bind("<p>",kev)
e.bind("<+>",kev)
e.bind("<%>",kev)
e.bind("<,>",kev)
e.pack()

# Hilfetext
lbhlp = tkinter.Label(main,
    text = "Taste: p oder + oder % oder ,",
    width=40)
lbhlp.pack()

# Anzeigetext
lbanz = tkinter.Label(main)
lbanz.pack()

# Ende-Button
bende = tkinter.Button(main, text = "Ende",
    command = ende)
bende.pack()

main.mainloop()
```

**Listing 11.16** Datei `gui_tastatur.py`

Abbildung 11.25 zeigt die Darstellung nach Eingabe eines Prozentzeichens.

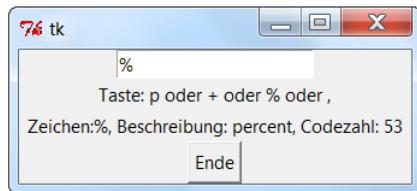


Abbildung 11.25 Tastaturereignis

Zur Erläuterung:

- bind()**
  - Das Entry-Widget wird mithilfe der Methode `bind()` mit insgesamt vier Tastaturereignissen verbunden:
    - Betätigen der Taste `[P]` (Buchstabe »p«)
    - Betätigen der Taste `[+]` (Pluszeichen)
    - Betätigen der Taste `[%]` (Prozentzeichen)
    - Betätigen der Taste `[,]` (Kommazeichen)
- Ereignisse verbinden**
  - Die Ereignisse werden jeweils mit der Funktion `kev()` verbunden, ebenfalls mithilfe der Methode `bind()`.
- Event-Objekt**
  - Jede Ereignisfunktion übermittelt Informationen über das Ereignis mithilfe eines Event-Objekts (hier: `e`). Bei einem Tastaturereignis sind die folgenden Objekteigenschaften relevant:
    - `char` (Zeichen)
    - `keysym` (Beschreibung des Zeichens)
    - `keycode` (Code des Zeichens)

Diese Objekteigenschaften werden zusätzlich angezeigt (siehe Abbildung 11.25).

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

### Unterschiede unter Ubuntu Linux

Nur das Betätigen der Buchstabentasten führt zu einem korrekt erkannten Tastaturereignis. Das Betätigen der Zahlentasten wird ignoriert. Die Bindung von Umlaut- und Sonderzeichentasten an ein Ereignis führt zu Fehlern.

### Unterschiede unter OS X

Die verschiedenen Tasten werden richtig erkannt, wie unter Windows. Der Code der verschiedenen Zeichen unterscheidet sich allerdings von den Codes unter Windows.

## 11.3 Geometrische Anordnung von Widgets

In den bisherigen Programmen wurde die Methode `pack()` als einfacher Geometriemanager zur Anordnung und Anzeige der Widgets innerhalb des Fensters genutzt. Geometriemanager bestimmen das Layout einer GUI-Anwendung, sie werden daher auch Layoutmanager genannt. Im Folgenden gehe ich auf die Möglichkeiten zur Anordnung von Widgets ein:

- ▶ `pack()`
- ▶ `place()`
- ▶ `grid()`

Außerdem beschreibe ich das Frame-Widget, mit dessen Hilfe Sie ein Fenster in Teile zerlegen. Innerhalb dieser Teile können Sie wiederum Widgets passend anordnen. Dies ermöglicht auch den Einsatz von verschiedenen Geometriemanagern innerhalb eines Programms.

Manager

Frame-Widget

### Hinweise

1. Die Widgets in den nachfolgend aufgeführten Programmen führen nicht immer zu Aktionen. Sie werden an einigen Stellen nur dargestellt, um die verschiedenen Methoden zu verdeutlichen.
2. In den Beispielen werden nur die Widget-Typen *Button* und *Label* verwendet. Natürlich können Sie alle Typen von Widgets auf die vorgestellte Art und Weise anordnen.

### 11.3.1 Frame-Widget, Methode »pack()«

Bisher wurden alle Widgets innerhalb des Hauptfensters angeordnet. Das unmittelbare *Elternelement*, also das übergeordnete Element, ist in diesem Fall das Hauptfenster.

Elternelement

Frame-WIDGETS werden allerdings häufig auch als Elternelemente für andere Widgets eingesetzt. Dies ermöglicht die Gruppierung von Widgets und vielfältigere Möglichkeiten der Anordnung.

Gruppierung

Das folgende Beispiel enthält drei Frames sowie mehrere Button-Widgets, die teilweise innerhalb, teilweise außerhalb der Frames angeordnet wurden. Dabei werden verschiedene Parameter der Methode `pack()` angewendet.

```
import tkinter

def ende():
    main.destroy()

main = tkinter.Tk()

# Frame 1 mit Button 1a und 1b
fr1 = tkinter.Frame(main, width=200, height=100,
                     relief="sunken", bd=1)
fr1.pack(side="left")
b1a = tkinter.Button(fr1, text="Button 1a")
b1a.pack(padx=5, pady=5)
b1b = tkinter.Button(fr1, text="Button 1b")
b1b.pack(padx=5, pady=5)

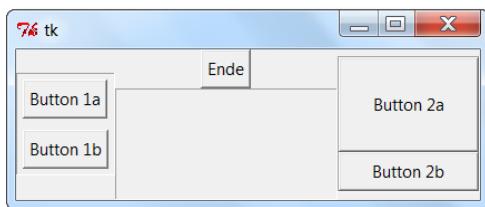
# Frame 2 mit Button 2a und 2b
fr2 = tkinter.Frame(main, width=200, height=100,
                     relief="sunken", bd=1)
fr2.pack(side="right")
b2a = tkinter.Button(fr2, text="Button 2a")
b2a.pack(ipadx=25, ipady=25)
b2b = tkinter.Button(fr2, text="Button 2b")
b2b.pack(fill="x")
# Frame 3
fr3 = tkinter.Frame(main, width=200, height=100,
                     relief="sunken", bd=1)
fr3.pack(side="bottom", expand=1, fill="both")

bende = tkinter.Button(main, text = "Ende",
                       command = ende)
bende.pack()

main.mainloop()
```

**Listing 11.17** Datei `gui_pack.py`

Die Darstellung sehen Sie in [Abbildung 11.26](#).



**Abbildung 11.26** Geometriemanager »pack«

Zur Erläuterung:

- ▶ Das Fenster ist zunächst in drei Frames eingeteilt: `fr1`, `fr2` und `fr3`. Für alle drei Frames wurden die Randart `sunken` und die Randbreite 1 gewählt, damit sie optisch erkennbar sind. Als Startgröße wurden jeweils eine Breite von 200 Pixeln und eine Höhe von 100 Pixeln vergeben. Allerdings werden diese Größen von den eingelagerten Widgets beeinflusst.
- ▶ Der erste Frame (`fr1`) wurde an der linken Seite des Hauptfensters angeordnet. Dies wird mit dem Parameter `side` und dem Wert `left` bei der Methode `pack()` erreicht. Der zweite Frame (`fr2`) und der dritte Frame (`fr3`) wurden an der rechten und an der unteren Seite des Hauptfensters angeordnet (`side= "right"` und `side= "bottom"`). side
- ▶ Beim dritten Frame wurden außerdem die Parameter `expand` (Wert 1) und `fill` (Wert `both`) eingestellt. Dies führt dazu, dass dieser Frame den restlichen verfügbaren Platz im Hauptfenster nutzt, auch bei einer Größenveränderung des Hauptfensters. expand, fill
- ▶ Der ENDE-Button wurde gar nicht angeordnet. Alle nicht angeordneten Elemente werden, wie bei den bisherigen Programmen, innerhalb ihrer jeweiligen Umgebung (Frame oder Hauptfenster) der Reihe nach von oben nach unten dargestellt.
- ▶ Bei den beiden Buttons `b1a` und `b1b` wurde als Elternelement nicht das Hauptfenster, sondern der erste Frame (`fr1`) angegeben. Daher werden diese Buttons innerhalb dieses Frames angeordnet. Elternelement
- ▶ Die Parameter `padx` und `pady` führen dazu, dass ein Abstand (hier: jeweils 5 Pixel in x-Richtung und in y-Richtung) zum Nachbar-Widget bzw. zum Rand des Elternelements eingehalten wird. padx, pady
- ▶ Die beiden Buttons `b2a` und `b2b` werden innerhalb des Frames `fr2` angeordnet.
- ▶ Die Parameter `ipadx` und `ipady` beim Button `b2a` führen dazu, dass dieses Widget vergrößert wird (hier: jeweils 5 Pixel in x-Richtung und in y-Richtung). ipadx, ipady
- ▶ Der Parameter `fill` mit dem Wert `x` beim Button `b2b` führt dazu, dass dieses Widget den restlichen verfügbaren Platz in x-Richtung innerhalb des Elternelements ausfüllt. fill

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

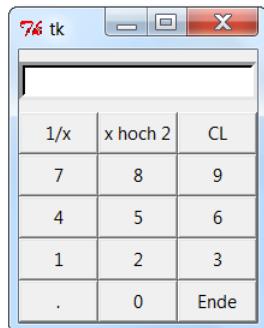
**Maßeinheiten****Hinweise**

- Notieren Sie die Angaben für die Abstände zum Nachbar-Widget (`padx` und `pady`) und für die erweiterte Ausdehnung eines Widgets (`ipadx` und `ipady`) als einfache Zahlen, so handelt es sich automatisch um die Einheit Pixel.
- Notieren Sie die Angaben als Zeichenkette, und versehen Sie sie mit einem Suffix, so sind auch andere Einheiten möglich. Als Suffixe erlaubt sind `c` (für Zentimeter), `i` (für Inches), `m` (für Millimeter) und `p` (für Punkte; ein Punkt entspricht circa 1/72 Inch).

**11.3.2 Ein einfacher Taschenrechner**

Als Anwendung für eine Anordnung mit mehreren Frames stelle ich im Folgenden einen einfachen Taschenrechner vor. Die Darstellung des Taschenrechners zeigt [Abbildung 11.27](#).

Der Benutzer kann durch Betätigung der Zifferntasten 0 bis 9 und der Taste für den Dezimalpunkt beliebige Zahlen zusammenstellen, die anschließend in der Anzeige erscheinen.



**Abbildung 11.27** Taschenrechner

**Tasten** Außerdem gibt es die folgenden Tasten:

- ▶ Taste 1/x: Berechnung und Anzeige des Kehrwerts der Zahl
- ▶ Taste X HOCH 2: Berechnung und Anzeige des Quadrats der Zahl
- ▶ Taste CL: Löschen der Anzeige
- ▶ Taste ENDE: Beenden des Programms

Das Programm:

```

import tkinter

def ende():
    main.destroy()

# Kehrwert
def kw():
    if not lb["text"]:
        return
    z = float(lb["text"])
    z = 1/z
    lb["text"] = str(z)

# Quadrat
def qu():
    if not lb["text"]:
        return
    z = float(lb["text"])
    z = z*z
    lb["text"] = str(z)

# Anzeige leeren
def cl():
    lb["text"] = ""

# Ziffern 0 bis 9
def anz(ziffer):
    lb["text"] += str(ziffer)

# Punkt, falls noch nicht vorhanden
def anzp():
    if lb["text"].find(".") == -1:
        lb["text"] += "."

main = tkinter.Tk()

# Frame A mit Anzeigelabel
fra = tkinter.Frame(main)
fra.pack(expand=1, fill="x")
lb = tkinter.Label(fra, bg="#FFFFFF", bd=5,
                   relief="sunken", anchor="e")
lb.pack(expand=1, fill="x", pady=10)
# Frame B mit Kehrwert, Quadrat und Leeren
frb = tkinter.Frame(main)
frb.pack()
tkinter.Button(frb, text="1/x", width=7,
               command=kw).pack(side="left")

```

Kehrwert

Quadrat

Zifferntasten

Dezimalpunkt

Hauptprogramm

Anordnung

```

tkinter.Button(frb, text="x hoch 2", width=7,
              command=qu).pack(side="left")
tkinter.Button(frb, text="CL", width=7,
              command=cl).pack(side="left")

# Frame C mit Ziffern 7, 8, 9
frc = tkinter.Frame(main)
frc.pack()
tkinter.Button(frc, text="7", width=7,
               command=lambda:anz(7)).pack(side="left")
tkinter.Button(frc, text="8", width=7,
               command=lambda:anz(8)).pack(side="left")
tkinter.Button(frc, text="9", width=7,
               command=lambda:anz(9)).pack(side="left")

# Frame D mit Ziffern 4, 5, 6
frd = tkinter.Frame(main)
frd.pack()
tkinter.Button(frd, text="4", width=7,
               command=lambda:anz(4)).pack(side="left")
tkinter.Button(frd, text="5", width=7,
               command=lambda:anz(5)).pack(side="left")
tkinter.Button(frd, text="6", width=7,
               command=lambda:anz(6)).pack(side="left")

# Frame E mit Ziffern 1, 2, 3
fre = tkinter.Frame(main)
fre.pack()
tkinter.Button(fre, text="1", width=7,
               command=lambda:anz(1)).pack(side="left")
tkinter.Button(fre, text="2", width=7,
               command=lambda:anz(2)).pack(side="left")
tkinter.Button(fre, text="3", width=7,
               command=lambda:anz(3)).pack(side="left")

# Frame F mit Dezimalpunkt, Ziffer 0 und Ende
frf = tkinter.Frame(main)
frf.pack()
tkinter.Button(frf, text=". ", width=7,
               command=anzp).pack(side="left")
tkinter.Button(frf, text="0", width=7,
               command=anz0).pack(side="left")
tkinter.Button(frf, text="Ende", width=7,
               command=ende).pack(side="left")

main.mainloop()

```

**Listing 11.18** Datei gui\_rechner\_pack.py

Erläuterung der Funktionen:

Funktionen

- ▶ In den Funktionen `kw()` und `qu()` zur Berechnung von Kehrwert und Quadrat der angezeigten Zahl wird zunächst überprüft, ob überhaupt eine Zahl zur Berechnung in der Anzeige steht. Ist dies der Fall, so wird die entsprechende Berechnung durchgeführt, und das Ergebnis wird angezeigt.
- ▶ Die Funktion `cl()` zum Löschen setzt den Anzeigetext auf eine leere Zeichenkette.
- ▶ Die Funktion `anz()` fügt die jeweilige Ziffer zur Anzeige hinzu. Die Funktion `anzp()` fügt einen Dezimalpunkt hinzu, falls bisher noch kein Dezimalpunkt vorhanden ist. Durch diese Funktionen können beliebige Zahlen, auch mit Nachkommastellen, zusammengesetzt werden.

### Hinweis

Das Hinzufügen von Ziffern ist natürlich nur dann sinnvoll, wenn nicht unmittelbar zuvor ein Kehrwert oder ein Quadrat berechnet wurde. In diesem Fall sollte vor der Eingabe einer neuen Ziffer zunächst die Anzeige gelöscht werden.

Erläuterung der geometrischen Anordnung:

Geometrische Anordnung

- ▶ Das Fenster ist in sechs Frames (Frame A bis F) unterteilt, die jeweils eine waagrechte Reihe bilden.
- ▶ Innerhalb des obersten Frames (Frame A) steht das Label-Widget für die Zahlenanzeige. Das Label-Widget nimmt die gesamte Breite des Fensters ein und ist optisch vom Rest des Fensters abgesetzt. Die Inhalte des Labels werden rechtsbündig angezeigt (`anchor="e"`), wie üblich bei Zahlen.
- ▶ Die restlichen Frames (Frame B bis F) enthalten je drei Buttons, die immer an der linken Seite des Frames bzw. des Nachbar-Widgets angesetzt werden (`side="left"`). Diese insgesamt 15 Buttons führen jeweils zu einer der 15 oben genannten Funktionen.
- ▶ Da es sich um eine regelmäßige Anordnung handelt, können wir die Zahlenbuttons in Schleifen erzeugen. Der Text auf dem Button ist der aktuelle Wert der Schleifenvariablen, mit der Funktion `str()` in Text konvertiert.
- ▶ Nach dem Parameter `command` kann nur der Name einer Funktion angegeben werden, jedoch kein Parameter für diese Funktion. Dieses Problem lösen wir durch Angabe einer Lambda-Funktion.

Lambda

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

### 11.3.3 Methode »grid()«

**Raster** Die Methode `grid()` dient als Geometriemanager zur Erzeugung von besonders regelmäßigen Anordnungen, die sich in einem Raster darstellen lassen. Ähnlich wie in einer Tabelle, die von einem Textverarbeitungsprogramm oder in einem HTML-Dokument erzeugt wird, werden die Widgets in Tabellenzellen angeordnet.

**Zeile, Spalte** Jede Tabellenzelle ist durch die Nummer der Zeile und die Nummer der Spalte gekennzeichnet. Jedem Widget werden bei Aufruf der Methode `grid()` die betreffenden Nummern (für die Eigenschaften `row` bzw. `column`) mitgegeben, jeweils beginnend bei 0.

**Zellen verbinden** Es ist auch möglich, ein Widget über mehrere Tabellenzellen waagrecht und/oder senkrecht auszudehnen. Dazu geben Sie die Spannweite (für die Eigenschaften `rowspan` und `columnspan`) an.

Das Taschenrechnerprogramm aus dem vorherigen Abschnitt wurde im Folgenden verändert. Die geometrische Anordnung der Buttons und des Labels erfolgt durch die Methode `grid()`.

Im Folgenden wird nur der Teil des Programms aufgeführt, in dem die Widgets erzeugt werden. Die Funktionen bleiben unverändert, ebenso die Darstellung.

```
. . . . .
# Row 0
lb = tkinter.Label(main, bg="#FFFFFF", bd=5, relief="sunken", anchor="e")
lb.grid(row=0, column=0, columnspan=3, sticky="we")

# Row 1
tkinter.Button(main, text="1/x", width=7,
               command=kw).grid(row=1, column=0)
tkinter.Button(main, text="x hoch 2", width=7,
               command=qu).grid(row=1, column=1)
tkinter.Button(main, text="CL", width=7,
               command=cl).grid(row=1, column=2)

# Row 2
tkinter.Button(main, text="7", width=7, command=
               lambda:anz(7)).grid(row=2, column=0)
tkinter.Button(main, text="8", width=7, command=
               lambda:anz(8)).grid(row=2, column=1)
tkinter.Button(main, text="9", width=7, command=
               lambda:anz(9)).grid(row=2, column=2)

# Row 3
tkinter.Button(main, text="4", width=7, command=
               lambda:anz(4)).grid(row=3, column=0)
tkinter.Button(main, text="5", width=7, command=
               lambda:anz(5)).grid(row=3, column=1)
```

```

tkinter.Button(main, text="6", width=7, command=
    lambda:anz(6)).grid(row=3, column=2)

# Row 4
tkinter.Button(main, text="1", width=7, command=
    lambda:anz(1)).grid(row=4, column=0)
tkinter.Button(main, text="2", width=7, command=
    lambda:anz(2)).grid(row=4, column=1)
tkinter.Button(main, text="3", width=7, command=
    lambda:anz(3)).grid(row=4, column=2)

# Row 5
tkinter.Button(main, text=". ", width=7,
    command=anzp).grid(row=5, column=0)
tkinter.Button(main, text="0", width=7,
    command=anz0).grid(row=5, column=1)
tkinter.Button(main, text="Ende", width=7,
    command=ende).grid(row=5, column=2)
. . .

```

**Listing 11.19** Datei `gui_rechner_grid.py`, Ausschnitt

Zur Erläuterung:

- ▶ Es sind keine Frames mehr notwendig. Ihre Aufgabe wurde von den einzelnen Zeilen (`rows`) der Tabelle übernommen. Tabellenzeilen
- ▶ In der ersten Zeile steht nur das Label mit den folgenden Eigenschaften:
  - `row = 0`, das Widget beginnt in Zeile 0 des Hauptfensters.
  - `column = 0`, das Widget beginnt in Spalte 0 des Hauptfensters.
  - `columnspan = 3`, das Widget erstreckt sich über drei Spalten.
  - `sticky = "we"`, das Widget wird zum westlichen (linken) und zum östlichen (rechten) Rand des Elternelements ausgedehnt. Mögliche Werte für `sticky` sind: `w` (westlich, links), `e` (östlich, rechts), `n` (nördlich, oben) und `s` (südlich, unten). Geben Sie diese Eigenschaft nicht an, so wird das Widget in der Mitte der Tabellenzelle(n) dargestellt.
- ▶ Für die insgesamt 15 Buttons sind jeweils die Nummer der Zeile (`row=1` bis `row=5`) und die Nummer der Spalte (`column=0` bis `column=2`) angegeben. Mehr ist zur Anordnung nicht notwendig.
- ▶ Bei den Zahlenbuttons wird wiederum mit einer Lambda-Funktion gearbeitet. Lambda

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

**Hinweis**

Bei der Methode `grid()` können Sie wie bei der Methode `pack()` zusätzlich die Eigenschaften `padx`, `pady`, `ipadx` und `ipady` angeben.

### 11.3.4 Methode »place()«, absolute Koordinaten

Die Methode `place()` ermöglicht die Positionierung von Widgets an absoluten oder an relativen Koordinaten.

**Absolute Koordinaten bleiben konstant**

Absolute Koordinaten haben den Nachteil, dass sie bei einer Größenänderung des Fensters konstant bleiben. Eine Anordnung mit absoluten Koordinaten zeigt das folgende Beispiel:

```
import tkinter

def ende():
    main.destroy()

main = tkinter.Tk()

b1 = tkinter.Button(main, text="b1", command=ende)
b1.place(x=50, y=50, anchor="se")

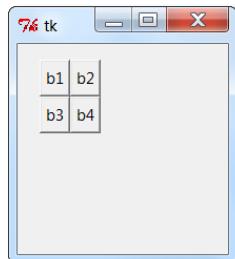
b2 = tkinter.Button(main, text="b2", command=ende)
b2.place(x=50, y=50, anchor="sw")

b3 = tkinter.Button(main, text="b3", command=ende)
b3.place(x=50, y=50, anchor="ne")
b4 = tkinter.Button(main, text="b4", command=ende)
b4.place(x=50, y=50, anchor="nw")

main.mainloop()
```

**Listing 11.20** Datei `gui_koord_absolut.py`

Das Fenster sieht aus wie in [Abbildung 11.28](#).



**Abbildung 11.28** Geometriemanager »place()«, absolute Koordinaten

Die Koordinatenangabe setzt sich aus drei Bestandteilen zusammen:

- ▶ x-Koordinate: Wird in Pixel oder anderen Einheiten angegeben (*c* für Zentimeter, *i* für Inches, *m* für Millimeter und *p* für Punkte), gemessen vom linken Rand des Elternelements.
- ▶ y-Koordinate: Wird in Pixel angegeben (oder anderen Einheiten, siehe x-Koordinate), gemessen vom oberen Rand des Elternelements.
- ▶ Ankerpunkt (anchor): Diese Stelle des Widgets liegt an den angegebenen Koordinaten des Elternelements. Mögliche Angaben sind:
  - se (Süd-Ost, rechte untere Ecke)
  - sw (Süd-West, linke untere Ecke)
  - ne (Nord-Ost, rechte obere Ecke)
  - nw (Nord-West, linke obere Ecke)
  - s (Süd, Mitte der unteren Seite)
  - n (Nord, Mitte der oberen Seite)
  - w (West, Mitte der linken Seite)
  - e (Ost, Mitte der rechten Seite)
- ▶ Alle vier Buttons sind an den gleichen Koordinaten ( $x=50, y=50$ ) angeordnet. Allerdings beziehen sich die Koordinaten auf jeweils andere Ecken des Widgets, daher überlappen sich die vier Buttons nicht.
  - Die rechte untere Ecke des Buttons *b1* liegt bei  $x=50, y=50$ .
  - Die linke untere Ecke des Buttons *b2* liegt bei  $x=50, y=50$ .
  - Die rechte obere Ecke des Buttons *b3* liegt bei  $x=50, y=50$ .
  - Die linke obere Ecke des Buttons *b4* liegt bei  $x=50, y=50$ .

### Unterschiede in Python 2

Das Modul heißt Tkinter statt tkinter.

#### 11.3.5 Methode »place()«, relative Koordinaten

Relative Koordinaten beziehen sich auf Anteile der Breite oder Höhe des Elternelements. Im folgenden Beispiel befinden sich fünf Buttons – relativ zum Elternelement – immer an der gleichen Stelle.

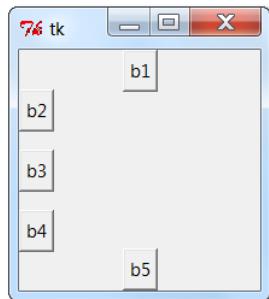
```
import tkinter
def ende():
    main.destroy()
main = tkinter.Tk()
b1 = tkinter.Button(main, text="b1", command=ende)
b1.place(relx=0.5, rely=0, anchor="n")
b2 = tkinter.Button(main, text="b2", command=ende)
b2.place(relx=0, rely=0.25, anchor="w")
```

Relative Koordinaten ändern sich mit Elternelement

```
b3 = tkinter.Button(main, text="b3", command=ende)
b3.place(relx=0, rely=0.5, anchor="w")
b4 = tkinter.Button(main, text="b4", command=ende)
b4.place(relx=0, rely=0.75, anchor="w")
b5 = tkinter.Button(main, text="b5", command=ende)
b5.place(relx=0.5, rely=1, anchor="s")
main.mainloop()
```

**Listing 11.21** Datei `gui_koord_relativ.py`

Das Fenster sieht aus wie in [Abbildung 11.29](#).

**Abbildung 11.29** Geometriemanager »place«, relative Koordinaten

Zur Erläuterung:

- ▶ Die Koordinatenangabe setzt sich auch hier aus drei Bestandteilen zusammen:

- Werte von 0 bis 1

**relx, rely**

- Relative x-Koordinate (`relx`): ein Wert zwischen 0 und 1. Der Wert 0 steht für den linken Rand des Elternelements, der Wert 1 für den rechten Rand.
  - Relative y-Koordinate (`rely`): ebenfalls ein Wert zwischen 0 und 1. Der Wert 0 steht für den oberen Rand des Elternelements, der Wert 1 für den unteren Rand.

**anchor**

- Ankerpunkt (`anchor`): wie bei den absoluten Koordinaten

- ▶ Im vorliegenden Beispiel sind die fünf Buttons wie folgt angeordnet:

- Die Mitte der oberen Seite (`anchor="n"`) von Button `b1` liegt auf der Mitte der oberen Seite (`relx=0.5, rely=0`) des Elternelements.
  - Die Mitte der linken Seite (`anchor="w"`) der Buttons `b2`, `b3` und `b4` liegt am linken Rand (`relx=0`) des Elternelements. Die Höhe ist jeweils unterschiedlich: Button `b2` liegt auf 25 % der Höhe des Elternelements (`rely=0.25`), Button `b3` liegt auf 50 % der Höhe des Elternelements (`rely=0.5`), Button `b4` liegt auf 75 % der Höhe des Elternelements (`rely=0.75`).
  - Die Mitte der unteren Seite (`anchor="s"`) von Button `b5` liegt auf der Mitte der unteren Seite (`relx=0.5, rely=1`) des Elternelements.

## Unterschiede in Python 2

Das Modul heißt Tkinter statt tkinter.

### 11.3.6 Absolute Veränderung von Koordinaten

Sie können die verschiedenen Geometriemanager auch mehrmals auf die Widgets anwenden. Dies ermöglicht die Programmierung von dynamisch veränderlichen Anordnungen, also von beweglichen Widgets.

Bewegliche Widgets

Das folgende Beispiel soll dies anhand eines Labels verdeutlichen, das mithilfe von zwei Buttons an zwei verschiedene Positionen bewegt werden kann.

```
import tkinter

def ende():
    main.destroy()

# bewegt nach ganz links
def movetoleft():
    lb.place(relx=0, rely=0, anchor="nw")

# bewegt nach ganz rechts
def moveright():
    lb.place(relx=1, rely=0, anchor="ne")

main = tkinter.Tk()

# Bewegtes Label
lb = tkinter.Label(main, text="Test",
                    relief="sunken", bd=1)
lb.place(relx=0, rely=0, anchor="nw")

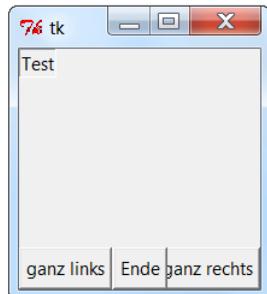
# bewegt nach ganz links
bleft = tkinter.Button(main, text="ganz links",
                        command=movetoleft)
bleft.place(relx=0, rely=1, anchor="sw")

# bewegt nach ganz rechts
bright = tkinter.Button(main, text="ganz rechts",
                        command=moveright)
bright.place(relx=1, rely=1, anchor="se")
bende = tkinter.Button(main, text="Ende", command=ende)
bende.place(relx=0.5, rely=1, anchor="s")

main.mainloop()
```

**Listing 11.22** Datei gui\_aendern\_absolut.py

Das Fenster sieht nach dem Start aus wie in [Abbildung 11.30](#) dargestellt.

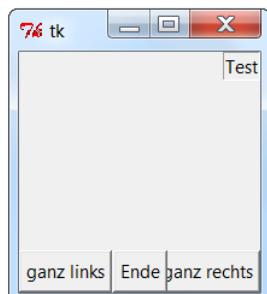


**Abbildung 11.30** Absolute Koordinatenveränderung, Startzustand

Nach der Betätigung des Buttons GANZ RECHTS sieht das Fenster aus wie in [Abbildung 11.31](#).

Zur Erläuterung:

- ▶ Das Label wird insgesamt dreimal platziert:
  - nach dem Start des Programms in die obere linke Ecke des Elternelements
  - nach Betätigung des Buttons GANZ RECHTS in die obere rechte Ecke des Elternelements, in der Funktion `movetoright()`
  - nach Betätigung des Buttons GANZ LINKS in die obere linke Ecke des Elternelements, in der Funktion `movetoleft()`



**Abbildung 11.31** Absolute Koordinatenveränderung, nach Aktion

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

### 11.3.7 Relative Veränderung von Koordinaten

Im vorhergehenden Beispiel haben wir zwar mit relativen Widget-Koordinaten gearbeitet, es handelt sich aber um eine absolute Koordinatenveränderung. Das Label bewegt sich nur zwischen zwei bestimmten Punkten hin und her.

Sie können ein Widget aber auch relativ bewegen, also ausgehend von den aktuellen Koordinaten, um einen bestimmten Wert in eine ausgewählte Richtung. Im folgenden Beispiel kann ein Label mithilfe von vier Buttons in vier verschiedene Richtungen bewegt werden.

```
import tkinter

def ende():
    main.destroy()

# Aktuelle Position
posx = 0
posy = 0

# bewegt nach links
def moveleft():
    global posx, posy
    posx -= 20
    lb.place(x=posx, y=posy, anchor="nw")

# bewegt nach rechts
def moveright():
    global posx, posy
    posx += 20
    lb.place(x=posx, y=posy, anchor="nw")

# bewegt nach oben
def moveup():
    global posx, posy
    posy -= 20
    lb.place(x=posx, y=posy, anchor="nw")

# bewegt nach unten
def movedown():
    global posx, posy
    posy += 20
    lb.place(x=posx, y=posy, anchor="nw")

# bewegt nach unten
def movestart():
    global posx, posy
    posx = 0
    posy = 0
    lb.place(x=posx, y=posy, anchor="nw")
```

```

main = tkinter.Tk()

# Frame mit bewegtem Label
fr1 = tkinter.Frame(main, width=200, height=150,
                     relief="sunken", bd=1)
fr1.pack(side="top")

# Frame mit Buttons zur Bewegung
fr2 = tkinter.Frame(main, height=80)
fr2.pack(side="left")

# Frame mit Ende-Button
fr3 = tkinter.Frame(main, width=50, height=80)
fr3.pack(side="left")

# Bewegtes Label
lb = tkinter.Label(fr1, text="Test", relief="sunken", bd=1)
lb.place(x=0, y=0, anchor="nw")

# bewegt nach links
bleft = tkinter.Button(fr2, text="nach links",
                       command=moveleft)
bleft.grid(row=1, column=0)

# bewegt nach rechts
bright = tkinter.Button(fr2, text="nach rechts",
                        command=moveright)
bright.grid(row=1, column=2)

# bewegt nach oben
bup = tkinter.Button(fr2, text="nach oben", command=moveup)
bup.grid(row=0, column=1)
# bewegt nach unten
bdown = tkinter.Button(fr2, text="nach unten",
                       command=movedown)
bdown.grid(row=2, column=1)
# Startposition
bstart = tkinter.Button(fr2, text="Start",
                        command=movestart)
bstart.grid(row=1, column=1)

# Ende-Button
bende = tkinter.Button(fr3, text="Ende",
                       command=ende)
bende.place(relx=1, rely=1, anchor="se")

main.mainloop()

```

Listing 11.23 Datei gui\_aendern\_relativ.py

Das Fenster stellt sich nach dem Start dar wie in Abbildung 11.32.

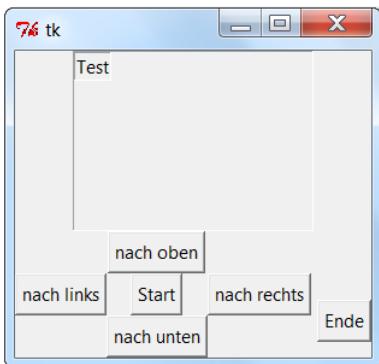


Abbildung 11.32 Relative Koordinatenveränderung, Startzustand

Nach mehrfacher Betätigung der Buttons NACH RECHTS und NACH UNTER sieht das Fenster wie in Abbildung 11.33 gezeigt aus.

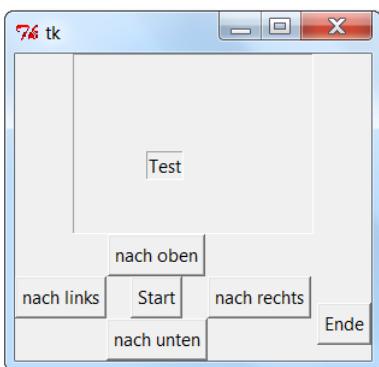


Abbildung 11.33 Relative Koordinatenveränderung, nach mehrfacher Aktion

Erläuterung der Funktionen:

- ▶ Zunächst wird das Label an der absoluten Position  $x=0, y=0$  platziert. Diese Position wird gleichzeitig den beiden Variablen posx und posy zugewiesen.
- ▶ In den vier Funktionen `moveleft()`, `moveright()`, `moveup()` und `movedown()` werden jeweils zunächst die Variablen posx und posy als globale Variablen bekannt gemacht. Dies bedeutet, dass es sich sowohl außerhalb als auch innerhalb dieser vier Funktionen um dieselben Variablen handelt.
- ▶ Die Werte von posx und posy werden anschließend in den vier Funktionen jeweils passend verändert und dazu genutzt, eine neue Platzierung mit der Methode `place()` durchzuführen. In den beiden Funktionen `moveleft()` und `moveright()` wird die x-Koordinate um 20 Pixel verändert. In den beiden

#### Globale Variablen

#### Relative Platzierung

Funktionen `moveup()` und `movedown()` wird die y-Koordinate um 20 Pixel verändert.

- ▶ Es gibt insgesamt drei Frames:
  - Der erste Frame ist am oberen Fensterrand angesetzt und enthält das Label. Das Label soll sich relativ zum ersten Frame bewegen.
  - Der zweite Frame ist am linken Fensterrand angesetzt. Er beherbergt die vier Buttons zum Bewegen des Labels. Sie sind mithilfe der Methode `grid()` in einem Raster angeordnet, um die möglichen Bewegungsrichtungen zu verdeutlichen.
  - Der dritte Frame ist ebenfalls nach links orientiert und erscheint daher neben dem zweiten Frame. Er enthält nur den ENDE-Button.

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

#### Hinweis

Es ist möglich, das Label über den Fensterrand hinaus zu bewegen. Durch zusätzlichen Programmieraufwand könnten Sie dies verhindern – um das Programm überschaubar zu halten, habe ich hier darauf verzichtet.

## 11.4 Menüs, Messageboxen und Dialogfelder

Viele Benutzeroberflächen verfügen zur Bedienung neben den Steuerelementen innerhalb des Dialogfelds (= Fenster) zusätzlich über

- Menü**
- ▶ eine Menüleiste am oberen Rand des Fensters, die permanent sichtbar ist, sowie

- Kontextmenü**
- ▶ Kontextmenüs, die einzelnen Steuerelementen zugeordnet sind und nur bei Bedarf eingeblendet werden.

- Menu-Widget**
- In Python stehen zu diesem Zweck Menu-Widgets zur Verfügung. Nachfolgend stelle ich die beiden genannten Menüarten vor. Außerdem behandle ich in diesem Abschnitt vorgefertigte Dialogfelder (Messageboxen) und eigene Dialogfelder.

### 11.4.1 Menüleisten

Die Erzeugung einer Menüleiste mit einzelnen Menüs und darin enthaltenen Menübefehlen erfordert grundsätzlich die folgenden Schritte:

- ▶ Erzeugung eines Objekts der Klasse `Menu` als Menüleiste
- ▶ Erzeugung von weiteren Objekten der Klasse `Menu` als einzelne Menüs innerhalb der Menüleiste
- ▶ Erzeugung von einzelnen Menüpunkten verschiedenen Typs innerhalb eines Menüs
- ▶ Hinzufügen der Menüs zur Menüleiste und Bezeichnen der Menüs
- ▶ Hinzufügen der Menüleiste zum Fenster

Reihenfolge der Erstellung

Das folgende Programm enthält eine Menüleiste mit zwei Menüs. Die beiden Menüs verfügen über vier bzw. über fünf Menüpunkte verschiedenen Typs:

```
import tkinter

def ende():
    main.destroy()

def farbwechsel():
    fr["bg"] = farbe.get()

def randwechsel():
    if rand.get():
        fr["relief"] = "ridge"
    else:
        fr["relief"] = "flat"

main = tkinter.Tk()

# Zielobjekt der Menuebefehle
fr = tkinter.Frame(main, height=100, width=300,
                   bg="#FFFFFF", bd=10)
fr.pack()

# erzeugt gesamte Menueleiste
mBar = tkinter.Menu(main)

# erzeugt erstes Menueobjekt der Menueleiste
mFile = tkinter.Menu(mBar)

# erzeugt Elemente in erstem Menue
mFile.add_command(label="Neu")
mFile.add_command(label="Laden")
mFile.add_command(label="Speichern")
mFile.add_separator()
mFile.add_command(label="Beenden", command=ende)

# Widget-Variablen der Radiobutton-Menüpunkte
# bzw. Checkbutton-Menüpunkte
farbe = tkinter.StringVar()
```

```

farbe.set("#FFFFFF")
rand = tkinter.IntVar()
rand.set(0)

# erzeugt zweites Menueobjekt der Menueleiste
mView = tkinter.Menu(mBar)
mView["tearoff"] = 0      # Menue nicht abtrennbar

# erzeugt Elemente in zweitem Menue
mView.add_radiobutton(label="Rot", variable=farbe,
                      value="#FF0000", underline=0, command=farbwechsel)
mView.add_radiobutton(label="Gelb", variable=farbe,
                      value="#FFFF00", underline=0, command=farbwechsel)
mView.add_radiobutton(label="Blau", variable=farbe,
                      value="#0000FF", underline=0, command=farbwechsel)
mView.add_radiobutton(label="Magenta", variable=farbe,
                      value="#FF00FF", underline=0, command=farbwechsel)
mView.add_separator()
mView.add_checkbutton(label="Rand sichtbar",
                      variable=rand, onvalue=1, offvalue=0, underline=5,
                      command=randwechsel)
# erstes und zweites Menue zur Menueleiste hinzufügen
mBar.add_cascade(label="Datei", menu=mFile)
mBar.add_cascade(label="Ansicht", menu=mView)

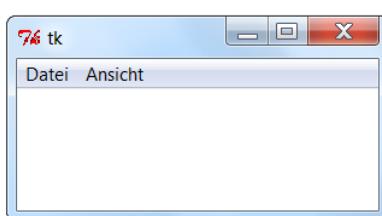
# gesamte Menueleiste zu Fenster hinzufügen
main["menu"] = mBar

main.mainloop()

```

**Listing 11.24** Datei gui\_menu.py

- Startzustand** Nach dem Start sind nur die beiden Einträge in der Menüleiste sichtbar sowie der weiße Frame ohne Rand wie in Abbildung 11.34.

**Abbildung 11.34** Hauptfenster mit Menü

- Menü »Datei«** Nach der Anwahl des Menüs DATEI klappt dieses mit seinen Menüpunkten auf wie in Abbildung 11.35.

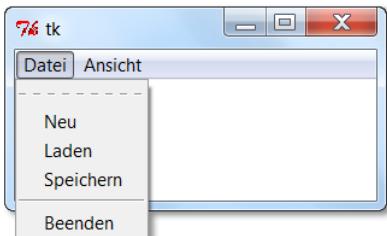


Abbildung 11.35 Menü »Datei«

Nur der Menüpunkt BEENDEN ist mit einer Funktion belegt (Beenden des Programms), die anderen Menüpunkte dienen ausschließlich der Darstellung. Nach der Anwahl des Menüs ANSICHT klappt dieses mit seinen Menüpunkten auf wie in [Abbildung 11.36](#).

Alle Menüpunkte sind mit einer Funktion belegt. Die Bedienung eines der ersten vier Menüpunkte sorgt dafür, dass die Farbe des dargestellten Frames wechselt. Gleichzeitig wird der betreffende Menüpunkt markiert und somit die aktuelle Farbe gekennzeichnet. Der letzte Menüpunkt dient zur Darstellung des Frames mit oder ohne Rand.

Einstellung  
der Farbe

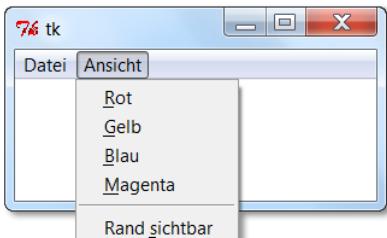


Abbildung 11.36 Menü »Ansicht«, Startzustand

Nach Anwählen von BLAU und des Menüpunkts RAND SICHTBAR sieht die Oberfläche inklusive Menü aus, wie in [Abbildung 11.37](#) dargestellt.

Nach Bedienung

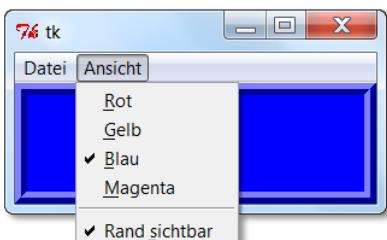


Abbildung 11.37 Hauptfenster und Menü »Ansicht«, nach Änderung

<b>Tear-off</b>	Oberhalb jedes Menüs wird normalerweise eine gestrichelte Linie angezeigt. Ein Klick auf diese Linie bewirkt die Auslagerung ( <i>Tear-off</i> ) des betreffenden Menüs als eigenständiges Fenster. Dies ist eine Standardeigenschaft von Python-GUI-Anwendungen und muss nicht gesondert programmiert werden. Beim zweiten Menü (in Abbildung 11.36) wurde diese Möglichkeit unterdrückt. Zur Erläuterung des Programms:
	► Es wird ein Frame der Höhe 100 Pixel und der Breite 300 Pixel erzeugt. Er dient als Zielobjekt für die Einstellung von Farbe und Rand. Zu Beginn werden die Farbe Weiß und die Randbreite 10 eingestellt. Da keine Randart gewählt wird, ist der Rand noch nicht sichtbar.
<b>Menu-Objekt</b>	► Es wird ein Objekt der Klasse <code>Menu (mBar)</code> als gesamte Menüleiste für das Fenster erzeugt.
<b>add_command()</b>	► Es wird das Menü DATEI ( <code>mFile</code> ) innerhalb der Menüleiste erzeugt.
	► Die Methode <code>add_command()</code> erzeugt einen einzelnen <i>normalen</i> Menüpunkt, der zur Ausführung eines Befehls führt. Der Wert der Eigenschaft <code>label</code> dient der sichtbaren Darstellung des Menüpunkts. Im Menü DATEI ist nur der Menüpunkt BEENDEN mit einer Funktion verbunden.
<b>add_separator()</b>	► Die Methode <code>add_separator()</code> erzeugt eine optische Trennlinie zwischen verschiedenen, logisch zu trennenden Menüpunkten innerhalb eines Menüs.
<b>Widget-Variablen</b>	► Zwei Widget-Variablen werden erzeugt, die später mit den zugehörigen Menüpunkten verbunden werden: <ul style="list-style-type: none"> <li>- zur Farbauswahl die Variable <code>farbe</code> vom Typ <code>StringVar</code> mit dem Startwert weiß</li> <li>- zur Auswahl der Sichtbarkeit des Rands die Variable <code>rand</code> vom Typ <code>IntVar</code> mit dem Startwert 0 (= kein Rand)</li> </ul>
	► Es wird das zweite Menü ( <code>mView</code> ) innerhalb der Menüleiste erzeugt.
<b>tearoff</b>	► Die Eigenschaft <code>tearoff</code> wird mit 0 (= <code>false</code> ) belegt. Diese Belegung bewirkt, dass das Menü nicht abgetrennt werden kann.
<b>add_radiobutton()</b>	► Mithilfe der Methode <code>add_radiobutton()</code> werden vier Menüpunkte als zusammengehörige Gruppe erzeugt, deren Auswahl zur Einstellung der Frame-Farbe führt. Die Verbindung wird über die gemeinsame Widget-Variablen hergestellt (Eigenschaft <code>variable</code> ) wie bei einem Radiobutton-Widget. Jeder Menüpunkt hat einen eigenen Wert für die Eigenschaft <code>value</code> . Dieser Wert wird der zugehörigen Widget-Variablen <code>farbe</code> zugewiesen. Bei Ausführung der Funktion <code>farbwechsel()</code> wird dieser Wert dem Frame zugewiesen.
<b>underline</b>	► Die Eigenschaft <code>underline</code> legt die Nummer des Buchstabens des sichtbaren Eintrags fest, der unterstrichen wird. Falls das Menü angewählt wurde, genügt die Eingabe des betreffenden Buchstabens zur Auswahl des Menüpunkts.

- ▶ Mithilfe der Methode `add_checkbutton()` wird ein Menüpunkt als einzelner Ein-Aus-Schalter erzeugt, dessen Auswahl zur Einstellung des Randes führt. Die Verbindung wird über die Widget-Variablen hergestellt (Eigenschaft `variable`) wie bei einem Checkbutton-Widget. Der Wert der Eigenschaften `onvalue` und `offvalue` wird im Fall der Markierung oder Entmarkierung der zugehörigen Widget-Variablen `rand` übergeben. Bei Ausführung der Funktion `randwechsel()` wird dieser Wert dem Frame zugewiesen.
- ▶ Die Methode `add_cascade()` fügt der Menüleiste die beiden vorher erzeugten Menüs (`mFile` und `mView`) hinzu.
- ▶ Dem Hauptfenster wird über die Eigenschaft `menu` die Menüleiste (`mBar`) hinzugefügt.
- ▶ In der Funktion `farbwechsel()` wird der Wert der Widget-Variablen `farbe` mit der Methode `get()` ermittelt und der Frame-Eigenschaft `bg` (= `background`) zugewiesen.
- ▶ In der Funktion `randwechsel()` wird der Wert der Widget-Variablen `rand` mit der Methode `get()` ermittelt und der Frame-Eigenschaft `relief` zugewiesen.

`add_checkbutton()``add_cascade()`

Eigenschaft »menu«

`get()`

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

### Unterschiede unter OS X

Die Menüs werden nicht umgesetzt.

## 11.4.2 Kontextmenüs

Ein Kontextmenü erzeugen Sie auf ähnliche Weise wie eine Menüleiste. Folgende Schritte sind erforderlich:

- ▶ Erzeugung eines Objekts der Klasse `Menu` als Kontextmenü
- ▶ Erzeugung von einzelnen Menüpunkten verschiedenen Typs innerhalb des Kontextmenüs
- ▶ Verbindung eines Ereignisses mit einem bestimmten Widget
- ▶ Aufruf des Kontextmenüs als Folge dieses Ereignisses in der Nähe des betreffenden Widgets

Reihenfolge der Erstellung

Das besondere Problem bei einem Python-Kontextmenü besteht darin, dass es an absoluten Bildschirmkoordinaten eingeblendet wird. Vor dem Einblenden müssen Sie also zunächst ermitteln, an welcher Stelle des Bildschirms sich das Hauptfenster befindet.

**Widget-Position  
ermitteln**

Außerdem müssen Sie die Position des zugehörigen Widgets innerhalb des Hauptfensters berücksichtigen. Sie müssen das Widget mit `place()` absolut positionieren, da sich sonst seine Koordinaten verändern können.

Das folgende Programm enthält ein Label mit einem Bild. Über ein Kontextmenü soll die Hintergrund- oder Rahmenfarbe des Labels geändert werden. Wird die rechte Maustaste innerhalb des Labels betätigt, so wird das Kontextmenü an der aktuellen Mausposition eingeblendet.

```
import tkinter

def ende():
    main.destroy()
def lbpop(e):
    global lbx, lby

    # ermittelt geometrischen Ort des Hauptfensters
    # Form: "width*height+x+y"
    gm = main.geometry()
    teil = gm.split("+")
    mainx = int(teil[1])
    mainy = int(teil[2])

    # Ermittlung der Position des Popup-Menues:
    # Fensterposition plus Labelposition plus Position
    # innerhalb des Labels plus Versatz
    mpop.tk_popup(mainx + lbx + e.x + 5,
                  mainy + lby + e.y + 30)

def farbwechsel():
    lb["bg"] = farbe.get()
main = tkinter.Tk()

# Frame zur Erzeugung der Fenstermasse
fr = tkinter.Frame(main, height=200, width=300)
fr.pack()

# Label mit Bild, Rahmen und Farbe
# Koordinaten des Labels in Variablen
im = tkinter.PhotoImage(file="figur.gif")
lb = tkinter.Label(main, image=im, relief="ridge",
                   bd=5, bg="#000000")
lb.bind("<Button 3>",lbpop)
lbx = 60
lby = 30
lb.place(x=lbx, y=lby, anchor="nw")

# Widget-Variable der Farbe
farbe = tkinter.StringVar()
farbe.set("#000000")
```

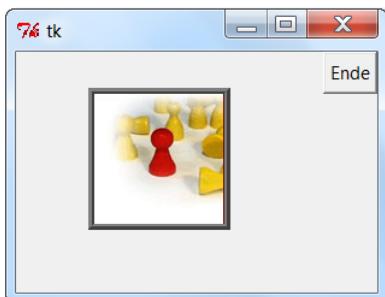
```
# Menue zur Farbeinstellung
mpop = tkinter.Menu(main)
mpop["tearoff"] = 0
mpop.add_radiobutton(label="rot", variable=farbe,
    value="#FF0000", command=farbwechsel)
mpop.add_radiobutton(label="gelb", variable=farbe,
    value="#FFFF00", command=farbwechsel)
mpop.add_radiobutton(label="schwarz", variable=farbe,
    value="#000000", command=farbwechsel)

# Ende
bende = tkinter.Button(main, text="Ende",
    command=ende)
bende.place(relx=1, rely=0, anchor="ne")

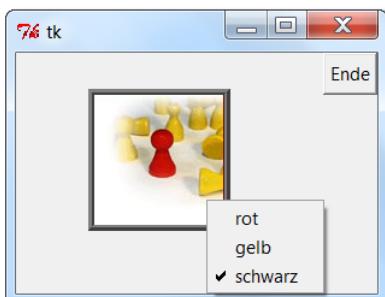
main.mainloop()
```

**Listing 11.25** Datei `gui_kontextmenu.py`

Nach dem Start ist das Label inklusive Bild mit schwarzem Rahmen sichtbar **Startzustand** (siehe [Abbildung 11.38](#)).

**Abbildung 11.38** Label mit Kontextmenü, Startzustand

Nach der Betätigung der rechten Maustaste innerhalb des Labels wird das Kontextmenü an der aktuellen Mausposition eingeblendet (siehe [Abbildung 11.39](#)).

**Nach Bedienung****Abbildung 11.39** Kontextmenü innerhalb des Labels

Zur Erläuterung des Menü-Widgets:

- Klasse »Menu«**
  - ▶ Das Menü (`m.pop`) wird als Menü des Hauptfensters erzeugt.
- tearoff**
  - ▶ Die Eigenschaft `tearoff` wird auf `0` (`= false`) gestellt, da das Abtrennen eines Kontextmenüs nicht sinnvoll ist.
- Menüpunkt**
  - ▶ Es werden drei Radiobuttons als Menüpunkte erzeugt. Diese werden über die Widget-Variable `farbe` miteinander verbunden und ermöglichen den Wechsel zwischen den Farben Rot, Gelb und Schwarz. Startwert für die Widget-Variable ist `schwarz`.

Zur Erläuterung des Kontextmenü-Aufrufs:

- Ereignis**
  - ▶ Beim Label `lb` wird das Ereignis *rechte Maustaste betätigt* mit der Funktion `lb.pop()` verbunden.
- geometry()**
  - ▶ Innerhalb der Funktion `lb.pop()` wird zum Hauptfenster die Methode `geometry()` aufgerufen. Diese Methode liefert eine Zeichenkette mit den Maßen und den Koordinaten des Fensters. Befindet sich das Fenster beispielsweise aktuell an der Position `x=589, y=201`, so sieht die Zeichenkette wie folgt aus: `300x200+589+201`. Die ersten beiden Zahlen bezeichnen die Maße des Labels, hinter den beiden Pluszeichen stehen die aktuellen `x`- und `y`-Koordinaten.
  - ▶ Mithilfe der Methode `find()` aus dem Modul `string` wird die Position der beiden Pluszeichen ermittelt. Die jeweils folgenden Teilstrings werden extrahiert.
- Position des Kontextmenüs**
  - ▶ Aus diesen Werten und den Koordinaten des Labels, der aktuellen Mausposition und einem kleinen Versatz werden die gewünschten Koordinaten des Kontextmenüs berechnet.
- tk\_popup()**
  - ▶ Die Methode `tk_popup()` wird aufgerufen, um das Kontextmenü an diesen Koordinaten erscheinen zu lassen.

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

### Unterschiede unter Ubuntu Linux

Das Kontextmenü öffnen Sie wie unter Windows mit der rechten Maustaste. Allerdings müssen Sie dann die Maustaste gedrückt halten und den Menüpunkt ebenfalls mit der rechten Maustaste auswählen, ansonsten schließt sich das Kontextmenü sofort wieder.

### Unterschiede unter OS X

Das Kontextmenü wird nicht umgesetzt.

### 11.4.3 Messageboxen

Es gibt eine Reihe von vorgefertigten Dialogfeldern, die die Informationsübermittlung zwischen Benutzer und Programm vereinfachen. Diese sogenannten Messageboxen rufen Sie mithilfe von Funktionen des Moduls `tkinter.messagebox` auf. [Tabelle 11.2](#) gibt eine Übersicht über diese Dialogfelder.

Dialogfelder

Funktionsname	Buttons	Funktion
<code>showinfo()</code>	OK	Information an den Benutzer übermitteln, bestätigen lassen
<code>showwarning()</code>	OK	Benutzer warnen, bestätigen lassen
<code>showerror()</code>	OK	Fehler an den Benutzer melden, bestätigen lassen
<code>askyesno()</code>	JA, NEIN	Antwort JA oder NEIN vom Benutzer erfragen
<code>askokcancel()</code>	OK, ABBRECHEN	Antwort OK oder ABBRECHEN vom Benutzer erfragen
<code>askretrycancel()</code>	WIEDERHOLEN, ABBRECHEN	Antwort WIEDERHOLEN oder ABBRECHEN vom Benutzer erfragen
<code>show()</code>	eigene Auswahl von Buttons	Antwort vom Benutzer erfragen, allgemein

**Tabelle 11.2** Funktionen des Moduls »`tkinter.messagebox`«

Im folgenden Programm werden alle sieben Funktionen, jeweils über einen eigenen Button, aufgerufen. Ergibt die Bedienung der Dialogbox eine Antwort, so wird diese Antwort in einem Label dargestellt.

Alle sieben Funktionen

```
import tkinter, tkinter.messagebox

def ende():
    main.destroy()

def msginfo():
    tkinter.messagebox.showinfo \
        ("Info","Eine Info-Box")

def msgwarning():
    tkinter.messagebox.showwarning \
        ("Warnung","Eine Warnungs-Box")
```

Info

Warnung

```

Fehler    def msgerror():
              tkinter.messagebox.showerror \
                  ("Fehler", "Eine Fehler-Box")

Ja, Nein   def msgyesno():
              antwort = tkinter.messagebox.askyesno \
                  ("Ja/Nein", "Eine Ja/Nein-Box")
              if antwort == 1:
                  lbanz["text"] = "Ja"
              else:
                  lbanz["text"] = "Nein"

OK/Abbrechen def msgokcancel():
              antwort = tkinter.messagebox.askokcancel \
                  ("Ok/Abbrechen", "Eine Ok/Abbrechen-Box")
              if antwort == 1:
                  lbanz["text"] = "Ok"
              else:
                  lbanz["text"] = "Abbrechen"

Wiederholen/ def msgretrycancel():
Abbrechen      antwort = tkinter.messagebox.askretrycancel \
                  ("Wiederholen/Abbrechen",
                   "Eine Wiederholen/Abbrechen-Box")
                  if antwort == 1:
                      lbanz["text"] = "Wiederholen"
                  else:
                      lbanz["text"] = "Abbrechen"

Allgemeines  def msgfrage():
Dialogfeld      # hier einmal in allgemeiner Technik, ohne Komfort
                  msgbox = tkinter.messagebox.Message(main,
                                                       type=tkinter.messagebox.ABORTRETRYIGNORE,
                                                       icon=tkinter.messagebox.QUESTION,
                                                       title="Beenden/Wiederholen/Ignorieren",
                                                       message="Beenden, Wiederholen oder Ignorieren")

                  antwort = msgbox.show()

                  if antwort == "abort":
                      lbanz["text"] = "Beenden"
                  elif antwort == "retry":
                      lbanz["text"] = "Wiederholen"
                  else:
                      lbanz["text"] = "Ignorieren"

                  main = tkinter.Tk()

```

```

# Button: Message Info
binfo = tkinter.Button(main,
    text = "Info", command=msginfo)
binfo.pack()

# Button: Message Box Warning
bwarning = tkinter.Button(main,
    text = "Warnung", command=msgwarning)
bwarning.pack()

# Button: Message Box Error
berror = tkinter.Button(main,
    text = "Fehler", command=msgerror)
berror.pack()

# Button: Message Box Ja/Nein
byesno = tkinter.Button(main,
    text = "Ja/Nein", command=msgyesno)
byesno.pack()

# Button: Message Box OK/Cancel
bokcancel = tkinter.Button(main,
    text = "Ok/Abbrechen", command=msgokcancel)
bokcancel.pack()
# Button: Message Box Retry/Cancel
bretrycancel = tkinter.Button(main,
    text = "Wiederholen/Abbrechen", command=msgretrycancel)
bretrycancel.pack()

# Button: Message Box Frage
bfrage = tkinter.Button(main,
    text = "Allgemeine Frage", command=msgfrage)
bfrage.pack()

# Ende-Button
bende = tkinter.Button(main,
    text = "Ende", command=ende)
bende.pack()

# Anzeigelabel
lbanz = tkinter.Label(main)
lbanz.pack()

main.mainloop()

```

**Listing 11.26 Datei gui\_message.py**

Abbildung 11.40 bis Abbildung 11.46 geben jeweils die Darstellung der verschiedenen Dialogboxen unter Windows wieder.



Abbildung 11.40 Info-Box

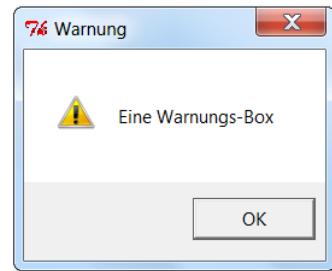


Abbildung 11.41 Warnungs-Box



Abbildung 11.42 Fehler-Box

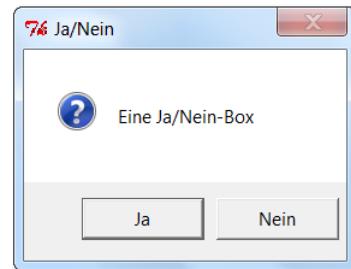


Abbildung 11.43 Ja/Nein-Box

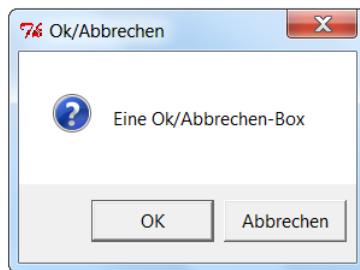


Abbildung 11.44 OK/Abbrechen-Box

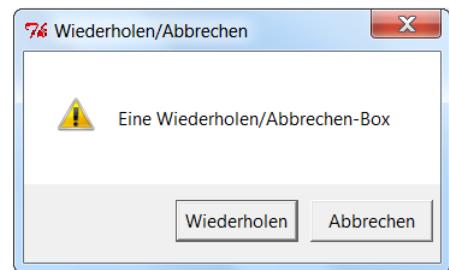


Abbildung 11.45 Wiederholen/  
Abbrechen-Box

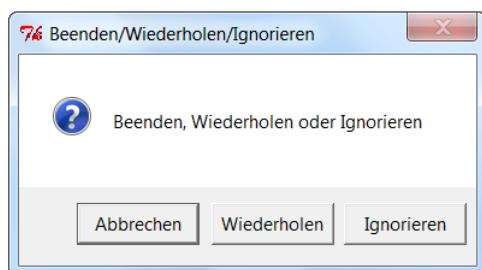


Abbildung 11.46 Allgemeine Frage-Box

Zur Erläuterung:

- ▶ Die verschiedenen Funktionen haben jeweils zwei Parameter:
  - Der erste Parameter enthält den Text für den Titel der Box.
  - Der zweite Parameter enthält den Informationstext für den Benutzer bzw. die Frage neben den Buttons.
- ▶ Zusätzlich werden passende Symbole (Icons) dargestellt und gegebenenfalls **Systemton**
- ▶ Die drei `ask`-Funktionen, die eine Frage an den Benutzer beinhalten, liefern **ask** als Rückgabewert 1 oder 0.
  - Der Wert 1 bedeutet, dass der erste Button gedrückt wurde, also eine positive Antwort gegeben wurde (JA oder OK oder WIEDERHOLEN).
  - Der Wert 0 bedeutet, dass der zweite Button gedrückt wurde, also eine negative Antwort gegeben wurde (NEIN oder ABBRECHEN).
- ▶ Die allgemeine Funktion `show()` ist nicht so komfortabel zu benutzen wie die anderen Funktionen, bietet aber mehr Möglichkeiten bei der Gestaltung: **show()**
  - Es wird zunächst ein Objekt der Klasse `Message` erzeugt (hier: `msgbox`), dem verschiedene Eigenschaften gegeben werden: `type` (welche Buttons), `icon` (dargestelltes Symbol), `title` (Titel) und `message` (Informationstext neben den Buttons).
  - Die Methode `show()` wird auf dieses Objekt angewendet.
  - Rückgabewert ist eine Zeichenkette, die die Bezeichnung des gedrückten Buttons enthält (hier: `abort`, `retry` oder `ignore`).

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`. Das Modul für die Messageboxen heißt `tkMessageBox` statt `tkinter.messagebox`.

#### 11.4.4 Eigene Dialogfelder

Sie haben auch die Möglichkeit, eigene Dialogfelder (= Fenster) zu erzeugen. Damit können Sie die einzelnen Bestandteile einer Anwendung für den Benutzer übersichtlicher gestalten. Allerdings sollten Sie darauf achten, dass der Benutzer nur die gewünschten Fenster und Elemente bedienen kann.

Zur Erzeugung eines eigenen Dialogfelds dient das `Toplevel`-Widget (siehe auch **Toplevel** Abbildung 11.47).

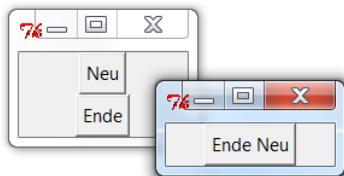


Abbildung 11.47 Eigenes Dialogfeld

Ein erstes Beispiel:

```
import tkinter

# Erzeugt neues Fenster mit Ende-Button
def fenster():
    global neu
    neu = tkinter.Toplevel(main)
    tkinter.Button(neu, text="Ende Neu",
                  command=endeneu).pack()

# Ende neues Fenster
def endeneu():
    neu.destroy()

# Ende Hauptfenster
def ende():
    main.destroy()

# Hauptfenster
main = tkinter.Tk()
tkinter.Button(main, text="Neu", command=fenster).pack()
tkinter.Button(main, text="Ende", command=ende).pack()
main.mainloop()
```

Listing 11.27 Datei gui\_fenster.py

Zur Erläuterung:

- ▶ Im Hauptfenster (`main`) gibt es zwei Buttons. Sie rufen die Funktionen `fenster()` und `ende()` auf.
- ▶ In der Funktion `ende()` wird wie gewohnt mit der Methode `destroy()` das Hauptfenster geschlossen.
- Toplevel()** ▶ In der Funktion `fenster()` wird mit der Methode `Toplevel()` ein neues Fenster erzeugt. Dieses Toplevel-Widget gehört, wie die bereits erwähnten Button-Widgets, zum Hauptfenster. In der globalen Variablen `neu` wird die Referenz auf das neue Fenster gespeichert, damit der Zugriff auf das neue Fenster auch außerhalb der Funktion möglich ist.

- Als Letztes wird in der Funktion `fenster()` ein Button-Widget in dem neuen Fenster erzeugt. Bei Betätigung des Buttons wird die Funktion `endeneu()` aufgerufen. Darin wird mit der Methode `destroy()` das neue Fenster geschlossen. Nach dem Schließen des neuen Fensters gibt es wiederum nur noch das Hauptfenster.

Beim Testen des Programms werden Sie feststellen, dass es noch einen »Schönheitsfehler« hat. Man kann das Hauptfenster auch bedienen, wenn das neue Fenster angezeigt wird. Damit ist es möglich, weitere neue Fenster zu erzeugen. Diese Art der Bedienbarkeit eines Programms ist meist nicht erwünscht. Im nächsten Abschnitt biete ich eine Alternative.

Fehler

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

#### 11.4.5 Ausführung verhindern

In dem folgenden Programm wird darauf geachtet, dass nur die Ereignisfunktionen des aktuell aktiven Fensters zu Aktionen führen. Dies wird über eine Statusvariable geregelt. Das Programm:

Statusvariable

```
import tkinter

# Erzeugt neues Fenster mit Ende-Button
def fenster():
    global status, neu
    if status != "main":
        return
    status = "neu"
    neu = tkinter.Toplevel(main)
    tkinter.Button(neu, text="Ende Neu",
                  command=endeneu).pack()

# Ende neues Fenster
def endeneu():
    global status
    neu.destroy()
    status = "main"

# Ende Hauptfenster
def ende():
    if status == "main":
        main.destroy()

# Hauptfenster
main = tkinter.Tk()
status = "main"
```

```
tkinter.Button(main, text="Neu", command=fenster).pack()
tkinter.Button(main, text="Ende", command=ende).pack()
main.mainloop()
```

**Listing 11.28** Datei `gui_fenster_sperren.py`

Zur Erläuterung:

- ▶ Die Statusvariable mit dem Namen `status` bekommt zunächst den Wert `main`. Nur wenn die Variable diesen Wert hat, kann das Hauptfenster geschlossen oder ein neues Fenster geöffnet werden.
- ▶ Sobald ein neues Fenster geöffnet wurde, bekommt die Statusvariable den Wert `neu`. Im Hauptfenster kann nun keine Aktion mehr ausgeführt werden.
- ▶ Erst nach dem Schließen des neuen Fensters bekommt die Statusvariable wieder den Wert `main`.

**modal** Natürlich ist das neue Fenster noch kein echtes modales Dialogfeld, wie es aus anderen Anwendungen bekannt ist. Dies ist nur mit größerem Programmieraufwand möglich. Damit würde der Rahmen dieses Buchs überschritten. Die Verhaltensweise können Sie damit jedoch zumindest nachempfinden.

### Unterschiede in Python 2

Das Modul heißt `Tkinter` statt `tkinter`.

### Unterschiede unter Ubuntu Linux und OS X

Falls Sie versuchen, das Hauptfenster zu schließen, während das Unterfenster noch geöffnet ist, blockiert dies die Anwendung.

## 11.5 Spiel, GUI-Version

Das bekannte Spiel setzen wir in diesem Abschnitt innerhalb einer Benutzeroberfläche um.

**Eingabe Name** Nach dem Start des Programms öffnet sich aus dem Hauptfenster heraus automatisch ein zweites Dialogfeld zur Eingabe des Spielernamens (siehe [Abbildung 11.48](#)). Zu diesem Zeitpunkt kann das Spiel nicht über das Hauptfenster beendet werden. Durch Betätigung des Buttons `START` wird das zweite Dialogfeld geschlossen, und das Spiel beginnt.

**Eingabe, Auswertung** Innerhalb des Hauptfensters stehen fünf Aufgaben ([Abbildung 11.49](#)). Die Betätigung des Buttons `FERTIG` beendet das Spiel und führt zur Auswertung. Je nach Ergebnis wird eine Highscore-Liste angezeigt ([Abbildung 11.50](#)). Die Daten werden in einer SQLite-Datenbank gespeichert.

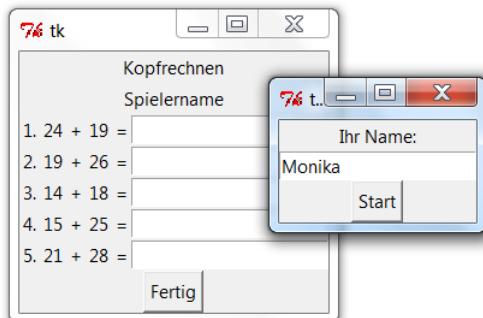


Abbildung 11.48 Spiel, GUI-Version, Eingabe des Namens



Abbildung 11.49 Spiel, GUI-Version, Aufgaben

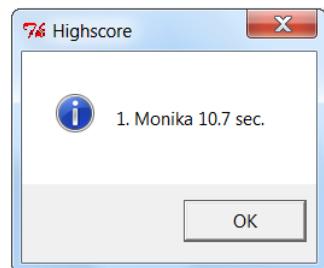


Abbildung 11.50 Spiel, GUI-Version, Highscore

Der Programmcode:

```
import time, random, glob, sqlite3, \
tkinter, tkinter.messagebox

def auswertung():
    if status != "main":
        return
```

```
# Zeit berechnen
endzeit = time.time()
differenz = endzeit - startzeit

# Auswertung
richtig = 0
for i in range(5):
    try:
        # Falsche Eingabe abfangen
        if int(enliste[i].get()) == ergliste[i]:
            richtig = richtig + 1
    except:
        pass

# Kein Highscore
if richtig < 5:
    tkinter.messagebox.showinfo("Kein Highscore",
                               "Leider kein Highscore")
    main.destroy()
    return

# Highscore-DB nicht vorhanden, erzeugen
if not glob.glob("gui_highscore.db"):
    con = sqlite3.connect("gui_highscore.db")
    cursor = con.cursor()
    sql = "CREATE TABLE daten(name TEXT, zeit FLOAT)"
    cursor.execute(sql)
    con.close()

# Datensatz in DB schreiben
con = sqlite3.connect("gui_highscore.db")
cursor = con.cursor()
sql = "INSERT INTO daten VALUES('" \
      + lbname["text"] + "','" + str(differenz) + "')"
cursor.execute(sql)
con.commit()
con.close()

# Highscores sortiert laden
con = sqlite3.connect("gui_highscore.db")
cursor = con.cursor()
sql = "SELECT * FROM daten ORDER BY zeit LIMIT 10"
cursor.execute(sql)
# Ausgabe Highscore
ausgabe = ""
i = 1
for dsatz in cursor:
    ausgabe += str(i) + ". " + dsatz[0] + " " \
              + str(round(dsatz[1],2)) + " sec.\n"
```

```
i = i+1
tkinter.messagebox.showinfo("Highscore", ausgabe)
con.close()
main.destroy()
```

**Listing 11.29** Datei spiel\_gui.py, Teil 1 von 2

Zur Erläuterung:

- ▶ Die Funktion zur Auswertung wird durch Betätigung des Buttons FERTIG im Hauptfenster aufgerufen. Sie hat den größten Umfang innerhalb des Programms.
- ▶ Falls das Hauptfenster nicht aktiv ist, wird die Funktion abgebrochen, ähnlich wie im vorherigen Programm. Ansonsten wird als Nächstes die Spieldauer berechnet.
- ▶ Es gibt eine Liste mit fünf Referenzen auf die Entry-Widgets, in denen der Benutzer seine Eingaben gemacht hat. Außerdem gibt es eine Liste mit den fünf richtigen Ergebnissen. Bei der Auswertung werden die jeweiligen Listenelemente miteinander verglichen.
- ▶ Falls nicht alle Eingaben richtig waren, erscheint eine Messagebox mit dem entsprechenden Hinweis. Das Hauptfenster wird geschlossen. Außerdem wird die Funktion abgebrochen, denn sie liefert auch ohne Hauptfenster weiter.
- ▶ Falls alle Eingaben richtig waren, werden die neu ermittelten Werte in die SQLite-Datenbank geschrieben. Gegebenenfalls muss vorher zunächst die Datenbank mit Tabelle angelegt werden. Anschließend werden die Highscore-Daten sortiert aus der Datenbank geladen und in einer Messagebox ausgegeben. Das Hauptfenster wird geschlossen.

Zwei Listen

SQLite

Der zweite Teil des Programms:

```
def endeneu():
    global startzeit, status
    lbnname["text"] = enname.get()
    startzeit = time.time()
    status = "main"
    neu.destroy()

# Hauptprogramm
main = tkinter.Tk()

# Titel
lbtitel = tkinter.Label(main, text="Kopfrechnen")
lbtitel.grid(row=0, column=0, columnspan=6)

# Name
lbnname = tkinter.Label(main, text="Spielername")
lbnname.grid(row=1, column=0, columnspan=6)
```

```

# Aufgaben
enliste = []      # Liste der Entries
ergliste = []      # Liste der richtigen Ergebnisse
for i in range(5):
    # Aufgabe mit Ergebnis
    a = random.randint(10,30)
    b = random.randint(10,30)
    ergliste.append(a + b)

    # Aufgabenstellung
    tkinter.Label(main, text=str(i+1)+"."). \
        grid(row=i+2, column=0)
    tkinter.Label(main, text=a).grid(row=i+2, column=1)
    tkinter.Label(main, text="+").grid(row=i+2, column=2)
    tkinter.Label(main, text=b).grid(row=i+2, column=3)
    tkinter.Label(main, text="=").grid(row=i+2, column=4)

    # Eingabefeld
    en = tkinter.Entry(main)
    en.grid(row=i+2, column=5)
    enliste.append(en)

    # Ergebnis
    b = tkinter.Button(main, text="Fertig", command=auswertung)
    b.grid(row=7, column=0, columnspan=6)

# Fenster zur Namenseingabe
neu = tkinter.Toplevel(main)
tkinter.Label(neu, text="Ihr Name:").pack()
ename = tkinter.Entry(neu)
ename.pack()
tkinter.Button(neu, text="Start", command=endeneu).pack()
status="neu"

main.mainloop()

```

**Listing 11.30** Datei spiel\_gui.py, Teil 2 von 2

Zur Erläuterung:

- ▶ In der Funktion `endeneu()` wird das zweite Dialogfeld, das zur Namenseingabe dient, geschlossen. Vorher wird der Spielername aus dem Eingabefeld des zweiten Dialogfelds in das Label des Hauptfensters übernommen.
- Startzeit**
- ▶ Außerdem wird die Startzeit festgehalten und die Bedienung des Hauptfensters mithilfe der Statusvariablen wieder freigegeben. Sowohl `startzeit` als auch `status` müssen als globale Variable bekannt gemacht werden.
- Leere Listen**
- ▶ Im Hauptprogramm werden die Elemente des Hauptfensters erzeugt. Zunächst sind dies die Label für Titel und Spielername. Es folgen zwei leere

Listen: für die Referenzen auf die Eingabefelder (= Entry-Widgets) und für die richtigen Ergebnisse.

- ▶ Die Elemente für die fünf Aufgaben werden mithilfe einer Schleife erschaffen. Bei jedem Schleifendurchlauf wird eine zufällige Aufgabe erzeugt und das richtige Ergebnis der oben genannten Liste hinzugefügt. Ergebnisliste füllen
- ▶ Außerdem wird jede Aufgabenstellung in mehreren Labels und einem Eingabefeld dargestellt. Dies geschieht in tabellarisch übersichtlicher Form. Die Referenz auf das Eingabefeld wird der oben genannten Liste hinzugefügt. Eingabeliste füllen
- ▶ Nach der Schleife folgt der Button FERTIG zur Auswertung.
- ▶ Anschließend wird das zweite Dialogfeld zur Namenseingabe erzeugt. Dies geschieht unmittelbar nach Programmstart, ohne ein weiteres Ereignis. Der Benutzer muss also als Erstes seinen Namen eingeben, bevor er weitere Aktionen auslösen kann.

### Unterschiede in Python 2

Das Modul heißt Tkinter statt tkinter. Das Modul für die Messageboxen heißt tkMessageBox statt tkinter.messagebox.



# Kapitel 12

## Neues in Python 3

In diesem Kapitel beschreibe ich einen Teil der vielen Änderungen, die Python mit der Version 3 erfahren hat. Anschließend erläutere ich eine der Umstieghilfen von Version 2 auf Version 3, das Tool *2to3*.

### 12.1 Neue und geänderte Eigenschaften

Alle Beispiele dieses Buchs wurden sowohl für Python 3 als auch für Python 2 entwickelt. Meist ist der Unterschied nicht sehr groß. In diesem Abschnitt beschreibe ich noch einmal viele Änderungen gesondert, die sich in Python 3 gegenüber Python 2 ergeben haben. Es sind Verweise auf die Stellen im Buch eingefügt, an denen die jeweiligen Eigenschaften genauer beschrieben sind.

Python 3 ist die erste Python-Version, die nicht mehr zu 100 % abwärtskompatibel ist. Es gibt mehr und wichtigere Änderungen als bei früheren Versionswechseln. Viele bekannte Störungen und Fehler wurden beseitigt. Für umfassende Informationen über die Änderungen verweise ich auf die Python-Dokumentation.

Alles für Python 3

Nicht 100 %  
abwärts-  
kompatibel

#### 12.1.1 Auffällige Änderungen

Die Anweisung `print` zur Ausgabe auf dem Bildschirm wurde ersetzt durch die Funktion `print()` (siehe [Abschnitt 5.2.1](#), »Funktion ›print()‹«). Es gibt eine Reihe von Funktionen, die *Iteratoren* (siehe [Abschnitt 5.4](#), »Iterierbare Objekte«) oder *Views* liefern (siehe [Abschnitt 4.5.3](#), »Views«).

`print()`

Die Funktion `raw_input()` zur Eingabe von Zeichenketten wurde ersetzt durch die Funktion `input()` (siehe [Abschnitt 3.2.2](#), »Eingabe einer Zeichenkette«).

`input()`

Es können nur noch sinnvolle Vergleiche zwischen Objekten stattfinden. Andernfalls wird eine Ausnahme erzeugt, z. B. beim Vergleich `1 < "abc"`. Statt der besonderen Objektmethode `__cmp__()` müssen Sie jetzt eine der Vergleichsmethoden nutzen, wie z. B. `__lt__()` (siehe [Abschnitt 6.5](#), »Operatormethoden«).

Objektvergleiche

Den Datentyp `long` gibt es nicht mehr, also auch keine Integer-Literale mit einem großen *L* oder einem kleinen *l* am Ende. Alle ganzen Zahlen laufen jetzt unter dem Datentyp `int`. Dieser Datentyp hat keine Obergrenze mehr. Oktalzahlen schreibt man nicht mehr mit `0` (Null) am Anfang, sondern mit `0o` (Null und Buchstabe `o`).

`int, long`

<b>Operatoren</b>	Der Operator / liefert immer eine mathematisch korrekte Division, eine Ganzzahldivision erreichen Sie mit // (siehe <a href="#">Abschnitt 2.1.3, »Division, Ganzzahldivision und Modulo«</a> ). Anstelle des Operators <> (für ungleich) wird nun nur noch der Operator != genutzt.
<b>str, bytes</b>	Zeichenketten werden mit Unicode-Zeichen aufgebaut und sind vom Datentyp str (siehe <a href="#">Abschnitt 4.2, »Zeichenketten«</a> ). Unicode-Literale dürfen nicht mehr mit u"..." oder U"..." gebildet werden. Binärdaten werden im Datentyp bytes gespeichert (siehe <a href="#">Abschnitt 4.2.7, »Datentyp ›bytes‹«</a> ). Sie können mit Byte-Literalen (b"...) gebildet werden.

## 12.1.2 Weitere Änderungen

Bei Definition und Aufruf einer Funktion können Sie eine variable Anzahl von Parametern zusammen mit benannten Parametern einsetzen.

<b>Tupel entpacken</b>	Tupel können auch entpackt werden, wenn nicht genügend Variablen zum Empfang bereitstehen – sofern einer der Variablen ein * (Stern) voransteht. Diese Variable stellt eine Liste der verbleibenden Tupel-Elemente dar, die nicht untergebracht werden konnten (siehe <a href="#">Abschnitt 4.4.3, »Tupel entpacken«</a> ).
<b>Sets</b>	Sets (Mengen) können mit Set-Literalen gebildet werden. Ein Beispiel: a={3,5,7} ergibt das Set a mit drei Elementen.
<b>Ausnahmen</b>	Die Anweisung raise zur Erzeugung von Ausnahmen und die Anweisung except zur Spezifizierung einer Ausnahmebehandlung haben eine geänderte Syntax (siehe <a href="#">Abschnitt 5.6.6, »Unterscheidung von Ausnahmen«</a> ). Die Syntax bei List Comprehensions wurde geändert (siehe <a href="#">Abschnitt 5.5, »List Comprehension«</a> ).
<b>True, False, None</b>	True, False (siehe <a href="#">Abschnitt 4.7.1, »Wahrheitswerte True und False«</a> ) und None (siehe <a href="#">Abschnitt 4.7.2, »Nichts, None«</a> ) gehören zu den reservierten Wörtern. Die Funktion exec() ersetzt die Anweisung exec (siehe <a href="#">Abschnitt 5.1.5, »Funktionen eval() und exec()«</a> ).
<b>import</b>	Zum Import von Modulen wird ohnehin die Schreibweise import Modulname empfohlen (siehe <a href="#">Abschnitt 5.10, »Eigene Module«</a> ). Die alternative Schreibweise from Modulname import ... darf nicht mehr in Funktionen, sondern nur noch auf Modulebene genutzt werden.
<b>format()</b>	Zur Formatierung von Zahlen und Zeichenketten sollten Sie nicht mehr den Operator % einsetzen, er wird in Kürze nicht mehr unterstützt. Stattdessen wird die Funktion format() empfohlen (siehe <a href="#">Abschnitt 5.2.2, »Formatierte Ausgabe mit ›format()‹«</a> ).

Die Funktion `has_key()` zur Prüfung der Existenz eines Dictionary-Elements gibt es nicht mehr. Stattdessen müssen Sie den Operator `in` nutzen (siehe [Abschnitt 4.5.2, »Funktionen«](#)).

Operator in

## 12.2 Konvertierung von Python 2 zu Python 3

Sie können Programme mit dem Kommandozeilentool `2to3` und anderen Hilfen von Python 2-Code in Python 3-Code überführen. Bei einfachen Programmen genügt häufig schon das Tool `2to3` allein. Es findet sich als Datei `2to3.py` im Unterverzeichnis `Tools/Scripts` der Python-Versionen 2 und 3. Der Vorgang soll an einem Beispiel gezeigt werden:

```
import thread, Tkinter
n = raw_input("Ihr Name: ")
print "Hallo", n
```

**Listing 12.1** Datei konvertierung.py

Folgende Elemente werden im Beispiel genutzt:

- ▶ die beiden Bibliotheken für Multithreading und GUIs, die in Python 3 einen anderen Namen haben
- ▶ die Funktion `raw_input()` zur Eingabe einer Zeichenkette, die es in Python 3 nicht mehr gibt
- ▶ die Anweisung `print`, die in Python 3 eine Funktion ist und keine Anweisung

Dieses Programm läuft einwandfrei unter Python 2, aber nicht unter Python 3. In [Abschnitt 2.3.2, »Ausführen unter Windows«](#), wurde beschrieben, wie Sie zur Kommandozeile und in das Python-Verzeichnis gelangen. Das Tool rufen Sie dort wie folgt auf:

```
python Tools/Scripts/2to3.py konvertierung.py
```

Python 2-Programm

Als Ausgabe folgt eine Beschreibung der Vorgänge, die bei der Konvertierung auftreten. Am Ende der Ausgabe steht:

**Files that need to be modified:** konvertierung.py

Die Datei ist noch unverändert. Die Datei wird erst verändert, wenn Sie das Tool wie nachfolgend mit der Option `-w` aufrufen:

```
python Tools/Scripts/2to3.py -w konvertierung.py
```

Am Ende der Ausgabe steht nun:

Python 3-Programm

**Files that were modified:** konvertierung.py

Die Datei wurde konvertiert, wie Sie sehen:

```
import _thread, tkinter  
n = input("Ihr Name: ")  
print("Hallo", n)
```

**Listing 12.2** Datei *konvertierung.py* (konvertiert)

Die alte Version steht noch in der Datei *konvertierung.py.bak* zur Verfügung.

### Unterschiede unter Ubuntu Linux und OS X

Das Tool *2to3* können Sie aus jedem Verzeichnis heraus aufrufen. Der Aufruf lautet einfach `/usr/bin/2to3 konvertierung.py`.

# Kapitel 13

## Raspberry Pi

Der *Raspberry Pi* ist ein vollständiger Computer auf einer kleinen Platine, der einen sehr preisgünstigen Einstieg sowohl in die Software-Welt der Programmierung als auch in die Hardware-Welt der Elektronik ermöglicht. Als Betriebssystem wird häufig das Open-Source-System *Raspbian* verwendet, eine Variante der Linux-Distribution Debian.

Eine Platine

Das *Pi* im Namen steht für *Python Interpreter*. Python steht als primäre Programmiersprache auf jedem Raspberry Pi zur Verfügung. Sie können also mit den Beispielprogrammen dieses Buchs arbeiten.

Python

Zusätzlich können Sie elektronische Schaltungen erstellen und diese mit dem Raspberry Pi über eine dafür vorgesehene Schnittstelle verbinden. Sie senden per Programm Signale zu diesen Schaltungen, um sie zu steuern. Außerdem empfangen Sie per Programm Signale von diesen Schaltungen, um bestimmte Werte zu messen. Diese Werte können Sie auf dem Bildschirm darstellen, speichern oder auch über das Internet verfügbar machen.

Schaltungen

### 13.1 Einzelteile und Installation

Der zentrale Anlaufpunkt im Internet ist die Website des Raspberry-Pi-Projekts: <http://www.raspberrypi.org>. Sie können den Raspberry Pi und die notwendige Peripherie über das Internet oder im Elektronik-Fachhandel erwerben.

[raspberrypi.org](http://www.raspberrypi.org)

#### 13.1.1 Einzelteile

Die Möglichkeiten für den Anschluss der Peripherie sind vielfältig. Im Rahmen dieses Einstiegskapitels schildere ich nachfolgend eine Beispiel-Konstellation. Ich gehe davon aus, dass Ihnen ein Monitor bereits zur Verfügung steht. In Tabelle 13.1 stehen die genutzten Einzelteile mit Preisbeispielen über <http://www.amazon.de>.

Beispiel-Konstellation

Einzelteil	Beispiel-Konstellation	Preis
Computer	Raspberry Pi Mainboard, Modell B, Revision 2.0, 512 MB RAM	35,50 €
Stromanschluss	Steckernetzteil, Micro-USB, 5 V, 1200 mA	8,49 €
Festplatte	SD-Karte, Sandisk SDHC, 8 GB, Class 4	6,99 €
Monitorkabel	HDMI-DVI-Kabel, 2 m, vergoldete Kontakte	5,99 €
Tastatur/Maus	schnurlose USB-Kombination	22,00 €
Internetverbindung	WLAN USB 2.0-Stick, 150 MBit, Nano, WPS-Button, vergoldete Kontakte	7,85 €
Gehäuse	Raspberry-Pi-Gehäuse, klar, belüftet	6,99 €

**Tabelle 13.1** Einzelteile und Preise einer Beispiel-Konstellation

Zur Erläuterung:

- Modell** ► Es gibt auch ältere Typen des Raspberry Pi, z. B. ein Modell A oder ein Modell B in Revision 1.0. Diese bieten andere Anschlüsse und geringeren Arbeitsspeicher.
- Netzteil** ► Der Raspberry Pi benötigt eine Spannung von 5 V. Das Steckernetzteil sollte genügend Strom (hier: 1.200 mA) liefern. Ansonsten könnten einzelne Anschlüsse nicht mehr ausreichend versorgt werden, die Funktionalität des Systems würde leiden, und es könnten Datenverluste auftreten.
- SD-Karte** ► Eine SD-Karte mit 8 GB bietet ausreichend Platz für das Betriebssystem und Ihre eigenen Daten. Durch die Angabe der Class wird die minimale Schreibgeschwindigkeit beim Datentransfer definiert. Sie ist auf der Karte durch eine Zahl in einem fast geschlossenen Kreis gekennzeichnet. Nicht alle Speicherkarten können für den Raspberry Pi genutzt werden.
- Monitor** ► Der Raspberry Pi verfügt über einen HDMI-Anschluss zum Betrieb eines Monitors mit einer HDMI- oder einer DVI-Schnittstelle.
- Internet** ► Der Typ B verfügt über zwei USB-Schnittstellen, die hier für die Tastatur-Maus-Kombination und den WLAN-Stick genutzt werden. Alternativ kann auch ein LAN-Kabel angeschlossen werden.
- Gehäuse** ► Ein Gehäuse ist für den Raspberry Pi nicht zwingend notwendig, schützt ihn aber vor Staub und versehentlicher Berührung.

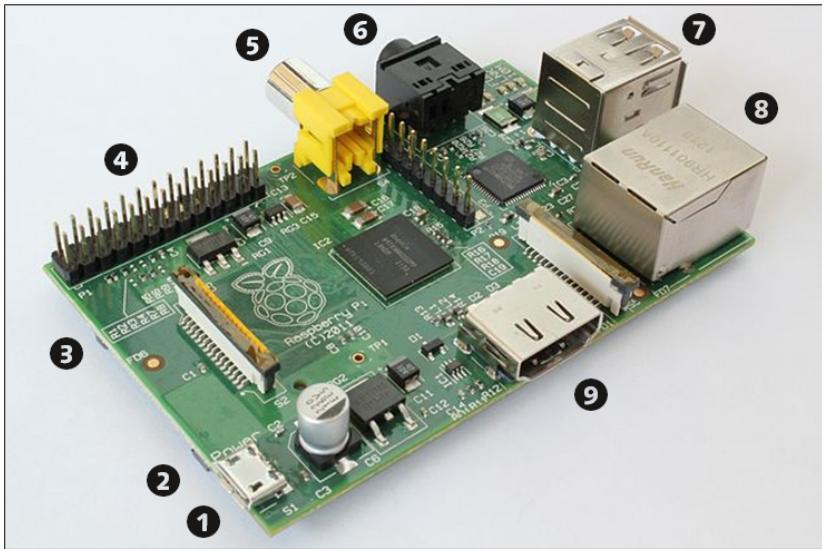


Abbildung 13.1 Raspberry-Pi-Platine, mit Anschlüssen

Zum Verständnis der Abbildung 13.1 wandern wir einmal im Uhrzeigersinn um die Platine herum, beginnend links unten:

- ▶ Bei ① wird der Micro-USB-Stecker des Steckernetzteils angeschlossen.
- ▶ Die SD-Karte wird auf der Rückseite der Platine eingeschoben. ② und ③ markieren den Rand des Einschubs.
- ▶ Bei ④ sehen Sie die 26 Pins des GPIO zum Anschluss von elektronischen Schaltungen. Damit werden wir uns ausführlich ab dem Abschnitt 13.3 beschäftigen. GPIO
- ▶ Bei ⑤ und ⑥ gibt es weitere nutzbare Video- und Audio-Ausgänge.
- ▶ Bei ⑦ liegen zwei USB-Anschlüsse übereinander. Sie werden für den WLAN-Stick und für den Anschluss des Empfängers der schnurlosen Tastatur-Maus-Kombination genutzt. USB
- ▶ Bei ⑧ kann ein LAN-Kabel eingesteckt werden.
- ▶ Bei ⑨ wird das Monitorkabel im HDMI-Anschluss eingesteckt.

### 13.1.2 Sicherheit und Schäden

An dieser Stelle einige Hinweise zur Erhöhung der Sicherheit und zur Vermeidung von Schäden:

- ▶ Bevor Sie die Platine des Raspberry Pi oder andere elektronische Bauteile anfassen, sollten Sie sicherstellen, dass Sie nicht elektrostatisch aufgeladen sind. Ansonsten könnten die Bauteile Schaden erleiden. Fassen Sie kurz an Elektrostatik

einen geerdeten Gegenstand, z. B. einen Heizkörper, dann können vorhandene Ladungen abfließen.

- ▶ Legen Sie die Platine oder andere elektronische Bauteile niemals auf leitende Gegenstände.

**Druck**

- ▶ Üben Sie beim Erstellen der Schaltungen und beim Verbinden mit dem Raspberry Pi keinen zu starken mechanischen Druck auf die Platine oder andere Bauteile aus.

**Stromlos**

- ▶ Verbinden Sie Ihre Schaltungen mit dem Raspberry Pi am besten im stromlosen Zustand, also vor dem Einschalten.

**Haftung**

- ▶ Die Schaltungen in diesem Buch sind genau beschrieben und getestet. Für Schäden, z. B. durch die Nutzung eines defekten Bauteils oder eine unsachgemäße Handhabung oder Schaltung, kann keine Haftung übernommen werden.

### 13.1.3 Zusammenbau

**NOOBS** Als Vorbereitung sollten Sie zunächst das Programm *New Out Of Box Software* (NOOBS) von der Projekt-Website <http://www.raspberrypi.org> auf einen anderen Computer herunterladen. Es umfasst eine Auswahl von installierbaren Linux-Varianten, unter anderem *Raspbian*.

**Formatieren**

Bevor Sie NOOBS auf die SD-Karte kopieren, sollte diese mit einem Programm formatiert werden, das Sie über die Website der Organisation der Hersteller von SD-Karten herunterladen können: [https://www.sdcard.org/downloads/formatter\\_4/eula\\_windows](https://www.sdcard.org/downloads/formatter_4/eula_windows). Beide Programme finden Sie auch auf dem Datenträger zum Buch.

**Anschlüsse**

Schieben Sie die SD-Karte in den Schlitz auf der Rückseite des Raspberry Pi. Die abgeschrägte Ecke verweist auf die korrekte Ausrichtung. Verbinden Sie Monitor, Tastatur und Maus mit der Platine, anschließend den Micro-USB-Anschluss des Steckernetzteils, siehe auch Abbildung 13.1.

**Internet**

Zur Verbindung mit Ihrem Router können Sie (jetzt oder vor einem späteren Start) den USB-WLAN-Stick oder auch ein LAN-Kabel mit dem entsprechenden Anschluss verbinden. Für die Installation ist allerdings keine Verbindung mit dem Internet notwendig.

**Ein/Aus**

Als Letztes wird das Steckernetzteil an das 220 V-Netz angeschlossen. Der Raspberry Pi hat keinen Ein-/Aus-Schalter. Nach der Benutzung fahren Sie das Betriebssystem per Befehl herunter (siehe weiter unten) und trennen anschließend die Verbindung zum 220 V-Netz.

### 13.1.4 Installation

Beim ersten Start erscheint eine Auswahl von Linux-Varianten, die NOOBS zur Verfügung stellt. Wählen Sie RASPBIAN und starten Sie die Installation, mit deutscher Sprache und deutscher Tastatur. Nach ca. 5 bis 10 Minuten startet Raspbian. Beim ersten Start wird noch kein Login verlangt. Bei späteren Starts müssen Sie sich als User `pi` mit dem Passwort `raspberry` anmelden. Lassen Sie sich nicht davon irritieren, dass bei der Passworteingabe keine Zeichen auf dem Bildschirm erscheinen.

Raspbian

Nach dem ersten Start bzw. nach dem Login befinden Sie sich zunächst im Kommandozeilenmodus. Rufen Sie die Benutzeroberfläche mit `startx` auf. Unter den Icons finden Sie das Programm SHUTDOWN zum Herunterfahren des Computers. Dies können Sie auch auf der Kommandozeile durchführen mit `sudo shutdown -h now`. Nicht vergessen: Nach dem Herunterfahren trennen Sie die Verbindung zum 220 V-Netz.

`startx, shutdown`

Hinweis: Der Befehl lautet eigentlich nur `shutdown -h now`. Als normaler Benutzer benötigen Sie allerdings für die Ausführung von systemrelevanten Befehlen höhere Rechte. Dies erreichen Sie, indem Sie `sudo` vor den Befehl setzen. Mit `sudo reboot` könnten Sie einen Neustart durchführen.

`sudo`

Sie sehen Icons für IDLE mit Python 2 und Python 3. Im LXTERMINAL können Sie Python von der Kommandozeile ausführen, siehe [Abschnitt 2.3.3, »Ausführen unter Ubuntu Linux und unter OS X«](#). Zum Editieren einer textbasierten Datei können Sie den Editor *Leafpad* durch Eingabe von `leafpad` aufrufen.

Programme

Hinweis: Sie finden weitere Kommandozeilenbefehle für Linux-Systeme wie z. B. Raspbian im Anhang A.2.

Linux-Befehle

### 13.1.5 Update

Sie sollten Ihr Raspbian-System regelmäßig auf den neuesten Stand bringen. Dies können Sie erreichen mit `sudo apt-get update`.

Probleme

Sollte ein Update einmal »hängen bleiben«, so können Sie den aktuellen Programmlauf mit `[Strg]+[C]` abbrechen. Ein erneuter Aufruf setzt an der Stelle fort, an der Sie vorher abgebrochen haben. Sollte dies mitten im Download einer Datei gewesen sein, dann wird nur noch der Rest der Datei geladen. Anschließend geht das Update weiter.

Falls ein Update immer wieder an derselben Stelle hängen bleibt, dann können Sie die Situation bereinigen, indem Sie eine bestimmte Statusdatei löschen und anschließend mit einem neuen Zeitstempel versehen: `sudo rm /var/lib/dpkg/status`, dann `sudo touch /var/lib/dpkg/status`. Das Update sollte daraufhin laufen.

### 13.1.6 WLAN und Internet

- Scan** Nach dem Anschluss des USB-WLAN-Sticks können Sie auf der Benutzeroberfläche das Programm WIFI-CONFIG aufrufen. Ein Scan erstellt nach kurzer Zeit eine Liste der erreichbaren Router. Mit einem Doppelklick auf den betreffenden Listeneintrag wählen Sie den richtigen aus.
- WPA-TKIP** Viele Router bieten einen verschlüsselten Zugang per WPA mit dem Sicherheitsprotokoll TKIP. Dies wird selbstständig erkannt. Nach Eingabe des Router-Passworts im Feld PSK sollte die Verbindung zum Internet stehen.
- Programme laden** Rufen Sie zum Testen z. B. den Browser MIDORI auf. Über den PI STORE können Sie weitere Programme laden. Über das LXTERMINAL können Sie das Datenbanksystem SQLite 3 (siehe [Abschnitt 10.2, »SQLite«](#)) installieren mit `sudo apt-get install sqlite3`.

## 13.2 Elektronische Schaltungen

- Steuern, Messen, Regeln** Elektronische Schaltungen dienen zum Steuern, Messen und Regeln mithilfe von elektrisch leitenden Verbindungen und elektronischen Bauelementen. Einige einfache Beispiele:
- ▶ Gleichspannungs-Stromkreis mit Widerstand
  - ▶ An- und Ausschalten einer Leuchtdiode (LED)
  - ▶ Abfragen des Zustands eines Tasters (An oder Aus)
  - ▶ Messen der Lufttemperatur und des Luftdrucks mit einem Sensor

### 13.2.1 Gleichspannungs-Stromkreis

- Ohmsches Gesetz** Am Beispiel eines Gleichspannungs-Stromkreises mit Widerstand sollen einige Grundbegriffe erläutert werden. Für den Stromkreis gilt das *Ohmsche Gesetz*  $R = U / I$ . Dabei bezeichnet R den Widerstand (Einheit Ohm, Zeichen  $\Omega$ ), U die Spannung (Einheit Volt, Zeichen V) und I den Strom (Einheit Ampere, Zeichen A).
- Stromfluss** Nehmen wir als Spannungsquelle eine 1,5 V-Batterie, einen Widerstand mit  $470 \Omega$ , einige Stücke Kabel und verbinden das Ganze wie in [Abbildung 13.2](#). Durch das Verbinden wird der Stromkreis geschlossen, und es fließt Strom. Die technische Richtung des Stromflusses ist vom Pluspol zum Minuspol definiert. Dieser Strom soll berechnet werden. Stellen wir die Gleichung für das Ohmsche Gesetz um, so ergibt sich:  $I = U / R = 1,5 \text{ V} / 470 \Omega = \text{ca. } 0,0032 \text{ A} = \text{ca. } 3,2 \text{ mA}$  (für Milli-Ampere).

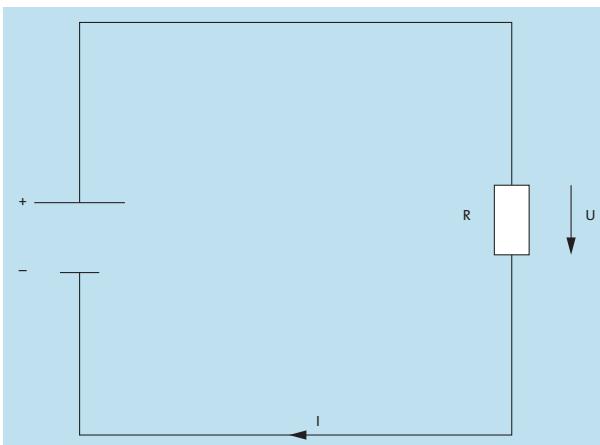


Abbildung 13.2 Gleichspannungs-Stromkreis mit Widerstand

An der Gleichung sehen Sie leicht, dass der Strom ansteigen würde, falls Sie eine größere Spannung, z.B. eine 5 V-Batterie, verwenden würden. Er würde abfallen, falls Sie einen größeren Widerstand, z.B.  $1,3 \text{ k}\Omega$  (für Kilo-Ohm) =  $1.300 \Omega$  nutzen würden.

### 13.2.2 Spannung ist Information

Die 26 Pins des GPIO des Raspberry Pi stellen uns unter anderem eine Spannungsversorgung über Plus- und Minuspole zur Verfügung.

GPIO

Außerdem können wir bestimmte Pins mithilfe eines Programms als Ausgänge definieren. Falls wir eine 1 an den Ausgang senden, dann stellt dieser Pin einem Stromkreis eine Spannung von 3,3 V zur Verfügung. Falls wir eine 0 senden, dann wird keine Spannung zur Verfügung gestellt. Eine 1 wird auch als hoher Spannungspegel (HIGH), eine 0 als niedriger Spannungspegel (LOW) bezeichnet.

Ausgang

Wir können bestimmte Pins auch als Eingänge definieren. Falls wir sie dann mit einzelnen Punkten einer Schaltung verbinden, können wir feststellen, ob dort eine Spannung anliegt oder nicht. Diese Information empfangen wir als Wahrheitswert, True oder False.

Eingang

### 13.2.3 Bauelemente und Ausrüstung

Für die Schaltungen, die in diesem Buch beschrieben werden, benötigen Sie ein Steckbrett, Kabel-Steckbrücken, Widerstände, Leuchtdioden, Taster und Sensoren. Diese Bauteile können Sie über das Internet oder im Elektronik-Fachhandel erwerben. In Tabelle 13.2 sehen Sie einige Bauteile mit Preisbeispielen über <http://www.amazon.de>.

Einzelteil	Beispiel-Konstellation	Preis
Steckbrett	Breadboard, 400 Kontakte	3,05 €
Kabel-Steckbrücken	40 Stück, je 20 cm, Stecker-Buchse	2,99 €
Kabel-Steckbrücken	65 Stück, Stecker-Stecker	2,65 €
Widerstände	ca. 200 Stück, Kemo	4,42 €
Leuchtdioden	ca. 50 Stück, Set S093, Kemo	5,49 €
Schalter und Taster	ca. 20 Stück, Kemo	3,30 €
Sensor	XINTE BMP085 Digital-Sensor für Luftdruck und Lufttemperatur	4,05 €

Tabelle 13.2 Bauelemente und Ausrüstung, Beispiele

- Breadboard** Sie bauen Ihre Schaltungen am besten mithilfe eines Steckbretts (engl.: *breadboard*) auf. Dabei handelt es sich um eine Platine mit sehr vielen Kontakt-Buchsen. Diese Buchsen dienen zur Aufnahme der Stecker der Kabel-Steckbrücken und der Anschlüsse der elektronischen Bauteile. Viele der Kontakt-Buchsen auf dem Steckbrett sind bereits fest miteinander verbunden. Auf diese Weise lassen sich Schaltungen schnell aufbauen, verändern und wieder abbauen.
- Steckbrücke** Die Steckbrücken vom Typ Stecker-Buchse dienen zur Verbindung der GPIO-Pins mit dem Steckbrett. Mithilfe der Steckbrücken vom Typ Stecker-Stecker können Sie zusätzliche Verbindungen auf dem Steckbrett herstellen.
- LED** Eine Leuchtdiode (LED = *Light Emitting Diode*) ist ein Halbleiter, der bei Anlegen von Strom leuchtet.
- Taster** Zum Öffnen und Schließen von Verbindungen innerhalb einer Schaltung können Sie Schalter und Taster nutzen. Schalter stellen eine dauerhafte Verbindung her. Taster verbinden nur, solange man sie drückt.
- Sensor** Die Messwerte von Sensoren können über bestimmte GPIO-Pins an den Raspberry Pi übermittelt werden. Sie können sie anzeigen, speichern oder auch über das Internet zur Verfügung stellen.

### 13.2.4 Widerstände

- Farbcod** Für die Schaltungen werden unterschiedliche Widerstände benötigt. Sie sind mit farbigen Ringen versehen, mit deren Hilfe Sie den Widerstandswert in  $\Omega$  ermitteln können. Es gibt Widerstände mit vier, fünf oder sechs Ringen. In

Tabelle 13.3 sehen Sie die Zuordnung der Farbcodes für Widerstände mit vier Ringen, in Tabelle 13.4 für Widerstände mit fünf (oder sechs) Ringen.

Farbe	Ring 1	Ring 2	Ring 3	Ring 4
Bedeutung	Zehner	Einer	Multiplikator	Toleranz
schwarz	0	0	1	
braun	1	1	10	+/- 1%
rot	2	2	100	+/- 2 %
orange	3	3	1 K	
gelb	4	4	10 K	
grün	5	5	100 K	+/- 0,5 %
blau	6	6	1 M	+/- 0,25 %
violett	7	7	10 M	+/- 0,1 %
grau	8	8	100 M	+/- 0,05 %
weiß	9	9	1 G	
gold			100 m	+/- 5 %
silber			10 m	+/- 10 %

**Tabelle 13.3** Widerstände mit vier Ringen

Zu den Bezeichnungen der Faktoren: 1 K steht für 1 Kilo-Ohm (= 1.000  $\Omega$ ), 1 M für 1 Mega-Ohm (= 1.000.000  $\Omega$ ), 1 G für 1 Giga-Ohm (= 1.000.000.000  $\Omega$ ) und 1 m für 1 Milli-Ohm (= 0,001  $\Omega$ ).

Farbe	Ring 1	Ring 2	Ring 3	Ring 4	Ring 5
Bedeutung	Hunderter	Zehner	Einer	Multiplikator	Toleranz
schwarz	0	0	0	1	
braun	1	1	1	10	+/- 1%
rot	2	2	2	100	+/- 2 %
orange	3	3	3	1 K	

**Tabelle 13.4** Widerstände mit fünf (oder sechs) Ringen

Farbe	Ring 1	Ring 2	Ring 3	Ring 4	Ring 5
Bedeutung	Hunderter	Zehner	Einer	Multiplikator	Toleranz
gelb	4	4	4	10 K	
grün	5	5	5	100 K	+/- 0,5 %
blau	6	6	6	1 M	+/- 0,25 %
violett	7	7	7		+/- 0,1 %
grau	8	8	8		+/- 0,05 %
weiß	9	9	9		
gold				100 m	
silber				10 m	

**Tabelle 13.4** Widerstände mit fünf (oder sechs) Ringen (Forts.)

Der sechste Ring gibt einen Temperaturkoeffizienten an. Daraus kann ermittelt werden, wie sich der Widerstand mit der Temperatur verändert. Dieser Wert wird hier nicht aufgeführt, da die Veränderung für die hier gezeigten Schaltungen nicht ins Gewicht fällt.

### 13.3 Aufbau des GPIO-Anschlusses

**Anschluss** Der GPIO auf der Platine des Raspberry Pi dient zum Anschluss von elektronischen Schaltungen. GPIO steht für *General Purpose Input Output*. In [Tabelle 13.5](#) sehen Sie die Belegung der 26 Pins des GPIO, bezogen auf das Raspberry-Pi-Mainboard, Modell B, Revision 2. Bei anderen Boards sind einzelne Pins anders belegt.

**Modell** Durch Eingabe des Kommandozeilenbefehls `cat /proc/cpuinfo` können Sie sich Informationen zur CPU Ihres Raspberry Pi anzeigen lassen. In der Zeile `REVISION` unterhalb der Zeile `HARDWARE` sehen Sie die Hardware-Revisionsnummer in hexadezimalen Ziffern. Von der Revisionsnummer 4 an aufwärts handelt es sich um ein Mainboard des Modells B, Revision 2.

Zur eindeutigen Zuordnung der Pins ist Pin 1 auf der Platine mit P1 gekennzeichnet. Dies sehen Sie in [Abbildung 13.1](#) oberhalb des Punkts 3.

Erklärung	Pin	Pin	Erklärung
3,3 V	1	2	5,0 V
I2C SDA	3	4	5,0 V
I2C SCL	5	6	GND
GPIO 4	7	8	UART TxD
GND	9	10	UART RxD
GPIO 17	11	12	GPIO 18
GPIO 27	13	14	GND
GPIO 22	15	16	GPIO 23
3,3 V	17	18	GPIO 24
SPI MOSI	19	20	GND
SPI MISO	21	22	GPIO 25
SPI SCLK	23	24	SPI CE0
GND	25	26	SPI CE1

Tabelle 13.5 GPIO, Pin-Belegung

Zur Erläuterung:

- ▶ Die vier Pins mit den Bezeichnungen 5,0 V bzw. 3,3 V können zur Spannungversorgung verwendet werden. Spannung
- ▶ Die vier Pins mit der Bezeichnung GND können als Minus-Pol für Ihre Schaltungen dienen. GND
- ▶ Die neun Pins mit der Bezeichnung GPIO xx können zum Senden von Signalen und zum Empfangen von Signalen programmiert werden. Sie werden auch IO-Pins genannt. Beim Senden eines Signals von einem IO-Pin stehen 3,3 V zur Verfügung. GPIO xx
- ▶ Beachten Sie, dass Sie von außen nicht direkt 5,0 V oder mehr an einen der Pins anlegen dürfen, ansonsten können Schäden auftreten. Beachten Sie außerdem, dass der Strom über einen der Pins GPIO xx einen Wert von 2,8 mA nicht überschreiten darf.
- ▶ Die neun Pins, deren Bezeichnung mit I2C, UART oder SPI beginnt, bieten Anschlüsse für drei unterschiedliche Schnittstellen zur seriellen Übermittlung von Daten. I2C steht für *Inter-Integrated Circuit*, UART steht für *Universal Asynchronous Receiver Transmitter*. I2C, UART, SPI

versal Asynchronous Receiver Transmitter und SPI für Serial Peripheral Interface. Teilweise bieten die Pins einen definierten internen Widerstand und können auch als IO-Pins genutzt werden.

**Modul** Zur Ansteuerung der GPIO-Pins mithilfe von Python wird das Modul `RPi.GPIO` benötigt. Es ist unter den Versionen von Python 2 und Python 3 der Raspbian-Version, die sich auf dem Datenträger zum Buch befindet, bereits installiert.

Geben Sie in IDLE `import RPi.GPIO` ein. Es sollte keine Fehlermeldung erfolgen. Falls doch, dann können Sie das Modul nachinstallieren über den Kommandozeilenbefehl `sudo apt-get install python-rpi.gpio`.

## 13.4 Leuchtdiode

**Halbleiter** Eine Leuchtdiode (LED = *Light Emitting Diode*) ist ein Halbleiter. Halbleiter lassen Strom nur in einer Richtung durch. Die zwei Anschlüsse eines Halbleiters werden Anode und Kathode genannt. Die Anode muss mit dem Pluspol, die Kathode mit dem Minuspol verbunden werden.

**Vorwiderstand** LEDs leuchten bei Anlegen von Strom. Sie erkennen die Anode einer LED daran, dass ihr Anschluss länger ist als der Anschluss der Kathode. LEDs müssen immer mit einem Vorwiderstand betrieben werden. Ansonsten ist der Stromfluss zu hoch, und sie gehen kaputt.

### 13.4.1 Schaltung

In Abbildung 13.3 sehen Sie links einen Stromkreis mit LED und Vorwiderstand. Rechts sehen Sie, wie Sie die betreffenden Bauelemente am GPIO anschließen können. Darin wird GPIO 18 als Ausgang genutzt, also eine Spannung angelegt. Es fließt Strom, und die Leuchtdiode leuchtet. Dieses Verhalten soll in den nachfolgenden Programmen genutzt werden.

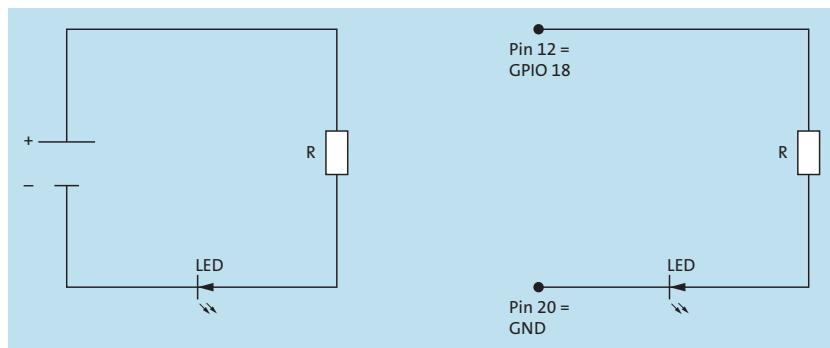


Abbildung 13.3 Stromkreis mit LED und Vorwiderstand

### 13.4.2 LED ein- und ausschalten

Zunächst ein Programm, mit dem Sie die LED einschalten und nach 1,5 Sekunden wieder ausschalten:

```
import RPi.GPIO as gp
import time

gp.setmode(gp.BCM)      # GPIO-Nummern verwenden
gp.setwarnings(False)   # Keine Ausgabe von Warnungen
gp.setup(18, gp.OUT)    # GPIO 18 wird Ausgang

gp.output(18, gp.HIGH)  # LED anschalten
time.sleep(1.5)         # Warten
gp.output(18, gp.LOW)   # LED ausschalten
```

**Listing 13.1** Datei pi\_led.py

Zur Erläuterung:

- ▶ Nach dem Import des Moduls für den GPIO des Raspberry Pi wird die Funktion `setmode()` aufgerufen, damit nachfolgend die GPIO-Nummern verwendet werden können.
- ▶ Eventuell auftretende Warnungen sollen nicht angezeigt werden, daher wird die Funktion `setwarnings()` mit dem Parameter `False` aufgerufen.
- ▶ Mithilfe der Methode `setup()` wird GPIO 18 als Ausgang gesetzt.
- ▶ Die Methode `output()` dient dazu, eine Spannung von 3,3 V oder von 0 V am Anschluss anzulegen (HIGH oder LOW). Zwischen dem Anschalten und dem Ausschalten wird mithilfe der Funktion `sleep()` aus dem Modul `time` 1,5 Sekunden gewartet.

Sie können das Programm mithilfe des folgenden Befehls vom LXTERMINAL aus aufrufen:

```
sudo python3 pi_led.py
```

Die Aufrufe der nachfolgenden Programme lauten entsprechend.

### 13.4.3 LED blinken lassen

Es folgt ein Programm, das die LED fünfmal blinken lässt:

```
import RPi.GPIO as gp
import time

gp.setmode(gp.BCM)
gp.setwarnings(False)
gp.setup(18, gp.OUT)
```

```

for i in range(5):
    gp.output(18, gp.HIGH) # LED anschalten
    time.sleep(0.5)       # Bleibt an
    gp.output(18, gp.LOW)  # LED ausschalten
    time.sleep(0.5)       # Bleibt aus

```

**Listing 13.2** Datei pi\_blink.py

Die LED leuchtet 0,5 Sekunden, ist anschließend 0,5 Sekunden aus, leuchtet dann wieder und so weiter.

#### 13.4.4 LED blinkt Morsezeichen

**Leuchtsignal** Der [Abschnitt 8.9](#), »Beispielprojekt Morsezeichen«, handelte von der Codierung eines Texts in Morsezeichen. Sie wurden als lesbare Zeichen auf dem Bildschirm und als hörbare Signale per Lautsprecher ausgegeben. In diesem Abschnitt werden sie als sichtbare Leuchtsignale einer LED ausgegeben. Das nachfolgende Programm *pi\_morsen.py* ist dem Programm *morse\_nton.py* recht ähnlich. Daher werden hier nur die Unterschiede gelistet und erläutert:

```

import sys, morsen, time
import RPi.GPIO as gp

# Beispieltext codieren
def lichtCode(text, code):
    # Einstellungen fuer GPIO
    gp.setmode(gp.BCM)
    gp.setwarnings(False)
    gp.setup(18, gp.OUT)

    # Zeitschema, Dauer eines Signals in sec.
    signalDauer = {".":0.2, "-":0.6}
    ...

    # Ausgabe des Symbols, kurz oder lang
    gp.output(18, gp.HIGH)
    time.sleep(signalDauer[signal])
    gp.output(18, gp.LOW)
    ...

# Lesefunktion aufrufen
code = morsen.leseCode()

# Ausgabefunktion aufrufen
lichtCode("Hallo Welt", code)

```

**Listing 13.3** Datei pi\_morsen.py, Ausschnitte

Zur Erläuterung der Unterschiede:

- ▶ Statt des Moduls `winsound` wird das Modul `RPi.GPIO` importiert.

- Die Funktion heißt `lichtCode()` statt `tonCode()`. Innerhalb der Funktion werden zunächst die notwendigen Einstellungen für den GPIO vorgenommen. Die Signaldauer wird in Sekunden statt Millisekunden angegeben.
- Zur Ausgabe des Signals wird die LED eingeschaltet und nach Ablauf der Signaldauer wieder ausgeschaltet.

## 13.5 Taster

Taster dienen dazu, für die Dauer der Betätigung eine kurzfristige Verbindung herzustellen. Der Stromkreis wird kurz geschlossen und wieder geöffnet.

### 13.5.1 Schaltung

In Abbildung 13.4 sehen Sie links einen Stromkreis mit Taster. Rechts sehen Sie, wie Sie die betreffenden Bauelemente am GPIO anschließen können. GPIO 18 wird als Eingang genutzt. Sie können damit die Spannung prüfen, die oberhalb des Widerstands anliegt. So können Sie feststellen, ob der Taster gedrückt ist, also Kontakt hat, oder offen ist.

Falls der Taster Kontakt hat, fließt Strom durch den Widerstand. An GPIO 18 liegen dann 3,3 V an. Falls der Taster offen ist, fließt kein Strom durch den Widerstand. An GPIO 18 liegt dieselbe Spannung an wie unterhalb des Widerstands, also 0 V. Diese Prüfung erfolgt in beiden nachfolgenden Programmen.

Kontakt

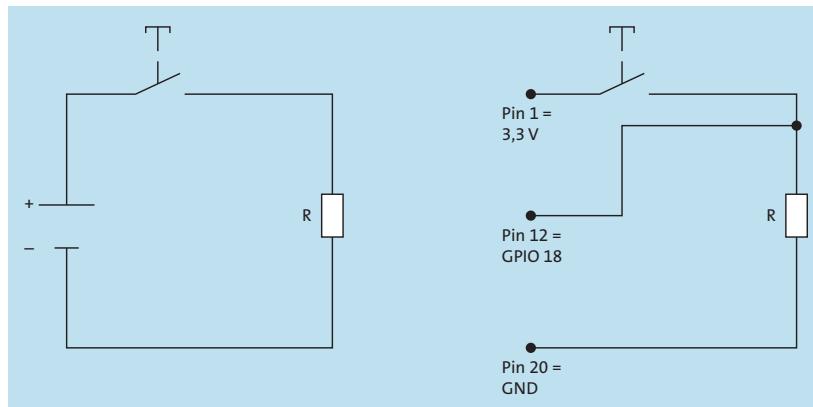


Abbildung 13.4 Stromkreis mit Taster

Es gibt Taster mit verschiedenen Anzahlen von Anschlüssen, z. B.:

- mit zwei Anschlüssen: Die Anschlüsse sind nur dann intern miteinander verbunden, wenn Sie den Taster betätigten.
- mit drei Anschlüssen, die die Bezeichnungen C, NC und NO haben: Mit C (für *common* = dt.: gemeinsam) wird der Basisanschluss bezeichnet. NO steht C, NC, NO

für *normally open*. Dieser Anschluss ist normalerweise nicht mit C verbunden. Dies ändert sich, falls Sie den Taster betätigen. NC steht für *normally connected*. Dieser Anschluss ist normalerweise intern mit C verbunden. Dies ändert sich, falls Sie den Taster betätigen.

### 13.5.2 Einfache Prüfung des Tasters

Es folgt das Programm zur einfachen Prüfung des Tasters:

```
import RPi.GPIO as gp
import time

gp.setmode(gp.BCM)      # GPIO-Nummern verwenden
gp.setwarnings(False)   # Keine Ausgabe von Warnungen
gp.setup(18, gp.IN)     # GPIO 18 wird Eingang

zeitAktuell = time.time()      # Zeiten setzen
zeitEnde = zeitAktuell + 5

while zeitAktuell < zeitEnde:  # Ende erreicht?
    if gp.input(18) == True:   # GPIO abfragen
        print(time.time())
    zeitAktuell = time.time()
```

**Listing 13.4 Datei pi\_taster.py**

- |                       |  |
|-----------------------|--|
| <b>Viele Abfragen</b> | Die Abfragen auf den Zustand des Tasters folgen sehr schnell aufeinander. Daher werden bereits durch eine einzelne, kurze Betätigung des Tasters sehr viele Zeilen ausgegeben. Ein kleiner Ausschnitt der Ausgabe könnte wie folgt aussehen: |
|-----------------------|--|

```
...
1386846620.83
1386846620.83
1386846620.83
1386846620.84
1386846620.84
...
```

Zur Erläuterung:

- |                   |   |
|-------------------|---|
| <b>input()</b>    | <ul style="list-style-type: none"> <li>▶ GPIO 18 wird mithilfe der Methode <code>setup()</code> als Eingang gesetzt.</li> <li>▶ Mithilfe der Methode <code>input()</code> können Sie feststellen, ob an einem Eingang 3,3 V oder 0 V anliegen. Bei jeder Betätigung des Tasters wird die aktuelle Systemzeit in Sekunden ausgegeben.</li> </ul> |
| <b>Systemzeit</b> | <ul style="list-style-type: none"> <li>▶ In der Variablen <code>zeitAktuell</code> wird jeweils die aktuelle Systemzeit ermittelt. Diese wird bei jedem Schleifendurchlauf mit dem Wert von <code>zeitEnde</code> verglichen, damit die Prüfung fünf Sekunden lang durchgeführt wird.</li> </ul>  |

## Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### 13.5.3 Verbesserte Prüfung des Tasters

Es folgt ein Programm, in dem eine verbesserte Prüfung des Tasters vorgenommen wird: Wird zweimal innerhalb von 0,5 Sekunden festgestellt, dass der Tasterkontakt geschlossen ist, zählt die zweite Berührung nicht als neuer Schließvorgang. Damit erreichen wir, dass nur der Moment festgehalten wird, in dem der Kontakt erstmals geschlossen wurde.

Nach Ablauf von 0,5 Sekunden können wir davon ausgehen, dass der Taster in der Zwischenzeit wieder losgelassen wurde. Dann kann ein erneuter Schließvorgang registriert werden.

Das Programm:

```
import RPi.GPIO as gp
import time

gp.setmode(gp.BCM)
gp.setwarnings(False)
gp.setup(18, gp.IN)

zeitAktuell = time.time()
zeitEnde = zeitAktuell + 5
zeitKontaktAlt = 0 # Allererster Wert

while zeitAktuell < zeitEnde:
    if gp.input(18) == True:
        # Neuer Wert
        zeitKontaktNeu = time.time()

        # Falls Zeitabstand groß genug ist
        if zeitKontaktNeu - zeitKontaktAlt > 0.5:
            print(time.time())

        # Für nächste Prüfung
        zeitKontaktAlt = zeitKontaktNeu
    zeitAktuell = time.time()
```

**Listing 13.5** Datei `pi_taster_verbessert.py`

Eine Ausgabe für drei verschiedene Betätigungen könnte wie folgt aussehen:

1386846864.98  
1386846866.26  
1386846868.16

**Nur erster Kontakt**

**Nächster Kontakt**

Zur Erläuterung:

- Zeitdifferenz**
- ▶ Der Zeitpunkt des ersten Tasterkontakte wird in `zeitKontaktAlt` festgehalten. Der nächste Tasterkontakt wird in `zeitKontaktNeu` gespeichert. Falls die Differenz dieser beiden Werte zu klein ist, erfolgt keine Ausgabe.
  - ▶ `zeitKontaktAlt` bekommt nach jeder Bewertung eines Tasterkontakte für die nächste Prüfung den Wert von `zeitKontaktNeu` zugewiesen.

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

## 13.6 Messwerte ermitteln

**BMPO85** In diesem Abschnitt sollen mithilfe einer Schaltung am GPIO und des Sensors BMPO85 die aktuellen Werte für Luftdruck und Lufttemperatur gemessen werden. Daraus kann als dritter Wert die Höhe über dem Meeresspiegel ermittelt werden. Der Sensor BMPO85 bietet eine Möglichkeit zum direkten Anschluss an die I2C-Schnittstelle des GPIO.

### 13.6.1 Schaltung

Die Schaltung ist recht einfach, siehe Abbildung 13.5. Nach der Verbindung des Sensors BMPO85 mit dem Steckbrett schauen Sie auf die Rückseite des Sensors. Daher habe ich die Schaltung von der Rückseite her gezeichnet.

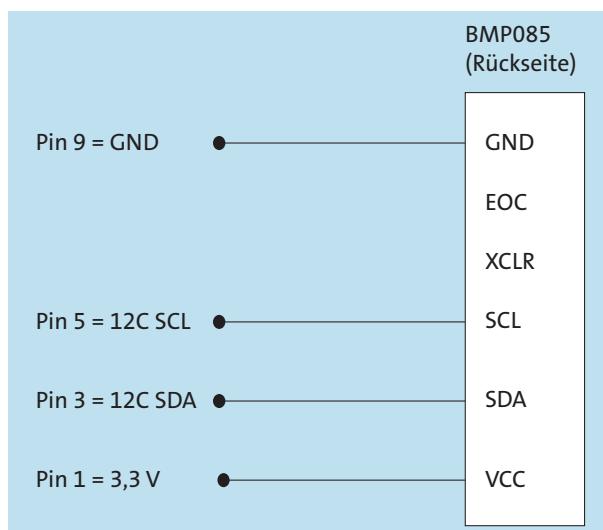


Abbildung 13.5 Schaltung am GPIO für Sensor BMP085

### 13.6.2 Programm zum Ermitteln der Messwerte

Das Programm ist schnell geschrieben:

```
import Adafruit_BMP085

# Sensor im Standardmodus initialisieren
bmp = Adafruit_BMP085.BMP085(0x77)

# Werte ausgeben
print "Lufttemperatur:", bmp.readTemperature(), "Grad Celsius"
print "Luftdruck:", bmp.readPressure()/100.0, "hPa"
print "Hoehe:", bmp.readAltitude(), "m"
```

**Listing 13.6** Datei pi\_bmp085.py

Das Programm wird (unter Python 2) wie folgt aufgerufen:

```
sudo python pi_bmp085.py
```

Eine mögliche Ausgabe des Programms:

**Lufttemperatur: 20.1 Grad Celsius**

**Luftdruck: 960.89 hPa**

**Hoehe: 444.478793435 m**

Das eingebundene Modul Adafruit\_BMP085 stellt seine Inhalte nur unter Python 2 zur Verfügung. Der Wert für die Höhe über Normalnull ist mit Vorsicht zu genießen. Hier muss offensichtlich zuvor eine Kalibrierung erfolgen. Die beiden anderen Werte sind realistisch.

**Adafruit\_BMP085**

Allerdings muss der Raspberry Pi für einen erfolgreichen Lauf dieses Programms zunächst vorbereitet werden. Der Empfang von Daten über die I2C-Schnittstelle muss konfiguriert werden, siehe nächster Abschnitt.

### 13.6.3 Vorbereitung

Zur Vorbereitung des Empfangs von Daten über die I2C-Schnittstelle müssen Sie dafür sorgen, dass beim Start des Raspberry Pi die beiden Kernel-Module *i2c-dev* und *i2c-bcm2708* geladen werden. Dies ist standardmäßig nicht der Fall, da zunächst nur die Module geladen werden, die für den »Normalbetrieb« benötigt werden.

**Kernel-Module**

Sie können sich die aktuell geladenen Module zur Information mit `lsmod` auf-listen lassen. Editieren Sie die Datei, in der die Module stehen, die beim Start geladen werden, mit dem Editor Leafpad: `sudo leafpad /etc/modules`. Tragen Sie die Namen der beiden zusätzlichen Module in zwei Zeilen am Ende der Datei ein. Nach dem Speichern starten Sie den Raspberry Pi neu, mit `sudo reboot`.

**lsmod**

<b>Blacklist</b>	Außerdem müssen Sie die o. a. Module von der Blacklist nehmen. Bei der Blacklist handelt es sich um eine Liste von Modulen, die beim Starten des Raspberry Pi nicht geladen werden sollen. Editieren Sie die Datei, in der die Blacklist steht, mit <code>sudo leafpad /etc/modprobe.d/raspi-blacklist.conf</code> . Setzen Sie vor den Eintrag <code>blacklist i2c-bcm2708</code> das Kommentarzeichen <code>#</code> . Dann wird die betreffende Zeile ignoriert. Speichern Sie die Datei.
<b>Informationen</b>	Die beiden Pakete <code>i2c-tools</code> und <code>python-smbus</code> liefern Ihnen Informationen über die Geräte, die am I2C-Bus angeschlossen sind. Installieren Sie diese beiden Pakete über <code>sudo apt-get install i2c-tools</code> und <code>sudo apt-get install python-smbus</code> .
<b>Adresse 0x77</b>	Nachdem Sie den BMPO85 wie in <a href="#">Abbildung 13.5</a> mit dem GPIO verbunden haben, können Sie mithilfe des Programms <code>i2cdetect</code> prüfen, an welcher Adresse er am I2C-Bus angeschlossen ist. Dies geschieht mit <code>sudo i2cdetect -y 1</code> . Als Ausgabe sehen Sie die hexadezimale Adresse <code>0x77</code> , die auch im <a href="#">Listing 13.6</a> zur Ansteuerung des BMPO85 genutzt wird. Diese Vorbereitungen sind für alle Geräte notwendig, die Sie mit der I2C-Schnittstelle des GPIO verbinden möchten.
<b>Adafruit</b>	Speziell für den Betrieb des BMPO85 wird noch weitere Software benötigt. Sie finden dazu auf dem Datenträger zum Buch die Datei <code>Adafruit.tar</code> . Sie können auch eine aktuelle Version mithilfe des Programms <code>git</code> über die Website der Firma Adafruit erhalten: <code>sudo git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git</code> . Dabei wird auf Ihrem Raspberry Pi das Verzeichnis <code>Adafruit-Raspberry-Pi-Python-Code</code> mit weiteren Unterverzeichnissen angelegt.
<b>Verzeichnis wechseln</b>	Wechseln Sie in das Verzeichnis <code>Adafruit-Raspberry-Pi-Python-Code</code> , von dort aus in das Verzeichnis <code>Adafruit_BMPO85</code> . Hier können Sie das Programm aus <a href="#">Listing 13.6</a> aufrufen. Es werden Module und Klassen benötigt, die in diesem Verzeichnis zu finden sind.
<b>13.7 Messwerte in Datenbank speichern</b>	
<b>SQLite3</b>	In diesem Abschnitt wird dieselbe Schaltung wie in <a href="#">Abschnitt 13.6</a> verwendet. Allerdings werden jetzt die Werte, die mithilfe des Sensors BMPO85 gemessen werden, zusätzlich in einer SQLite3-Datenbank gespeichert. Wie bereits erwähnt, lässt sich SQLite3 mit <code>sudo apt-get install sqlite3</code> schnell unter Raspbian installieren.

### 13.7.1 Datenbank erstellen

Zunächst ein Programm zur Erstellung einer geeigneten Datenbank:

```
import os, sys, sqlite3

# Falls vorhanden, nicht neu erzeugen
if os.path.exists("bmp085.db"):
    print("Datei bereits vorhanden")
    sys.exit(0)

# Verbindung, Cursor
connection = sqlite3.connect("bmp085.db")
cursor = connection.cursor()

# Tabelle erzeugen, Ende
sql = "CREATE TABLE werte(zeit FLOAT, " \
      "temperatur FLOAT, druck FLOAT, hoehe FLOAT)"
cursor.execute(sql)
connection.close()
```

**Listing 13.7** Datei pi\_bmp085\_db.py

Zur Erläuterung:

- ▶ Die Datenbank wird in der Datei *bmp085.db* angelegt. Falls die Datei bereits existiert, wird das Programm wieder verlassen.
- ▶ Falls die Datei noch nicht existiert, dann wird sie erzeugt und geöffnet. Es wird eine Tabelle mit dem Namen *werte* erstellt. Sie beinhaltet vier Felder. Im Feld *zeit* soll der Zeitpunkt festgehalten werden, zu dem die Messung erfolgt. In den drei anderen Feldern werden die drei gemessenen Werte gespeichert.
- ▶ Auf diese Weise lässt sich der zeitliche Verlauf der gemessenen Größen jederzeit später ermitteln.

### 13.7.2 Messen und Speichern

Als Nächstes folgt das Programm zum Messen und Speichern der Werte:

```
import Adafruit_BMP085, os, sys, sqlite3, time

# Falls DB nicht vorhanden, Ende
if not os.path.exists("bmp085.db"):
    print("Datei nicht vorhanden")
    sys.exit(0)

# Sensor im Standardmodus initialisieren
bmp = Adafruit_BMP085.BMP085(0x77)
```

```

# Verbindung, Cursor
connection = sqlite3.connect("bmp085.db")
cursor = connection.cursor()

# Zeiten setzen
zeitAktuell = time.time()
zeitEnde = zeitAktuell + 5

# Ende erreicht?
while zeitAktuell < zeitEnde:
    # Werte ermitteln
    zeit = time.time()
    temp = bmp.readTemperature()
    druck = bmp.readPressure()
    hoehe = bmp.readAltitude()

    # Datensatz erzeugen
    sql = "INSERT INTO werte VALUES(" + str(zeit) + ", " \
          + str(temp) + ", " + str(druck) + ", " \
          + str(hoehe) + ")"
    cursor.execute(sql)
    connection.commit()

    # Warten
    time.sleep(0.3)

    # Zeit fuer Schleife ermitteln
    zeitAktuell = time.time()

# Verbindung beenden
connection.close()

```

**Listing 13.8** Datei pi\_bmp085\_speichern.py

Zur Erläuterung:

- ▶ Nach Aufnahme der Verbindung zur Datenbank wird eine `while`-Schleife über einen Zeitraum von fünf Sekunden gestartet.
- ▶ Innerhalb der Schleife werden die aktuelle Systemzeit und die drei aktuellen Messwerte ermittelt. Diese vier Zahlenwerte werden mithilfe der Funktion `str()` in Zeichenketten umgewandelt und zu einem SQL-Befehl zusammenge setzt. Anschließend wartet das Programm 0,3 Sekunden bis zur nächsten Messung.

### 13.7.3 Anzeige der Daten

Eine Abfrage der Daten könnte wie folgt aussehen:

```
import os, sys, sqlite3

# Falls DB nicht vorhanden, Ende
if not os.path.exists("bmp085.db"):
    print("Datei nicht vorhanden")
    sys.exit(0)

# Verbindung, Cursor
connection = sqlite3.connect("bmp085.db")
cursor = connection.cursor()

# SQL-Abfrage
sql = "SELECT * FROM werte order by zeit desc"
cursor.execute(sql)
for dsatz in cursor:
    print(dsatz[0], dsatz[1], dsatz[2], dsatz[3])

# Verbindung beenden
connection.close()
```

**Listing 13.9** Datei pi\_bmp085\_anzeigen.py

Ein Ausschnitt der möglichen Ausgabe:

```
1386866076.27 18.3 960.05 452.564060119
1386866075.81 18.3 960.12 452.129207088
1386866075.41 18.3 960.12 451.694372394
1386866075.02 18.3 960.15 451.520443649
...

```

Zur Erläuterung:

- ▶ Die Daten werden nach dem Zeitstempel fallend ausgegeben, also die jüngste Messung zuerst.
- ▶ Dank der Speicherung in einer Datenbank können auch andere Ausgaben leicht erstellt werden, z.B. sortiert nach Temperatur.
- ▶ Neben der einfachen Ausgabe der Zahlenwerte wäre auch die Erstellung einer Verlaufsgrafik möglich, z.B. durch Ausgabe in eine CSV-Datei und Übernahme in eine MS Excel-Datei mit Diagramm. Grafik



# Anhang

Hier im Anhang finden Sie die Installation des Pakets XAMPP für die verschiedenen Betriebssysteme, einige wichtige UNIX-Befehle und die Lösungen der Übungsaufgaben.

## A.1 Installation von XAMPP

Zum Testen der Programme des Kapitels 9, »Internet«, über die Internetprogrammierung wird ein lokaler Webserver benötigt. Zusätzlich wird für die Programme des Abschnitts 10.4, »MySQL«, über MySQL-Datenbanken ein Datenbankserver benötigt.

XAMPP ist ein vorkonfiguriertes und einfach zu installierendes Paket, das neben einem Apache-Webserver weitere Software umfasst, z. B. die Webserver-Sprache *PHP* und das Datenbanksystem *MySQL*.

Sie können XAMPP unter <http://www.apachefriends.org> sowohl für Windows als auch für Ubuntu Linux und für OS X herunterladen. Die jeweils aktuellen Versionen stehen auch auf dem Datenträger zum Buch zur Verfügung.

lokaler Server

apache-friends.org

### A.1.1 Installation von XAMPP unter Windows

Für Windows-PCs ab Windows Vista können Sie die Installationsdatei *xampp-win32-1.8.3-2-VC11-installer.exe* der Version 1.8.3-2 von XAMPP für Windows nutzen. Für Windows-PCs ab Windows 2000 können Sie die Installationsdatei *xampp-win32-1.8.2-3-VC9-installer.exe* verwenden.

Version

Durch den Aufruf der jeweiligen Installationsdatei leiten Sie die Installation ein. Es erscheinen zunächst zwei Warnungen, die auf eine möglicherweise eingeschränkte Funktionalität von XAMPP hinweisen. Eine solche konnte aber nicht festgestellt werden. Es können alle bereits markierten Komponenten installiert werden. Das vorgeschlagene Zielverzeichnis *C:\xampp* sollten Sie beibehalten.

Den Apache-Webserver und den MySQL-Datenbankserver starten Sie dann vor der Benutzung über das XAMPP *Control Panel* über die beiden Buttons STARTEN. Sie können die beiden Server nach der Benutzung über die gleichen Buttons wieder anhalten. Dies hat den Vorteil, dass die beiden Server nur dann laufen, wenn Sie sich mit PHP und MySQL beschäftigen. Das XAMPP Control Panel finden Sie nach der Installation im Startmenü und auf dem Desktop.

Server starten

Nach dem Start der Server rufen Sie einen Browser auf und geben dort die Adresse *http://localhost* ein. Damit erreichen Sie das Basisverzeichnis des

C:\xampp\htdocs

lokalen Webservers. Bei einer Standardinstallation von XAMPP im Verzeichnis C:\xampp befindet sich das Basisverzeichnis des lokalen Webservers im Festplattenverzeichnis C:\xampp\htdocs. Darunter werden für die Beispiele dieses Buchs die Unterverzeichnisse Python27 und Python34 angelegt.

#### Hinweis

Unter Microsoft Vista können vereinzelt Probleme auftreten. Aktuelle Informationen finden Sie unter <http://www.apachefriends.org/de/xampp-windows.html> und dort über FAQ – VISTA-PROBLEME.

### A.1.2 Installation von XAMPP unter Ubuntu Linux

In diesem Abschnitt beschreibe ich die Installation von XAMPP unter *Ubuntu Linux 13.10*. Es wird als normaler User gearbeitet, ohne besondere Rechte. Diese werden nur bei Bedarf mithilfe von sudo genutzt.

**Entpacken** Zunächst muss die Datei *xampp-linux-1.8.3-2-installer.run* (mit XAMPP 1.8.3-2 für Linux) in das Verzeichnis /opt/lampp entpackt werden. Dies geschieht mithilfe der Anweisung:

```
sudo ./xampp-linux-1.8.3-2-installer.run
```

Anschließend wechseln Sie in das genannte Verzeichnis mit:

```
cd /opt/lampp
```

**Server starten** Nun können Sie die beiden XAMPP-Server starten:

```
sudo ./xampp start
```

**/opt/lampp/htdocs** Jetzt sollte die Meldung erscheinen, dass die beiden Server gestartet worden sind. Die Adresse der XAMPP-Hauptseite lautet nach der Installation *http://localhost*. Ihre eigenen Programme müssen Sie im Unterverzeichnis /opt/lampp/htdocs oder in darunterliegenden Verzeichnissen speichern. Sie wechseln in das Verzeichnis *htdocs* mit:

```
cd htdocs
```

**gedit** Sie können (neben IDLE) auch den Editor *gedit* zum Schreiben der Python-Programme nutzen. Zum Beispiel rufen Sie nun die Datei mit dem Namen *beispiel.py* auf:

```
sudo gedit beispiel.py
```

**Server beenden** Falls Sie die Server nicht mehr benötigen, können Sie sie beenden mit:

```
sudo ./xampp stop
```

## Probleme bei XAMPP

Bei Nutzern von 64-Bit-Systemen erscheint nach dem Start der XAMPP-Server möglicherweise eine Fehlermeldung. Eine Anleitung zur Beseitigung des Fehlers finden Sie auf der Seite <http://wiki.ubuntuusers.de/XAMPP>. Es müssen die beiden Pakete *gcc-multilib* (für den GNU C Compiler) und *eglibc-source* (für die GNU C Library) installiert werden.

Anschließend sollte das Verzeichnis */opt/lampp* vollständig gelöscht und XAMPP wie oben beschrieben neu installiert werden. Jetzt sollte der Start der beiden XAMPP-Server gelingen.

### A.1.3 Installation von XAMPP unter OS X

In diesem Abschnitt beschreibe ich die Installation von XAMPP unter *OS X 10.9 Mavericks*. Sie rufen die Datei *xampp-osx-1.8.3-2-installer.dmg* mit XAMPP 1.8.3-2 für OS X per Doppelklick auf. Dadurch wird ein neues Laufwerk angelegt. Anschließend können Sie die Installer-Datei auf diesem Laufwerk aufrufen. Dadurch wird XAMPP im Verzeichnis *Programme/XAMPP* auf Ihrem Mac installiert.

DMG-Datei

Nun rufen Sie das Programm *MANAGER-OSX* im Verzeichnis *Programme/XAMPP* auf und wechseln auf die Registerkarte *MANAGE SERVERS*. Hier haben Sie die Möglichkeit, den Apache-Webserver und den MySQL-Server zu starten und zu beenden.

manager-osx

Die Adresse der XAMPP-Hauptseite lautet nach der Installation *http://localhost*. Ihre eigenen PHP-Programme müssen Sie im Unterverzeichnis *Programme/XAMPP/htdocs* oder in darunterliegenden Verzeichnissen speichern.

Programme/  
XAMPP/htdocs

## A.2 UNIX-Befehle

Zur Verwaltung von Verzeichnissen und Dateien unter dem Betriebssystem UNIX oder einem seiner Abkömmlinge, wie z. B. Ubuntu Linux, OS X oder Raspbian können Sie mit Kommandozeilenbefehlen arbeiten. Diese können in einem Terminal unter den genannten Betriebssystemen eingegeben werden.

Kommandozeile

In diesem Abschnitt lernen Sie die nützlichen Befehle *ls*, *mkdir*, *rmdir*, *cd*, *cp*, *mv* und *rm* kennen. Achten Sie auf den Unterschied zwischen Groß- und Kleinschreibung. Ein Beispiel: Es kann zwei verschiedene Dateien mit den Namen *hallo.txt* und *Hallo.txt* geben.

Groß- und Klein-  
schreibung

## A.2.1 Inhalt eines Verzeichnisses

**Hierarchie** UNIX-Systeme verfügen wie Windows-Systeme über eine Hierarchie von Verzeichnissen. Das bedeutet, es gibt ein Hauptverzeichnis, darunter kann es Unterverzeichnisse geben. Diese können wiederum Unterverzeichnisse haben und so weiter.

**.. und .** Mit .. (zwei Punkten) wird immer das jeweils übergeordnete Verzeichnis angeprochen, mit . (einem einzelnen Punkt) das aktuelle Verzeichnis. Diese Bezeichnungen kommen nachfolgend bei einigen Befehlen zum Einsatz.

**ls -l** Mithilfe des Befehls ls -l können Sie sich eine Liste der Dateien und Unterverzeichnisse des aktuellen Verzeichnisses in ausführlicher Form anzeigen lassen. Dies kann eine nützliche Kontrolle nach jeder Veränderung sein. Eine Beispielausgabe:

```
-rw-r--r-- 1 theis theis 12 Dez 3 08:52 hallo.txt
-rw-r--r-- 1 theis theis 12 Dez 3 08:51 gruss.txt
drwxr-xr-x 2 theis theis 4096 Dez 3 08:57 haus
```

Sie sehen zwei Dateien mit der Endung .txt und ein Unterverzeichnis. Die wichtigsten Informationen dazu:

**Verzeichnis** ► Im Unterschied zu Windows sind die Rechte klar unterteilt. Nicht jeder Benutzer darf alles. Falls in der ersten Spalte ein d (für *directory*) steht, so handelt es sich um ein Unterverzeichnis.

**rwx** ► Anschließend folgen die Rechte bezüglich des Eintrags. Dabei steht r (für *read*) für das Leserecht, w (für *write*) für das Schreibrecht und x (für *execute*) für das Ausführungsrecht, z. B. eines Programms.

**Benutzer, Gruppe, Andere** ► Diese Rechte werden dreimal nacheinander aufgelistet: zuerst für den aktuellen Benutzer, dann für die Arbeitsgruppe des aktuellen Benutzers und zuletzt für alle Benutzer des Systems.

**Größe, Zeitpunkt** ► Sie können die Größe der Dateien sehen. In diesem Fall sind es jeweils 12 Byte für die beiden Textdateien. Außerdem sehen Sie Datum und Uhrzeit der letzten Änderung.

## A.2.2 Verzeichnis anlegen, wechseln und löschen

**mkdir** Der Befehl mkdir (*make directory*) dient zum Anlegen eines neuen Verzeichnisses unterhalb des aktuellen Verzeichnisses. Ein Beispiel:

► mkdir meineTexte : Anlegen des Unterverzeichnisses *meineTexte*, relativ zum aktuellen Verzeichnis

**cd** Mit dem Befehl cd (*change directory*) wechseln Sie das Verzeichnis. Einige Beispiele:

- ▶ `cd meineTexte` : Wechsel in das Unterverzeichnis `meineTexte`, relativ zum aktuellen Verzeichnis
- ▶ `cd ..` : Wechsel in das übergeordnete Verzeichnis
- ▶ `cd (ohne weitere Angaben)` : Wechsel in Ihr Heimatverzeichnis, unabhängig vom aktuellen Verzeichnis
- ▶ `cd /usr/bin` : Wechsel in das absolute Verzeichnis `usr/bin`, unabhängig vom aktuellen Verzeichnis

Der Befehl `rmdir` (*remove directory*) dient zum Löschen eines leeren Unterverzeichnisses, das sich unterhalb des aktuellen Verzeichnisses befindet. Zum Löschen von Dateien aus einem Verzeichnis verweise ich auf den nächsten Abschnitt. Ein Beispiel:

`rmdir`

- ▶ `rmdir meineTexte` : Löschen des Unterverzeichnisses `meineTexte`, falls es leer ist

### A.2.3 Datei kopieren, verschieben und löschen

Textbasierte Dateien können Sie mit einem Editor anlegen, z. B. *gedit*, *leafpad* oder *nano*. Python-Programme erstellen Sie am besten innerhalb der IDLE-Umgebung.

`Editor`

Zum Kopieren von Dateien nutzen Sie den Befehl `cp` (*copy*). Es werden immer zwei Angaben benötigt, zum einen für die Quelle und zum anderen für das Ziel der Kopieraktion. Einige Beispiele:

`cp`

- ▶ `cp hallo.txt gruss.txt` : Kopieren der Datei `hallo.txt` in die Datei `gruss.txt` innerhalb des aktuellen Verzeichnisses
- ▶ `cp hallo.txt ..` : Kopieren der Datei `hallo.txt` in das übergeordnete Verzeichnis
- ▶ `cp hallo.txt ../nochMehrTexte` : Kopieren der Datei `hallo.txt` in das Verzeichnis `nochMehrTexte`, das sich unter demselben übergeordneten Verzeichnis wie das aktuelle Verzeichnis befindet
- ▶ `cp ../hallo.txt .` : Kopieren der Datei `hallo.txt` aus dem übergeordneten Verzeichnis in das aktuelle Verzeichnis. Beachten Sie den einzelnen Punkt am Ende des Befehls.
- ▶ `cp ../*.txt .` : Kopieren aller Dateien mit der Endung `.txt` aus dem übergeordneten Verzeichnis in das aktuelle Verzeichnis
- ▶ `cp ../nochMehrTexte/hallo.txt .` : Kopieren der Datei `hallo.txt` aus dem Verzeichnis `nochMehrTexte` (siehe oben) in das aktuelle Verzeichnis

Mithilfe des Befehls `mv` (*move*) können Sie Dateien umbenennen bzw. verschieben. Die Benutzung ist der von `cp` sehr ähnlich. Einige Beispiele:

`mv`

- ▶ `mv hallo.txt gruss.txt` : Umbenennen der Datei `hallo.txt` in die Datei `gruss.txt` innerhalb des aktuellen Verzeichnisses

- ▶ `mv hallo.txt ..` : Verschieben der Datei *hallo.txt* in das übergeordnete Verzeichnis
  - ▶ `mv ../hallo.txt .` : Verschieben der Datei *hallo.txt* aus dem übergeordneten Verzeichnis in das aktuelle Verzeichnis
  - ▶ `mv .../*.txt .` : Verschieben aller Dateien mit der Endung *.txt* aus dem übergeordneten Verzeichnis in das aktuelle Verzeichnis
- rm** Zum Löschen von Dateien nutzen Sie den Befehl `rm` (*remove*). Einige Beispiele:
- ▶ `rm hallo.txt` : Löschen der Datei *hallo.txt* innerhalb des aktuellen Verzeichnisses
  - ▶ `rm *.txt` : Löschen aller Dateien mit der Endung *.txt* innerhalb des aktuellen Verzeichnisses

## A.3 Lösungen

Sollten Sie die Übungsaufgaben nicht vollständig gelöst haben, geben Ihnen die folgenden Lösungen eine Hilfestellung. Wenn Sie selbst eine lauffähige Lösung gefunden haben, sollten Sie diese mit der hier vorgestellten Lösung vergleichen.

### A.3.1 Lösungen zu Kapitel 2

Es folgen die Lösungen zu den Übungen in [Kapitel 2](#).

**Übung u\_grundrechenarten:**

```
>>> 13 - 5 * 2 + 12 / 6
5.0
>>> 7 / 2 - 5 / 4
2.25
>>> (12 - 5 * 2) / 4
0.5
>>> (1 / 2 - 1 / 4 + (4 + 3) / 8) * 2
2.25
```

**Übung u\_inch:**

```
>>> inch = 2.54
>>> 5 * inch
12.7
>>> 20 * inch
50.8
>>> 92.7 * inch
235.458
```

### Unterschiede in Python 2

Der Operator / ergibt in Python 2 bereits eine Ganzzahldivision. Zur Vermeidung des Verlusts der Nachkommastellen können Sie z. B. die folgenden Änderungen durchführen (auch wenn es im ersten Fall mathematisch nicht notwendig ist):

```
>>> 13 - 5 * 2 + 12.0 / 6
5.0
>>> 7.0 / 2 - 5.0 / 4
2.25
>>> (12 - 5 * 2) / 4.0
0.5
>>> (1.0 / 2 - 1.0 / 4 + (4 + 3) / 8.0) * 2
2.25
```

### A.3.2 Lösungen zu Kapitel 3

Es folgen die Lösungen zu den Übungen in [Kapitel 3](#).

#### Übung u\_eingabe\_inch:

```
# Umrechnungsfaktor
inch = 2.54

# Eingabe des Inch-Wertes
print("Bitte geben Sie den Inch-Wert ein:")
xi = float(input())

# Umrechnung
xcm = xi * inch

# Ausgabe
print(xi, "inch sind", xcm, "cm")
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

#### Übung u\_eingabe\_gehalt:

```
# Eingabe
print("Geben Sie Ihr Bruttogehalt in Euro ein:")
bruttobetrag = float(input())
```

```
# Umrechnung  
steuerbetrag = bruttobetrag * 0.18  
  
# Ausgabe  
print("Es ergibt sich ein Steuerbetrag von",  
      steuerbetrag, "Euro")
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### Übung u\_verzweigung\_einfach:

```
# Eingabe  
print("Geben Sie Ihr Bruttogehalt in Euro ein:")  
bruttobetrag = float(input())  
  
# Umrechnung  
if bruttobetrag > 2500:  
    steuerbetrag = bruttobetrag * 0.22  
else:  
    steuerbetrag = bruttobetrag * 0.18  
  
# Ausgabe  
print("Es ergibt sich ein Steuerbetrag von",  
      steuerbetrag, "Euro")
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### Übung u\_verzweigung\_mehrfach:

```
# Eingabe  
print("Geben Sie Ihr Bruttogehalt in Euro ein:")  
bruttobetrag = float(input())  
  
# Umrechnung  
if bruttobetrag > 4000:  
    steuerbetrag = bruttobetrag * 0.26  
elif bruttobetrag < 2500:
```

```

steuerbetrag = bruttobetrag * 0.18
else:
    steuerbetrag = bruttobetrag * 0.22

# Ausgabe
print("Es ergibt sich ein Steuerbetrag von",
      steuerbetrag, "Euro")

```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### Übung u\_operator:

```

# Eingabe
print("Geben Sie Ihr Bruttogehalt in Euro ein:")
gehalt = float(input())
print("Geben Sie Ihren Familienstand ein"
      + " (1=ledig, 2=verheiratet):")
fs = int(input())

# Umrechnung
if gehalt > 4000 and fs == 1:
    sb = gehalt * 0.26
elif gehalt <= 4000 and fs == 2:
    sb = gehalt * 0.18
else:
    sb = gehalt * 0.22

# Ausgabe
print("Es ergibt sich ein Steuerbetrag von", sb, "Euro")

```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### Übung u\_range:

Schleife 1

2

3

6.5

-7

Schleife 2

3

6

9

Schleife 3

-3

1

5

9

13

Schleife 4

3

0

-3

-6

-9

### Übung u\_range\_inch:

```
# Umrechnungsfaktor  
inch = 2.54  
  
# Schleife  
for xi in range (15, 41, 5):  
    xcm = xi * inch  
    print(xi, "inch =", xcm, "cm")
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

### Übung u\_while:

```
# Umrechnungsfaktor  
inch = 2.54  
  
# Erste Eingabe  
print("Bitte geben Sie den Inch-Wert ein:")  
xi = float(input())  
  
# while-Schleife  
while xi != 0:  
    # Umrechnung, Ausgabe  
    xcm = xi * inch  
    print(xi, "inch sind", xcm, "cm")
```

```
# Weitere Eingabe
print("Bitte geben Sie den Inch-Wert ein:")
xi = float(input())
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

### Übung u\_feher:

```
# Umrechnungsfaktor
inch = 2.54
# Initialisierung der while-Schleife
fehler = 1

# Schleife bei falscher Eingabe
while fehler == 1:
    # Eingabe
    print("Bitte geben Sie den inch-Wert ein")
    xi = input()

    # Versuch der Umwandlung
    try:
        xi = float(xi)
        fehler = 0
    # Fehler bei Umwandlung
    except:
        print("Falsche Eingabe")

# Umrechnung, Ausgabe
xcm = xi * inch
print(xi, "inch sind", xcm, "cm")
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Die Funktion zur Eingabe heißt `raw_input()`.

### Übung u\_parameter:

```
# Funktion
def steuer(bb):
    # Umrechnung
    if bb > 2500:
        sb = bb * 0.22
```

```
else:  
    sb = bb * 0.18  
  
    # Ausgabe  
    print("Bruttobetrag:", bb,  
          "Euro, Steuerbetrag:", sb, "Euro")  
  
# Verschiedene Werte  
steuer(1800)  
steuer(2200)  
steuer(2500)  
steuer(2900)
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### Übung u\_rueckgabewert:

```
# Funktion  
def steuer(bb):  
    # Umrechnung  
    if bb > 2500:  
        sb = bb * 0.22  
    else:  
        sb = bb * 0.18  
  
    # Ergebnis senden  
    return sb  
  
# Verschiedene Werte  
print("Bruttobetrag: 1800 Euro, Steuerbetrag:",  
      steuer(1800), "Euro")  
print("Bruttobetrag: 2200 Euro, Steuerbetrag:",  
      steuer(2200), "Euro")  
print("Bruttobetrag: 2500 Euro, Steuerbetrag:",  
      steuer(2500), "Euro")  
print("Bruttobetrag: 2900 Euro, Steuerbetrag:",  
      steuer(2900), "Euro")
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

### A.3.3 Lösungen zu Kapitel 5

Es folgen die Lösungen zu den Übungen in Kapitel 5.

#### Übung u\_tabelle:

```
# Umrechnungsfaktor
inch = 2.54

# Tabellenkopf
print("{0:>7}{1:>7}".format("inch", "cm"))

# Schleife
for xi in range (15, 41, 5):
    xcm = xi * inch
    print("{0:7.1f}{1:7.1f}".format(xi,xcm))
```

#### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen.

#### Übung u\_modul

Datei `u_modul_finanz.py`:

```
# Funktion
def steuer(bb):
    # Umrechnung
    if bb > 2500:
        sb = bb * 0.22
    else:
        sb = bb * 0.18

    # Ergebnis senden
    return sb
```

Datei `u_modul.py`:

```
# Import
import u_modul_finanz

# Verschiedene Werte
print("Bruttobetrag: 1800 Euro, Steuerbetrag:",
      u_modul_finanz.steuer(1800), "Euro")
print("Bruttobetrag: 2200 Euro, Steuerbetrag:",
      u_modul_finanz.steuer(2200), "Euro")
print("Bruttobetrag: 2500 Euro, Steuerbetrag:",
```

```
u_modul_finanz.steuer(2500), "Euro")
print("Bruttobetrag: 2900 Euro, Steuerbetrag:",
      u_modul_finanz.steuer(2900), "Euro")
```

### Unterschiede in Python 2

Die Klammern bei der Anweisung `print` entfallen. Es wird das Zeichen \ für den Umbruch von langen Programmzeilen eingesetzt.

# Index

- (minus)	
<i>Set</i>	128
<i>Subtraktion</i>	24
<i>!=</i> (ungleich)	44
# (Kommentar)	33
<i>\$_GET</i>	291
<i>\$_POST</i>	294
% (Prozent)	
<i>Format</i>	153, 422
<i>Modulo</i>	25, 91
%% (Format)	155
%= (Zuweisung)	142
%c (Zeitformat)	219
%d (Format)	155
%e (Format)	155
%f (Format)	155
%o (Format)	155
%s (Format)	155
%x (Format)	155
& (Bitoperator)	93
& (Set)	128
' (Hochkomma)	97
* (Stern)	
<i>Multiplikation</i>	24
<i>Parameter</i>	173
<i>Vervielfachung</i>	98, 111
** (Potenz)	88
**= (Zuweisung)	142
*= (Zuweisung)	142
+ (plus)	
<i>Addition</i>	24
<i>Verkettung</i>	98, 111
+= (Zuweisung)	142
. (Verzeichnis)	452
.. (Verzeichnis)	452
/ (Division)	24
// (Ganzahldivision)	24
//= (Zuweisung)	142
/= (Zuweisung)	142
: (Doppelpunkt)	45, 63
< (kleiner)	
<i>Format</i>	154
<i>Python</i>	44
<i>SQL</i>	326
<< (Bitoperator)	93
<= (kleiner gleich)	
<i>Python</i>	44
<i>SQL</i>	326
<> (ungleich)	
<i>Python</i>	422
<i>SQL</i>	326
=	
<i>Python (Zuweisung)</i>	27, 38
<i>SQL (gleich)</i>	326
-= (Zuweisung)	142
== (gleich)	44, 125, 136
> (größer)	
<i>Format</i>	154
<i>Python</i>	44
<i>SQL</i>	326
>= (größer gleich)	
<i>Python</i>	44
<i>SQL</i>	326
>> (Bitoperator)	93
[ ], Dictionary	121
^ (hoch)	
<i>Bitoperator</i>	93
<i>Set</i>	128
<i>\$_omega_(Ohm)</i>	430
<i>__add__( )</i>	202
<i>__cmp__( )</i>	421
<i>__del__( )</i>	199
<i>__eq__( )</i>	202
<i>__floordiv__( )</i>	202
<i>__ge__( )</i>	202
<i>__gt__( )</i>	202
<i>__init__( )</i>	199
<i>__le__( )</i>	202
<i>__lt__( )</i>	202
<i>__mod__( )</i>	202
<i>__mul__( )</i>	202
<i>__ne__( )</i>	202
<i>__pow__( )</i>	202
<i>__repr__( )</i>	200
<i>__str__( )</i>	200
<i>__sub__( )</i>	202
<i>__truediv__( )</i>	202
<i>_thread</i>	
<i>get_ident()</i>	230
<i>Modul</i>	229

_thread (Forts.)	
start_new_thread()	229
{}	
Dictionary	120
Formatierung	152
(Bitoperator)	93
(Set)	128
~ (Bitoperator)	93
Ob	86
Oo	86
Ox	86
100-Dollar-Laptop	18
2to3	423
<b>A</b>	
A (Ampere)	430
abs()	96
Adafruit	444
Adafruit_BMPO85	443
add()	127
add_cascade()	403
add_checkbutton()	403
add_command()	402
add_radiobutton()	402
add_separator()	402
Aktionsabfrage (Datenbank)	322
Alt-Taste (Linux)	378
Ampere	430
anchor	352
and	
logisch	48
and (SQL)	326
Anode	436
Antwort-Box (GUI)	407
Anweisung	
ausführen	147
leere	145
Anzahl	
Elemente	100
Teiltexte	103
Anzeigefeld (GUI)	351
Apache-Webserver	285, 337, 449
append()	115, 228
appendleft()	228
argv	
sys	192
Arithmetisches Mittelwert	189
Array	109
as	172, 191
askokcancel()	407
Audio-Ausgabe	241
Ausdruck	
auswerten	147
bedingter	156
Ausgabe	
formatieren	151, 155
lange	34
Ausnahme	
Thread	233
Ausnahmebehandlung	68
Ausrichtung (GUI)	352
Auswahlabfrage (Datenbank)	323
Auswahlmenü	
einfaches (GUI)	361
mehrfaches (GUI)	363
Auswahlmenü (HTML)	303
<b>B</b>	
b	
Format	154
Prefix	108
Basisverzeichnis	449
Bauteil	
elektronisch	431
Bedingter Ausdruck	156
Bedingung	
44, 63, 131	
verknüpfen	48
Beep()	
winsound	241, 273
Benannter Parameter	174, 422
Benutzername	
pi	429
Benutzeroberfläche	347
Betrag	96
bg	352
Bild einbinden (GUI)	352
Bild-Widget	354
bin()	86, 93
bind()	378, 380
Bit	
schieben	93
setzen	93
Bitoperator	
Blacklist	444
BMPO85	442
Bogenmaß	90

bool .....	131
borderwidth .....	352
Breadboard .....	432
break .....	54
Breakpoint .....	169
Breite (GUI) .....	351
Browser .....	286
<i>aufrufen</i> .....	309
Bruch .....	94
<i>annähern</i> .....	95
Button .....	349
Button-Widget .....	349
Byte .....	91
Byte-Literal .....	108, 288
bytes .....	108
<i>decode()</i> .....	109
bytes() .....	108

**C**

C (Taster) .....	439
Callback (GUI) .....	349
cd .....	452
C-Formatierung .....	155
cgi	
<i>FieldStorage</i> .....	300
<i>Modul</i> .....	296, 299
cgi-bin .....	296, 299
CGI-Skript .....	296
<i>Standardverzeichnis</i> .....	299
cgitb	
<i>enable()</i> .....	299
<i>Modul</i> .....	299
checkbox (HTML) .....	306
Checkbutton-Widget .....	371
checked (HTML) .....	306
chr() .....	186
class .....	196
clear() .....	127
close() .....	245
<i>connection</i> .....	323, 338
<i>cursor</i> .....	338
cmd .....	31
collections	
<i>deque()</i> .....	228
<i>Modul</i> .....	226
command .....	349, 369, 372
Comma-separated Values .....	252

commit() .....	322, 338
Compiler .....	18
Conditional Expression .....	156
configure() .....	351
connect() .....	322
<i>connector</i> .....	338
connection	
<i>close()</i> .....	323, 338
<i>commit()</i> .....	322, 338
<i>cursor()</i> .....	322, 338
<i>sqlite3</i> .....	322
connector	
<i>connect()</i> .....	338
<i>mysql</i> .....	338
Connector / Python .....	337
Content-type .....	297
continue .....	70
copy	
<i>deepcopy()</i> .....	139, 203
<i>Modul</i> .....	139, 203
copy() .....	127
copyfile() .....	268
cos() .....	90
count() .....	115
<i>str</i> .....	103
cp .....	453
cp1252 .....	30, 297
create table (SQL) .....	322
CSV-Datei .....	252
cursor	
<i>close()</i> .....	338
<i>execute()</i> .....	338
<i>fetchall()</i> .....	342
cursor() .....	322
<i>connection</i> .....	338

**D**

d (Format) .....	154
date (SQL) .....	337
Datei	
<i>anlegen</i> .....	453
<i>Eigenschaften</i> .....	266
<i>Ende erkennen</i> .....	250
<i>Existenz prüfen</i> .....	268
<i>festgelegte Struktur</i> .....	256
<i>formatierte Ausgabe</i> .....	256
<i>Größe</i> .....	267

Datei (Forts.)	
<i>kopieren</i>	268, 453
<i>lesen</i>	247
<i>lesen und schreiben</i>	259
<i>Liste erstellen</i>	265
<i>löschen</i>	268, 454
<i>öffnen</i>	244
<i>Position verändern</i>	258
<i>schließen</i>	245
<i>schreiben</i>	245
<i>speichern</i>	243
<i>Typ</i>	243
<i>umbenennen</i>	268, 453
<i>verschieben</i>	453
<i>verwalten</i>	267
<i>Zugriffszeitpunkt</i>	267
Dateiendung	
<i>.csv</i>	252
<i>.py</i>	29
Dateizugriff	
<i>binärer</i>	243
<i>sequentieller</i>	243
<i>wahlfreier</i>	243
Datenbank	319
<i>erzeugen</i>	320, 337
<i>im Internet</i>	334
<i>Verbindung herstellen</i>	322
<i>verwalten</i>	337
Datenbankmanagement-System	320, 336
Datenbankserver	337
Datensatz	319
<i>ändern</i>	331
<i>anlegen</i>	320, 322, 340
<i>Anzahl begrenzen</i>	343
<i>anzeigen</i>	323, 341
<i>auswählen</i>	324
<i>Cursor</i>	322
<i>sortieren</i>	328
Datentyp	
<i>bool</i>	131
<i>date (SQL)</i>	337
<i>Datenbank</i>	319
<i>deque</i>	226
<i>double (SQL)</i>	337
<i>ermitteln</i>	91
<i>float</i>	87
<i>float (SQL)</i>	320
<i>int</i>	85
<i>int (SQL)</i>	336
Datentyp (Forts.)	
<i>integer (SQL)</i>	320
<i>long</i>	421
<i>NoneType</i>	134
<i>spezieller</i>	226
<i>str</i>	97
<i>text (SQL)</i>	320
<i>varchar (SQL)</i>	336
Datum und Zeit	215
<i>Differenz</i>	220
<i>erzeugen</i>	219
<i>formatieren</i>	215, 217
<i>Nullpunkt</i>	215
Debian	425
Debug Control	166
Debug Off	169
Debug On	166
Debugger	165
decode()	
<i>bytes</i>	109
Decoding	30, 297
deepcopy()	139, 203
def	74
<i>Klassenmethode</i>	196
del	112, 122, 138, 199
delete (SQL)	333
denominator	95
deque	226
deque()	228
Deserialisierung	261
destroy()	349
Destruktor	198
Dezimalformat	154
Dezimalzahl	85
Dezimalzeichen	24
<i>ändern</i>	283
Dialogfeld	
<i>eigenes</i>	411
<i>modales</i>	414
Dictionary	120
<i>Element löschen</i>	122
<i>Elementexistenz</i>	122
<i>erzeugen</i>	120
<i>Funktionen</i>	121
<i>KeyError</i>	271
<i>leeres</i>	131
<i>Reihenfolge</i>	120
<i>vergleichen</i>	124
View	122

Differenzmenge .....	129
discard() .....	127
DMG-Datei .....	21, 451
double (SQL) .....	337
Double-ended Queue .....	226
DoubleVar .....	369
Duale Zahl .....	85
Dualformat .....	154
Dualzahl .....	93
dump() .....	261

**E**

e	
<i>Eulersche Zahl</i> .....	90
<i>Exponentialschreibweise</i> .....	87
<i>Format</i> .....	152
Editor	
<i>Raspberry Pi</i> .....	429
Eigenschaft .....	196
eindeutiger Index .....	322
Eindeutiger Index (Datenbank) .....	320, 332
Eingabe	
<i>kommentierte</i> .....	144
<i>nicht sinnvolle</i> .....	171
<i>versteckte (GUI)</i> .....	356
<i>versteckte (HMTL)</i> .....	303
<i>wiederholen</i> .....	70
<i>Zeichenkette</i> .....	39, 98
Eingabeaufforderung .....	32
Eingabefeld	
<i>einzeiliges (GUI)</i> .....	354
<i>HTML</i> .....	304
<i>mehrzeiliges (GUI)</i> .....	358
<i>scrollbares (GUI)</i> .....	360
Eingegebene Funktion .....	183
Einplatinencomputer .....	425
Einrücken .....	63
<i>doppelt</i> .....	56
Einzelschrittverfahren .....	166
Elektronische Schaltung .....	425, 430
Elektronisches Bauteil .....	431
Elektrostatik .....	427
elif .....	47
else .....	45, 48
Elternelement (GUI) .....	381
enable() .....	299
Encoding .....	30, 297
end .....	148

Endlosschleife .....	131, 347
endswith() .....	104
Entry-Widget .....	354
Entwicklung .....	65
Entwicklungsumgebung .....	21
enum .....	209
Enumeration .....	209
Eszett .....	30, 297
EVA (Eingabe, Verarbeitung, Ausgabe) .....	40
eval() .....	147
Excel .....	253, 280
except .....	68, 170
exclusives oder	
<i>bitweise</i> .....	93
exec() .....	147
execute() .....	322
<i>cursor</i> .....	338
exists()	
<i>os.path</i> .....	268
exit .....	32
exit()	
<i>sys</i> .....	245
Exponentialformat .....	152
extend() .....	228
extendleft() .....	228

**F**

f(Format) .....	152
Falsch .....	131
False .....	131
Farbcode	
<i>Widerstand</i> .....	433
Fehler .....	66, 163
<i>abfangen</i> .....	67, 69
<i>erzeugen</i> .....	170
<i>Laufzeitfehler</i> .....	165
<i>logischer</i> .....	165
<i>Nachricht (GUI)</i> .....	407
<i>Syntaxfehler</i> .....	163
<i>Unterscheidung</i> .....	171
Feld (Datenbank) .....	319
Fenster	
<i>öffnen</i> .....	348
<i>schließen</i> .....	349
fetchall()	
<i>cursor</i> .....	342
fg .....	352

FieldStorage	
<i>cgi</i>	300
<i>value</i>	300
filter()	160
find()	103
findall()	235
Fließkommazahl	87
float	87
float (SQL)	320
float()	40, 105
font	351
for	54
<i>Dictionary</i>	124
<i>Liste</i>	112
<i>Set</i>	126
<i>Tupel</i>	117
<i>Zeichenkette</i>	101
form (HTML)	290
format()	151, 257
Fraction()	95
fractions	
<i>Fraction()</i>	95
<i>gcd()</i>	95
<i>Modul</i>	94
Frame-Widget	381
from	191
Frozenset	130
frozenset()	130
Funktion	73
<i>als Parameter</i>	182
<i>Aufruf</i>	75
<i>beenden</i>	78
<i>benannter Parameter</i>	174
<i>Definition</i>	74
<i>eingebaute</i>	183
<i>mehrere Rückgabewerte</i>	176
<i>Mehrfachaufruf</i>	158
<i>Name</i>	75
<i>Parameter, ein</i>	75
<i>Parameter, mehrere</i>	77
<i>Parameterübergabe</i>	177
<i>Parametervoreinstellung</i>	175
<i>Rückgabewert</i>	77
<i>statistisch</i>	188
<i>trigonometrische</i>	90
<i>überspringen</i>	169
<i>variable Parameteranzahl</i>	173
<i>verlassen</i>	169

## G

Ganze Zahl	85
Ganzzahldivision	25
<i>Rest</i>	25
Ganzzahligkeit ermitteln	91
gcd()	95
General Purpose Input Output	434
Geometriemanager (GUI)	349, 381, 388, 390
geometry()	406
get()	356, 363, 369
get_ident()	230
GET-Methode	290
Gimp-Toolkit	347
git	444
Gleich	
<i>in Python</i>	44
<i>in SQL</i>	326
glob	265
glob()	265
global	180
GPIO	427, 431, 434
<i>Ausgang</i>	437
<i>Ein- und Ausgang</i>	435
<i>Eingang</i>	439
<i>GND</i>	435
<i>konfigurieren</i>	437
<i>Pin 1</i>	434
<i>Spannungsversorgung</i>	435
Graphical User Interface	347
grid()	388
Groß- und Kleinschreibung	27
Größter gemeinsamer Teiler	95
GTK+	347
GUI-Anwendung	347
<i>starten</i>	349

## H

Halbleiter	436
Haltepunkt	169
has_key()	423
HDMI	426
Header	297
height	351
hex()	86
Hexadezimale Zahl	85
Hexadezimalformat	154
HIGH	431

Hintergrundfarbe (GUI) .....	352
Höhe (GUI) .....	351
htdocs .....	450
HTML	
<i>Datei kopieren</i> .....	289
<i>Datei lesen</i> .....	286
<i>Formular</i> .....	298
<i>Formulardaten</i> .....	300
<i>Formularelement</i> .....	301
<i>lernen</i> .....	286
<b>I</b>	
I (Strom) .....	430
I2C .....	435
<i>Informationen</i> .....	444
<i>nutzen</i> .....	442
i2c-bcm2708 .....	443
i2cdetect .....	444
i2c-dev .....	443
i2c-tools .....	444
Identität .....	136
IDLE .....	21, 23, 165
if .....	45
image .....	352
import .....	43, 190
in	
<i>Dictionary</i> .....	122
<i>Element-View</i> .....	124
<i>for</i> .....	54
<i>Key-View</i> .....	124
<i>Liste</i> .....	111
<i>Sequenz</i> .....	98
<i>Set</i> .....	126
<i>Werte-View</i> .....	123
Index einer Sequenz .....	100
index() .....	115
Info-Nachricht (GUI) .....	407
input (HTML) .....	290
input() .....	144
<i>RPi.GPIO</i> .....	440
insert into (SQL) .....	322
insert() .....	115, 359, 362
Instanz .....	197
int .....	85
int (SQL) .....	336
int() .....	40, 88, 105
integer (SQL) .....	320
IntEnum .....	209
Inter-Integrated Circuit .....	435
Internet .....	285
<i>Daten lesen</i> .....	286
<i>Daten senden</i> .....	289, 293
<i>Datenbank im</i> .....	334
Interpreter .....	18
IntVar .....	369
Inversion	
<i>bitweise</i> .....	93
is .....	136, 204
items() .....	124
Iterable .....	157
<i>filtern</i> .....	160
<i>verbinden</i> .....	157
Iterierbares Objekt .....	157
<b>J</b>	
Ja-Nein-Box (GUI) .....	407
<b>K</b>	
Kalenderwoche .....	217
Kathode .....	436
Kernel-Modul .....	443
<i>auflisten</i> .....	443
<i>Blacklist</i> .....	444
KeyError	
<i>Dictionary</i> .....	271
keys() .....	124
Klammersetzung .....	25
Klasse .....	195
<i>abgeleitete</i> .....	205
<i>Basisklasse</i> .....	205
<i>Definition</i> .....	196
Kommandozeile .....	31, 192, 451
Kommentar .....	33
Konstanten	
<i>Aufzählung</i> .....	209
Konstruktor .....	198
Kontrollkästchen	
<i>HTML</i> .....	303
Kontrollstruktur, geschachtelte .....	55
Konvertierung .....	423
Kopfrechnen .....	37

**L**

Label-Widget .....	351
Lambda .....	181
Lambda-Funktion .....	387, 389
Lange Ausgabe .....	34
LAN-Kabel .....	426
Laufzeitfehler .....	165
Layoutmanager (GUI) .....	381
Leafpad .....	429
LED .....	436
<i>anschließen</i> .....	436
<i>blinken</i> .....	437
<i>ein- und ausschalten</i> .....	437
<i>morse</i> .....	438
<i>Vorwiderstand</i> .....	436
len() .....	100
Leuchtdiode .....	436
LibreOffice Calc .....	254, 280
Light Emitting Diode .....	436
like (SQL) .....	326
limit .....	343
limit_denominator() .....	95
Linksbündig .....	154
Linux .....	18, 21
<i>Terminal</i> .....	192
List Comprehension .....	161
Listbox-Widget .....	361, 363
Liste .....	109
<i>Anzahl bestimmter Elemente</i> .....	115
<i>eingebettete</i> .....	111
<i>Element anfügen</i> .....	115
<i>Element einfügen</i> .....	115
<i>Element löschen</i> .....	112, 115
<i>filtern</i> .....	161
<i>leere</i> .....	131
<i>mehrdimensionale</i> .....	110
<i>Operation</i> .....	112
<i>Position ermitteln</i> .....	115
<i>sortieren</i> .....	115
<i>umdrehen</i> .....	115
load() .....	263
localhost .....	449
localtime() .....	215
Logischer Fehler .....	165
Logischer Operator .....	48
Lokaler Namensraum .....	180
Lokaler Webserver .....	285, 449
long .....	421

Lösungen .....	20, 454
LOW .....	431
ls -l .....	452
lsmod .....	443
Luftdruck .....	442
Lufttemperatur .....	442
LXTerminal .....	429

**M**

Mac OS X .....	18, 21
mainloop() .....	349
manager-osx .....	451
map() .....	158
math	
<i>cos()</i> .....	90
<i>e</i> .....	90
<i>Modul</i> .....	90
<i>pi</i> .....	90
<i>sin()</i> .....	90
<i>tan()</i> .....	90
Mausereignis (GUI) .....	375
max() .....	185
Maximum .....	185
mean()	
<i>statistics</i> .....	189
Median .....	189
median()	
<i>statistics</i> .....	189
median_high() .....	189
<i>statistics</i> .....	189
median_low() .....	189
<i>statistics</i> .....	189
Menge .....	125
<i>Differenzmenge</i> .....	129
<i>Element hinzufügen</i> .....	127
<i>Element löschen</i> .....	127
<i>kopieren</i> .....	127
<i>leere</i> .....	131
<i>leeren</i> .....	127
<i>Schnittmenge</i> .....	129
<i>Teilmenge</i> .....	128
<i>Vereinigungsmenge</i> .....	129
Menü	
<i>tkinter</i> .....	399
Menü	
<i>Debug</i> .....	166
<i>File</i> .....	28
<i>Run</i> .....	30

Menü (GUI) .....	399	Modul (Forts.)	
<code>abtrennen</code> .....	402	<code>umbenennen</code> .....	191
Menüleiste (GUI) .....	398	<code>urllib</code> .....	285
messagebox .....	407	<code>urllib.parse</code> .....	285
Methode .....	196	<code>urllib.request</code> .....	285
<code>besondere</code> .....	200	<code>webbrowser</code> .....	309
Microsoft Vista .....	450	<code>winsound</code> .....	241
Midori .....	430	Modularisierung .....	73, 190
min() .....	185	Monatsname .....	217
Minimum .....	185	Monitor .....	426
Mittelwert .....	189	Morsezeichen .....	269, 438
mkdir .....	452	MPKG-Datei .....	21
mktimes() .....	219	multiple (HTML) .....	306
mod_python .....	296	Multithreading .....	228
mod_wsgi .....	296	mv .....	453
Modul .....		MySQL .....	336
<code>_thread</code> .....	229	<code>mysql</code> .....	
<code>Adafruit_BMPO85</code> .....	443	<code>connector</code> .....	338
<code>cgi</code> .....	296, 299		
<code>cgitb</code> .....	299		
<code>collections</code> .....	226		
<code>copy</code> .....	139, 203	N	
<code>eigenes</code> .....	190		
<code>einbinden</code> .....	43, 190	Nachkommastellen .....	27
<code>enum</code> .....	209	<code>Anzahl</code> .....	151
<code>erzeugen</code> .....	190	Nachrichtenbox (GUI) .....	407
<code>fractions</code> .....	94	Namensraum .....	180
<code>glob</code> .....	265	<code>lokaler</code> .....	180
<code>importieren</code> .....	190	NC (Taster) .....	439
<code>importieren als</code> .....	191	Nenner eines Bruchs .....	95
<code>math</code> .....	90	New Out Of Box Software .....	428
<code>mysql.connector</code> .....	338	nicht .....	
<code>os</code> .....	266, 267	<code>logisches (Python)</code> .....	48
<code>os.path</code> .....	268	<code>logisches (SQL)</code> .....	326
<code>pickle</code> .....	261	Nichts .....	134
<code>random</code> .....	43	NO (Taster) .....	439
<code>re</code> .....	235	None .....	134
<code>RPi.GPIO</code> .....	436	NoneType .....	134
<code>ScrolledText</code> .....	361	NOOBS .....	428
<code>shutil</code> .....	267	not .....	
<code>sqlite3</code> .....	320	<code>logisch</code> .....	48
<code>statistics</code> .....	188	not (SQL) .....	326
<code>sys</code> .....	192, 245	numerator .....	95
<code>thread</code> .....	230		
<code>time</code> .....	215		
<code>Tkinter</code> .....	349	O	
<code>tkinter</code> .....	347		
<code>tkinter.messagebox</code> .....	407	o (Format) .....	154
<code>tkinter.scrolledtext</code> .....	361	Objekt .....	85, 196

Objekt (Forts.)	
<i>Identität</i>	204
<i>in Datei schreiben</i>	261
<i>Informationen</i>	200
<i>iterierbares</i>	157
<i>kopieren</i>	139, 203
<i>Referenz</i>	136, 202
<i>Typ ermitteln</i>	91
<i>vergleichen</i>	202
Objektorientierte Programmierung	195
oct()	86
oder	
<i>bitweise</i>	93
<i>logisches (Python)</i>	48
<i>logisches (SQL)</i>	326
offvalue	372
Ohm	430
Ohmsches Gesetz	430
OK-Abbrechen-Box (GUI)	407
Oktale Zahl	85
Oktalformat	154
Online-Bestellung	303
onvalue	372
OOP	195
open()	
<i>Datei</i>	244
<i>webbrowser</i>	309
Operator	
<i>Bit-</i>	91
<i>für Sequenzen</i>	98
<i>logischer</i>	48
<i>mehrere Vergleichsoperatoren</i>	50
<i>Rangfolge</i>	26, 52
<i>Rechenoperator</i>	88, 202
<i>Vergleichsoperator</i>	44, 202
<i>Verkettung</i>	99
<i>Vervielfachung</i>	99
Operatormethode	201
opt/lampp	450
Optionsfeld (GUI)	367
or	
<i>logisch</i>	48
or (SQL)	326
ord()	186
order by (SQL)	328
os	
<i>Modul</i>	266, 267
<i>remove()</i>	268
<i>rename()</i>	268
os (Forts.)	
<i>stat()</i>	266
OS X	18, 21
os.path	
<i>exists()</i>	268
Out	169
output()	
<i>RPi.GPIO</i>	437
Over	169
P	
pack()	349
Parallele Verarbeitung	228
Parameter	75, 77
<i>benannter</i>	174, 422
<i>variable Anzahl</i>	173, 422
<i>Voreinstellung</i>	175
partition()	104
pass	145
password (HTML)	306
Passworteingabe	
<i>GUI</i>	356
<i>HTML</i>	303
Peripherie	425
Photolmage	354
PHP	285, 289, 449
phpMyAdmin	337
pi	
<i>Benutzername</i>	429
<i>math. Konstante</i>	90
Pi Store	430
pickle	
<i>dump()</i>	261
<i>load()</i>	263
place()	390, 391, 393, 395
PlaySound()	
<i>winsound</i>	241
pop()	228
popleft()	228
Position Teiltext	103
POST-Methode	294
Primärschlüssel	322
primary key (SQL)	322, 332
print()	29, 148
<i>mehrere Ausgaben</i>	34
Programm	
<i>abbrechen</i>	169, 245
<i>Abbruch vermeiden</i>	67

Programm (Forts.)	
anhalten	222
ausführen in IDLE	30
ausführen von Kommandozeile	31
eingeben	28
konvertieren	423
Parameter	192
Programmentwicklung	65
Programmzeile, lange	143
pyGTK	347
PyQt	347
Python	
auf dem Webserver	297
beide Versionen	21
installieren	20
Interpreter	31, 297, 349
MySQL-Schnittstelle	337
Shell	23
starten	23
Python 2	18, 28
Python 3	18, 421
Python Interpreter	425
python -V	32
python.exe	297
python.org	20, 21
python-smbus	444
<b>Q</b>	
QT	347
Quit	169
<b>R</b>	
R (Widerstand)	430
RAD	17
radio (HTML)	306
Radiobutton	
GUI	367
HTML	303
Rahmen (GUI)	381
raise	170
Randart (GUI)	352
Randbreite (GUI)	352
randint()	43
random	
Modul	43
randint()	43
seed()	43
range()	58
Rapid Application Development	17
Raspberry Pi	
Benutzeroberfläche	429
Browser	430
CPU-Informationen	434
Ein-/Aus-Schalter	428
Festplatte	426
Gehäuse	426
Haftung	428
herunterfahren	429
Maus	426
Modell	426
Neustart	429
Revision	426
Schäden	427
Sensor	442
Sicherheit	427
Tastatur	426
Update	429
WLAN	430
raspberrypi.org	425
Raspbian	
installieren	429
raw_input()	421
re	
findall()	235
Modul	235
sub()	239
read()	247
readline()	247
readlines()	247
reboot	429
Rechenregel	24
Rechte	
UNIX	429
Unix	452
Rechtsbündig	154
Referenz	136, 202
erzeugen	351
löschern	138
übergeben	177
Regulärer Ausdruck	234
relief	352
remove()	115
os	268
rename()	268
replace()	103
repr()	200, 211

Reset-Button (HTML) .....	304
return .....	78
<i>mehrere Werte</i> .....	177
reverse() .....	115
reversed() .....	187
rfind() .....	103
rm .....	454
rmdir .....	453
rotate() .....	228
round() .....	88
rpartition() .....	104
RPi.GPIO .....	436
Rückgabewert .....	77, 134
<i>mehrere</i> .....	176
Run Module .....	30, 166
Runden .....	88
RuntimeError .....	172
rwx .....	452
<b>S</b>	
Scale-Widget .....	373
Schaltfläche (GUI) .....	349
Schaltung	
<i>elektronisch</i> .....	425, 430
Schieberegler (GUI) .....	373
Schleife .....	53
<i>Abbruch</i> .....	54
<i>Durchlauf abbrechen</i> .....	70
<i>for</i> .....	54
<i>while</i> .....	62
Schlüssel-Wert-Kombination .....	120
Schnittmenge .....	129
Schnittstelle	
<i>seriell</i> .....	435
Schrift (GUI) .....	351
Scrollbar (GUI) .....	365
Scrollbar .....	366
Scrollbar-Widget .....	365
ScrolledText .....	361
ScrolledText() .....	361
ScrolledText-Widget .....	360
Scrollen (GUI) .....	360
SD-Karte .....	426
<i>formatieren</i> .....	428
seed() .....	43
seek() .....	258
select (HTML) .....	306
select (SQL) .....	323
selected (HTML) .....	306
self .....	196
Selfhtml .....	286
Sendebutton (HTML) .....	304
sep .....	148
Separator .....	148
Sequenz .....	97
<i>Index</i> .....	100
<i>Operatoren</i> .....	98
<i>Teilbereich</i> .....	99
<i>umdrehen</i> .....	187
Serial Peripheral Interface .....	436
Serialisierung .....	261, 274
Serielle Schnittstelle .....	435
Set .....	125
<i>Reihenfolge</i> .....	125
<i>unveränderliches</i> .....	130
set() .....	126
<i>GUI</i> .....	366, 369
Set-Literal .....	422
setmode()	
<i>RPi.GPIO</i> .....	437
setup()	
<i>RPi.GPIO</i> .....	437
setwarnings()	
<i>RPi.GPIO</i> .....	437
show .....	356
show() .....	407
showerror() .....	407
showinfo() .....	407
showwarning() .....	407
shutdown .....	429
shutil	
<i>copyfile()</i> .....	268
<i>Modul</i> .....	267
sin() .....	90
sleep()	
<i>time</i> .....	437
Slice .....	99
SND_ALIAS	
<i>winsound</i> .....	241
SND_FILENAME	
<i>winsound</i> .....	241
sort() .....	115
sorted() .....	187
Sortieren .....	187
Spannung .....	430
Spannungsquelle .....	430
Spannungsversorgung .....	431

SPI .....	435
Spiel Kopfrechnen .....	37
split() .....	105
SQL .....	319
SQL-Befehl senden .....	322
SQLite3 .....	444
sqlite3	
<i>connect()</i> .....	322
<i>connection</i> .....	322
<i>cursor</i> .....	322
<i>Modul</i> .....	320
start_new_thread() .....	229
startswith() .....	104
startx .....	429
stat() .....	266
statistics .....	188
<i>mean()</i> .....	189
<i>median()</i> .....	189
<i>median_high()</i> .....	189
<i>median_low()</i> .....	189
Statistik .....	188
Steckbrett .....	432
Steckbrücke .....	432
Steckernetzteil .....	426
Step .....	167
str	
<i>count()</i> .....	103
<i>Datentyp</i> .....	97
<i>endswith()</i> .....	104
<i>find()</i> .....	103
<i>partition()</i> .....	104
<i>replace()</i> .....	103
<i>rfind()</i> .....	103
<i>rpartition()</i> .....	104
<i>split()</i> .....	105
<i>startswith()</i> .....	104
str() .....	107
strftime() .....	217
String .....	97
<i>leerer</i> .....	131
StringVar .....	369
Strom .....	430
Stromfluss	
<i>Richtung</i> .....	430
Stromkreis .....	430
<i>schließen</i> .....	439
sub() .....	239
Suchen und Ersetzen .....	234
sudo .....	429
sum() .....	185
Summe .....	185
Syntaxfehler .....	163
sys .....	192, 245
Systemton .....	241, 411
T	
Tabelle (Datenbank) .....	319
<i>anlegen</i> .....	320, 322, 338
tan() .....	90
Taschenrechner .....	23
Tastaturereignis (GUI) .....	375
Taste F5 .....	30, 166
Taster .....	439
<i>prüfen</i> .....	439
Teilbereich einer Sequenz .....	99
Teilmenge .....	128
Teiltext ersetzen .....	103
Temperaturkoeffizient	
<i>Widerstand</i> .....	434
Terminal .....	32, 451
<i>Raspberry Pi</i> .....	429
text (SQL) .....	320
text/html .....	297
Text-Area (HTML) .....	306
Text-Widget .....	358
Thread .....	228
<i>Ausnahme</i> .....	233
<i>globale Daten</i> .....	232
<i>Identifikation</i> .....	230
thread .....	230
time .....	215
<i>localtime()</i> .....	215
<i>mktime()</i> .....	219
<i>sleep()</i> .....	222, 437
<i>strftime()</i> .....	217
<i>time()</i> .....	215
time() .....	215
Tk .....	347
<i>mainloop</i> .....	349
<i>tkinter</i> .....	348
Tkinter .....	349
tkinter .....	347
<i>Button</i> .....	349
<i>Menu</i> .....	399
<i>messagebox</i> .....	407
<i>PhotoImage</i> .....	354
<i>Scrollbar</i> .....	366

tkinter (Forts.)	
<i>Tk</i>	348
tkinter.messagebox	
<i>askokcancel()</i>	407
<i>askretrycancel()</i>	407
<i>askyesno()</i>	407
<i>Modul</i>	407
<i>show()</i>	407
<i>showerror()</i>	407
<i>showinfo()</i>	407
<i>showwarning()</i>	407
tkinter.scrolledtext	
<i>ScrolledText()</i>	361
tkMessageBox	411
Tonsignal	273
Toplevel-Widget	411
touch	429
True	131
try	68, 170
Tupel	115
<i>entpacken</i>	119
<i>leeres</i>	131
<i>mehrdimensionales</i>	117
<i>verpacken</i>	119
type()	91
<b>U</b>	
U (Spannung)	430
UART	435
Ubuntu Linux	18, 21
Umlaut	30, 297
Umrechnung	
<i>in dual</i>	86
<i>in hexadezimal</i>	86
<i>in oktal</i>	86
Umwandlung	
<i>in ganze Zahl</i>	88
<i>in Zahl</i>	39, 105
<i>in Zeichenkette</i>	107
und	
<i>bitweise</i>	93
<i>logisches (Python)</i>	48
<i>logisches (SQL)</i>	326
Ungleich	
<i>Python</i>	44
<i>SQL</i>	326
Unicode	186
<i>Literal</i>	422
unique (SQL)	332
Universal Asynchronous Receiver	
Transmitter	436
UNIX	
<i>Befehle</i>	451
<i>neuer Zeitstempel</i>	429
update	429
update (SQL)	331
update()	122
urlencode()	295
urllib	285
urllib.parse	
<i>Modul</i>	285
<i>urlencode()</i>	295
urllib.request	
<i>Modul</i>	285
<i>urlopen()</i>	287
<i>urlretrieve()</i>	289
urlopen()	287
urlretrieve()	289
USB	426
<b>V</b>	
V (Volt)	430
value, FieldStorage	300
ValueError	67
values()	123, 189
varchar (SQL)	336
Variable	26, 38
<i>kontrollieren</i>	167
<i>Name</i>	27
Verarbeitung, parallele	228
Vereinigungsmenge	129
Vererbung	204
<i>Mehrfachvererbung</i>	207
Vergleichsoperator	44
Versionsnummer	32, 33
Verzeichnis	
<i>aktuell</i>	452
<i>anlegen</i>	452
<i>Hierarchie</i>	452
<i>Inhalt</i>	452
<i>löschen</i>	453
<i>übergeordnet</i>	452
<i>verwalten</i>	267
<i>wechseln</i>	452
Verzweigung	44
<i>einfache</i>	44

## Verzweigung (Forts.)

<i>mehrfache</i>	47
View	122
Volt	430
Vordergrundfarbe (GUI)	352
Vorzeichen	53

**W**

Wahr	131
Wahrheitswert	131
Wahrheitswerttabelle	48
Warnung (GUI)	407
WAV-Datei	241
Webbrowser	286
webbrowser	
<i>Modul</i>	309
<i>open()</i>	309
Webserver	285, 449
<i>lokaler</i>	285
<i>Programmierung</i>	296
while	62
while(True)	131
Widerstand	430, 432
<i>Farocode</i>	433
<i>Temperaturkoeffizient</i>	434
Widget	347
<i>add_cascade()</i>	403
<i>add_checkbutton()</i>	403
<i>add_command()</i>	402
<i>add_radiobutton()</i>	402
<i>add_separator()</i>	402
<i>anchor</i>	352
<i>anordnen</i>	381, 388, 390, 391
<i>Bewegung</i>	393, 395
<i>bg</i>	352
<i>bind()</i>	378, 380
<i>borderwidth</i>	352
<i>command</i>	349, 369, 372
<i>configure()</i>	351
<i>destroy()</i>	349
<i>DoubleVar</i>	369
<i>Eigenschaften ändern</i>	349
<i>Ereignis</i>	378
<i>fg</i>	352
<i>font</i>	351
<i>geometry()</i>	406
<i>get()</i>	356, 363, 369
<i>grid()</i>	388

## Widget (Forts.)

<i>Größe, Ort</i>	406
<i>height</i>	351
<i>image</i>	352
<i>insert()</i>	359, 362
<i>IntVar</i>	369
<i>offvalue</i>	372
<i>onvalue</i>	372
<i>pack()</i>	349
<i>place()</i>	390, 391, 393, 395
<i>relief</i>	352
<i>scrollbars</i>	365
<i>set()</i>	366, 369
<i>show</i>	356
<i>StringVar</i>	369
<i>Toplevel</i>	411
<i>Variable</i>	367
<i>width</i>	351
<i>yview()</i>	366
<i>width</i>	351
Wiederholen-Abbrechen-Box (GUI)	407
Wiederholung	53
Windows	18
Windows 8	20
winsound	241
<i>Beep()</i>	241, 273
<i>SND_ALIAS</i>	241
<i>SND_FILENAME</i>	241
WLAN	426
<i>konfigurieren</i>	430
Wochentagsname	217
WPA-TKIP	430
write()	245
writelines()	245
wxPython	347
wxWidgets	347

**X**

x (Format)	154
XAMPP	285
<i>Control Panel</i>	449
<i>für OS X</i>	451
<i>für Ubuntu Linux</i>	450
<i>für Windows</i>	449

**Y**

---

yview() ..... 366

**Z**

---

Zahl ..... 85  
  *als Bruch annähern* ..... 95  
  *ganze* ..... 85  
  *mit Nachkommastellen* ..... 87  
Zahlensystem ..... 85  
Zähler eines Bruchs ..... 95  
Zeichenkette ..... 97  
  *beginnt mit* ..... 104  
  *Eingabe* ..... 98  
  *endet mit* ..... 104  
  *mehrzeilige* ..... 98

**Zeichenkette (Forts.)**

*partitionieren* ..... 104  
  *zerlegen* ..... 105  
Zeilenende ..... 148  
  *Linux* ..... 259  
Zeilenumbruch in Code ..... 143  
Zeitzone ..... 217  
Zentralwert ..... 189  
zip() ..... 157  
Zufallszahl  
  *Generator* ..... 42  
  *Generator initialisieren* ..... 43  
  *liefern* ..... 43  
Zuweisung ..... 27, 38  
  *kombinierte Operatoren* ..... 141  
  *mehrfache* ..... 117



# Die Serviceseiten

Im Folgenden finden Sie Hinweise, wie Sie Kontakt zu uns aufnehmen können.

## Lob und Tadel

Wir hoffen sehr, dass Ihnen dieses Buch gefallen hat. Wenn Sie zufrieden waren, empfehlen Sie das Buch bitte weiter. Wenn Sie meinen, es gebe doch etwas zu verbessern, schreiben Sie direkt an die Lektorin dieses Buches: [anne.scheibe@galileo-press.de](mailto:anne.scheibe@galileo-press.de). Wir freuen uns über jeden Verbesserungsvorschlag, aber über ein Lob freuen wir uns natürlich auch!

Auch auf unserer Webkatalogseite zu diesem Buch haben Sie die Möglichkeit, Ihr Feedback an uns zu senden oder Ihre Leseerfahrung per Facebook, Twitter oder E-Mail mit anderen zu teilen. Folgen Sie einfach diesem Link: <http://www.galileo-press.de/3594>.

## Zusatzmaterialien

Zusatzmaterialien (Beispielcode, Übungsmaterial, Listen usw.) finden Sie in Ihrer Online-Bibliothek sowie auf der Webkatalogseite zu diesem Buch: <http://www.galileo-press.de/3594>. Wenn uns sinnentstellende Tippfehler oder inhaltliche Mängel bekannt werden, stellen wir Ihnen dort auch eine Liste mit Korrekturen zur Verfügung.

## Technische Probleme

Im Falle von technischen Schwierigkeiten mit dem E-Book oder Ihrem E-Book-Konto bei Galileo Press steht Ihnen gerne unser Leserservice zur Verfügung: [ebooks@galileo-press.de](mailto:ebooks@galileo-press.de).

## Über uns und unser Programm

Informationen zu unserem Verlag und weitere Kontaktmöglichkeiten bieten wir Ihnen auf unserer Verlagswebsite <http://www.galileo-press.de>. Dort können Sie sich auch umfassend und aus erster Hand über unser aktuelles Verlagsprogramm informieren und alle unsere Bücher und E-Books schnell und komfortabel bestellen. Alle Buchbestellungen sind für Sie versandkostenfrei.

# Rechtliche Hinweise

In diesem Abschnitt finden Sie die ausführlichen und rechtlich verbindlichen Nutzungsbedingungen für dieses E-Book.

## Copyright-Vermerk

Das vorliegende Werk ist in all seinen Teilen urheberrechtlich geschützt. Alle Nutzungs- und Verwertungsrechte liegen beim Autor und beim Verlag Galileo Press. Insbesondere das Recht der Vervielfältigung und Verbreitung, sei es in gedruckter oder in elektronischer Form.

© Galileo Press, Bonn 2014

## Ihre Rechte als Nutzer

Sie sind berechtigt, dieses E-Book ausschließlich für persönliche Zwecke zu nutzen. Insbesondere sind Sie berechtigt, das E-Book für Ihren eigenen Gebrauch auszudrucken oder eine Kopie herzustellen, sofern Sie diese Kopie auf einem von Ihnen alleine und persönlich genutzten Endgerät speichern. Zu anderen oder weitergehenden Nutzungen und Verwertungen sind Sie nicht berechtigt.

So ist es insbesondere unzulässig, eine elektronische oder gedruckte Kopie an Dritte weiterzugeben. Unzulässig und nicht erlaubt ist des Weiteren, das E-Book im Internet, in Intranets oder auf andere Weise zu verbreiten oder Dritten zur Verfügung zu stellen. Eine öffentliche Wiedergabe oder sonstige Weiterveröffentlichung und jegliche den persönlichen Gebrauch übersteigende Vervielfältigung des E-Books ist ausdrücklich untersagt. Das vorstehend Gesagte gilt nicht nur für das E-Book insgesamt, sondern auch für seine Teile (z.B. Grafiken, Fotos, Tabellen, Textabschnitte).

Urheberrechtsvermerke, Markenzeichen und andere Rechtsvorbehalte dürfen aus dem E-Book nicht entfernt werden, auch nicht das digitale Wasserzeichen.

## Digitales Wasserzeichen

Dieses E-Book-Exemplar ist mit einem **digitalen Wasserzeichen** versehen, einem Vermerk, der kenntlich macht, welche Person dieses Exemplar nutzen darf. Wenn Sie, lieber Leser, diese Person nicht sind, liegt ein Verstoß gegen das Urheberrecht vor, und wir bitten Sie freundlich, das E-Book nicht weiter zu nutzen und uns diesen Verstoß zu melden. Eine kurze E-Mail an [info@galileo-press.de](mailto:info@galileo-press.de) reicht schon. Vielen Dank!

## Markenschutz

Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

## Haftungsausschluss

Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Verlag noch Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen.

# Über den Autor

**Thomas Theis**, Dipl.-Ing. für Technische Informatik, verfügt über langjährige Erfahrung als EDV-Dozent, unter anderem an der Fachhochschule Aachen. Er leitet Schulungen zu C/C++, Visual Basic und Webprogrammierung. <http://www.theisweb.de/>